Generalized Cosine and Similarity Metrics: A Supervised Learning Approach based on Nearest Neighbors

Ali Mustafa Qamar

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Ali Mustafa QAMAR

Thèse dirigée par **Eric GAUSSIER**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Mesures de Similarité et Cosinus Généralisé : Une Approche d'Apprentissage Supervisé Fondée sur les k Plus Proches Voisins

Generalized Cosine and Similarity Metrics: A Supervised Learning Approach based on Nearest Neighbors

Thèse soutenue publiquement le **19 Novembre 2010**,
devant le jury composé de :

Mme Marie-Christine ROUSSET
Professeur, Université de Grenoble (Présidente)
M. Richard NOCK
Professeur, Université des Antilles et de la Guyane (Rapporteur)
M. Marc SEBBAN
Professeur, Université Jean Monnet de Saint Etienne (Rapporteur)
M. Patrick GALLINARI
Professeur, Université Pierre et Marie Curie (Examinateur)
M. Eric GAUSSIER
Professeur, Université de Grenoble (Examinateur)
Mme Mirta GORDON
Directrice de Recherche, CNRS (Examinatrice)

Mis en page avec la classe thloria.

Acknowledgements

First of all, I am highly indepted to my supervisor Professor Eric Gaussier for his guidance, patience, support and help throughout the duration of my thesis. I do not find words to explain how he has stood beside me throughout these three years. Whenever I had any problem, he was always there for my help. I learned a variety of research skills from my advisor.

I warmly thank the official referees Professor Richard Nock and Professor Marc Sebban for being my thesis reviewers and dedicating a lot of time out of their busy schedule. I thank them for their critical reviews and useful suggestions.

My sincere thanks are due to Professor Patrick Gallinari for being my external examiner.

Many thanks to Professor Marie-Christine Rousset for being the President of the jury.

I also thank Higher Education Commission, Pakistan and Dr. Atta-ur-Rehman for selecting me for the MS scholarship. That paved the way for my PhD education funded by the French Ministry of Higher Education and Research.

I thank all of my colleagues for their support and wish them all the best for their future endeavors. My special thanks goes to Vincent Bodinier, Bo Li, Clement Grimal, Cedric Lagnier, Trong-Ton Pham, Ali Aït-Bachir, Andy Tseng, Rami Albatal and Nathalie Denos. I was really lucky to be a part of a dynamic and innovative LIG lab. I equally wish to thank Cecile Amblard, Gilles Bisson, Christophe Brouard, Ahlame Douzal and Mirta Gordon for their guidance.

I also thank the Pakistani students without whom the stay in France would have been much more difficult.

Last but not the least, I want to express my gratefulness to my mother and my father for always supporting me and praying for me despite being at a distance of more than 8000 kms. I also thank my brothers for their help. My very special thanks go for my wife and daughters for being with me during every moment of this Phd thesis and allowing me to work late and even on the weekends.

Dedicated to my parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Preface

Machine learning [73, 12] is the art of designing, developing and evaluating algorithms which are capable of evolving behaviors based on the empirical data. Machine learning algorithms automatically improve their performance P based on some experience E at some task T. As an example, consider the problem of developing a system which learns to play checkers. In this case, the task T is to play checkers, the performance measure P is the percentage of games won in a world tournament and E is the opportunity of play against self.

Machine learning has recently emerged as one of the key areas of artificial intelligence. One of the primary reasons for its popularity lies in the eager wish of humans to explore and replicate the human learning process. Machine learning can be viewed as a two-fold task, consisting of learning the invariant and common properties of a set of samples characterizing a class, and of deciding that a new sample is a possible member of the class by noting that it has properties common to those of the set of samples [78].

Machine learning algorithms can be broadly categorized in three different categories: supervised learning in which case the learning is based on a set of labeled data (also called training data), unsupervised learning which does not require any sort of human intervention and does not have a training phase (it is usually used when the classes are not known in advance), and semi-supervised learning lying in between the supervised and unsupervised settings.

Machine learning has been successfully applied in various different settings like classification (e.g. handwritten digit recognition [63], document classification [55], face recognition [105] etc.), clustering (k-means clustering [11], spectral clustering [115]), bio-informatics, finance, information filtering systems that automatically learn users' interests, detection of hazardous smokes on industrial facilities [39] etc. It is based on learning from data and hence is closely related to the field of data mining. Data mining deals with extracting useful patterns from raw data so as to make it a more useful commodity.

Every machine learning algorithm works with a set of examples. Among this set, some of the examples are used to learn the underlying characteristics of the data based on a set of features. This subset is termed training set. In order to validate the algorithm, the trained or learned algorithm is run on unseen examples, also known as the test set. A validation set can be optionally employed so as to fine tune the different parameters of the algorithm.

1.2 Motivation

We consider two objects e.g. documents or images which need to be compared. In order to do this comparison, similarity or distance can be found between these two objects. Most of the time, default measures, i.e. Euclidean distance in the case of images and cosine similarity for text classification, are employed which consider that the metric between different objects is parametrized by an identity matrix. In other words, measures like Euclidean distance and cosine similarity consider a very simple underlying geometry for the space in which the data lie. Many works have proved that it is far better to learn the metric structure from the data rather than assuming a simple geometric structure.

The recent popularity of Internet has led to an enormous increase in the amount of information

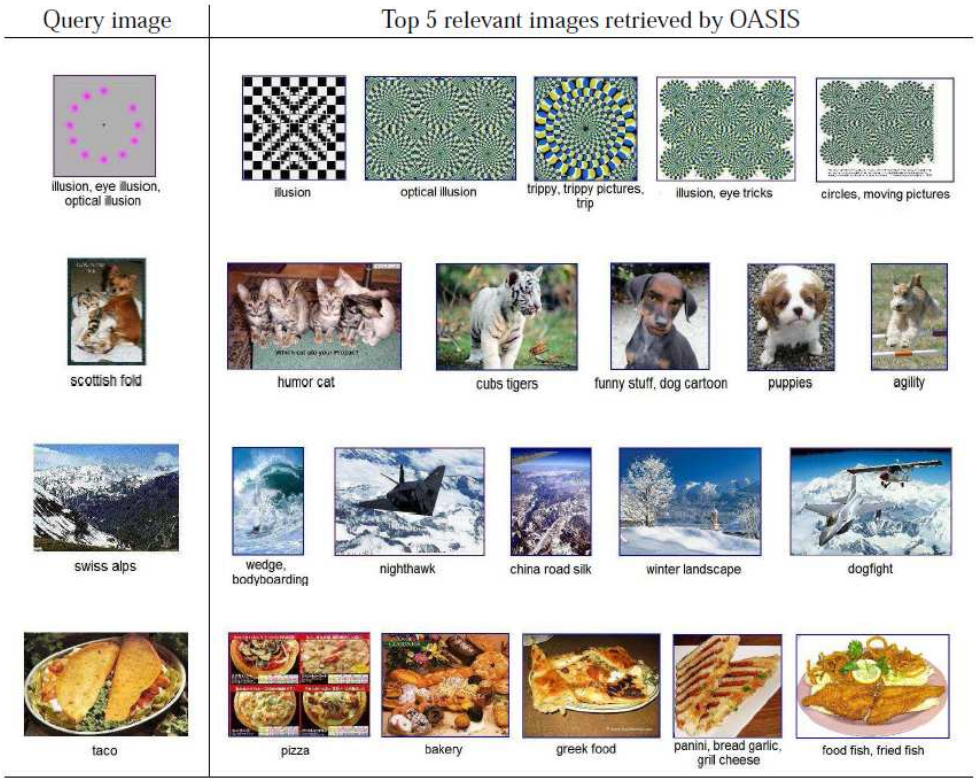

Figure 1.1: OASIS: A distance metric learning algorithm to find similar images [16]

as well as the growth of research areas devoted to automated organization of this information. An evaluation forum named Cross Language Evaluation Forum (CLEF) has been run every year since 2000, with the aim of evaluating information retrieval systems operating on European languages in monolingual as well as cross-language contexts. An information filtering (INFILE) campaign has been run as a pilot track of CLEF in 2008 and 2009. The aim in INFILE was to filter a continuous stream of documents into different predefined topics. In the case of information filtering, cosine based thresholds could be learned based on the incoming stream of documents, provided there is at least some sort of supervision. The Online algorithm was developed in 2008 and was the only participation for INFILE in that year. Furthermore, the batch algorithm got the best F-score during 2009 among different participants. Learning a complete metric is a wiser decision than learning only the thresholds, if one is working in a fully supervised setting. This has given rise to a domain called *metric learning* [54, 53]. Figure B.1 shows the top five images as ranked by *OASIS* [16], an image distance learning algorithm, [1] on four examples of query-images in a Google proprietary dataset. The relevant text queries for each image are written beneath the image. The top most row shows a query-image, originally retrieved in response to the text query *illusion*. It may also be noticed that all of the five images ranked highly by *OASIS* are semantically related, portraying other sorts of visual illusions. The rest of the three examples show that *OASIS* was able to grab the semantics of animal photos (cats and dogs), mountains and different food items.

[1] In this work, no distinction is made between the distance and the similarity.

The primary aim of metric learning is to learn a metric well adapted to the problem under consideration. Algorithms for data classification and clustering rely heavily on the presence of a good metric. Apart from these areas, metric learning is a very important ingredient in problems like face recognition, visual object recognition, automated speech recognition [107], language problems, music similarity, pose estimation, image similarity and search [59] etc. For many metric learning algorithms, both online as well as batch learning is possible. Metric learning can be further subdivided into two different types: distance metric learning and similarity metric learning.

Most of the works related to metric learning concentrate on distance metric learning only and try to learn Mahalanobis distance metric. However, in many practical situations, similarities may be preferred over distances. This is typically the case when one is working on texts, for which the cosine measure has been deemed more appropriate than the standard distance metrics like the Euclidean or the Mahalanobis ones. Furthermore, several experiments show that the use of the cosine similarity should be preferred over the Euclidean distance on several, non textual collections as well (see e.g. [18, 72, 84, 87]). Being able to efficiently learn appropriate similarity measures, as opposed to distances, e.g. for *kNN* classification, is thus of high importance for various collections. If several works have partially addressed this problem (as for example [1, 46, 52]) for different applications, we know of no previous work which has fully addressed it in the context of learning similarity metrics for *kNN* classification. This is the basic motivation behind this work. In the first instance, an unconstrained similarity metric learning algorithm is developed in which case the normalization is completely independent of the similarity matrix. Proofs were developed to show that the error on unseen examples is limited and the algorithm has good generalization properties. This is followed by the development of an algorithm based on generalized cosine having a normalization dependent on the similarity matrix. Moreover the unconstrained similarity learning is compared with the *RELIEF* family of algorithms. Although *RELIEF* is basically a feature re-weighting algorithm, it has been recently proved by Sun and Wu [102] that it is a distance metric learning algorithm which optimizes an approximation of the $0-1$ loss. We show here that this approximation is too loose, and propose a stricter one, better suited for classification.

1.3 Thesis Plan

- We describe in Chapter 2 the basic concepts related to Machine Learning along with the survey of various state of the art techniques for metric learning. The two main types of machine learning, i.e. supervised and unsupervised learning, are discussed in detail. Furthermore, the basics of online as well as batch algorithms are discussed. Some of the key distance metric learning algorithms, e.g. Weinberger's Large Margin Nearest Neighbor (LMNN) [112], Davis's Information Theoretic Metric Learning [28] and Shalev's POLA [99], are discussed and compared thoroughly. Furthermore, similarity as well as similarity based methods are also examined. *RELIEF*, a well known feature reweighting algorithm along with its mathematical interpretation is also presented. Evaluation metrics and the techniques for classifiers' comparison are finally discussed.

- In Chapter 3, we show how cosine based thresholds can be learned effectively when little or

no supervision is present. This technique is established for a filtering task where a huge set of documents is filtered according to user profiles. Online as well as Batch algorithms are discussed and compared extensively. The algorithms are developed as a part of the InFile campaign of the CLEF competition.

- Chapter 4 starts with the description of an unconstrained similarity metric learning method, called *SiLA*, where the normalization is completely independent of the learned similarity matrix. *SiLA* is compared with the *RELIEF* algorithm for which Sun and Wu [102] have shown that it basically learns a distance metric while optimizing a cost function approximating the $0-1$ loss. We show that the approximation used by *RELIEF* is loose, and propose a stricter version using a cost function closer to the $0-1$ loss. This stricter version leads to a new, and better *RELIEF* based algorithm for classification. Furthermore, a generalized cosine similarity learning algorithm (*gCosLA*) is developed, in which case the normalization is dependent on the similarity matrix.

- The different similarity metric learning algorithms developed during the course of this thesis are evaluated in Chapter 5. In order to assess whether the results are significantly different or not, a s-test is used. We show that similarities are a more viable option as compared to the distance metrics on many datasets. Furthermore, the unconstrained similarity metric learning algorithm as well as the generalized cosine similarity one are compared with different state of the art classification algorithms. The similarity learning algorithms outperform their counterparts on some of the UCI datasets.

- Chapter 6 presents the conclusion along with the limitations of the proposed approaches and the future perspectives.

- Finally proofs for convergence and good behavior have been provided for *SiLA* and *gCosLA*.

Chapter 2

State of the Art Approaches to Metric Learning

Contents

2.1 Introduction

Machine learning is basically a process by which an unknown dependency (input, output) of a system is estimated, using a limited number of observations or examples. A typical machine learning system is composed of three components: a generator of random input vectors denoted by x, a system that returns an output y for a given input vector x, and the learning machine which estimates the mapping of the system from the observed samples composed of input and output. This scenario describes many real world problems like classification, regression (e.g. Gaussian processes [92]), clustering etc. The generator produces random vectors $x \in \mathbb{R}^d$ having d dimensions, drawn independently from a fixed but unknown probability density function $p(x)$. The system provides an output value y for every input vector x, based on the fixed but unknown conditional density $p(y|x)$ (probability of observing y given x). The third component of a machine learning system is the *learning machine* which is capable of implementing a set of functions $f(x, \omega)$, $\omega \in \Omega$, where Ω is a set of abstract parameters used to index the set of functions. Here the set can be any set of functions, chosen before the learning has begun. The learning machine must select a function (from a set of functions it supports) which best approximates the system's response. This selection process is based on the observation of a finite number n of examples. The training data, composed of inputs and outputs is independent and identically distributed (i.i.d.) as per the joint probability density function (pdf):

$$p(x, y) = p(x)\, p(y|x)$$

The training data from this distribution can be described as:

$$(x^{(i)}, y^{(i)}),\ (i = 1, \cdots, n)$$

An instance space, $\mathcal{X}$ is defined as a space containing all of the instances i.e. $x^{(1)}, x^{(2)}, \cdots, x^{(n)}$. Similarly, a label set, $\mathcal{Y}$ contains all of the possible labels or classes.

The quality of the learning process is measured using a loss function $L(y, f(x, \omega))$ which represents the discrepancy between the actual output y produced by the system for a given example x, and its approximation $y' = f(x, \omega)$ by the learning machine. In general, the loss is always non-negative, with higher values indicating a poor approximation [19]. In the rest of this chapter, various approaches for metric learning are discussed in detail which constitutes the core of this thesis.

After explaining a typical machine learning system, the next section discusses the fundamental concepts related to machine learning including a comparison between supervised and unsupervised learning, and an insight into the differences between online and batch learning.

2.2 Machine Learning Fundamentals

Some notations are provided here, which will be used throughout the thesis. An input object can be represented as $x \in \mathbb{R}^d$ where $\mathbb{R}$ is the set of real numbers and d denotes the number of features or dimensions. As x is a vector, the features of x can be accessed by the subscripts $x_i, 1 \geq i \geq d$. The output is denoted by y. The vectors are not written in bold and the transpose of x is represented as x^t.

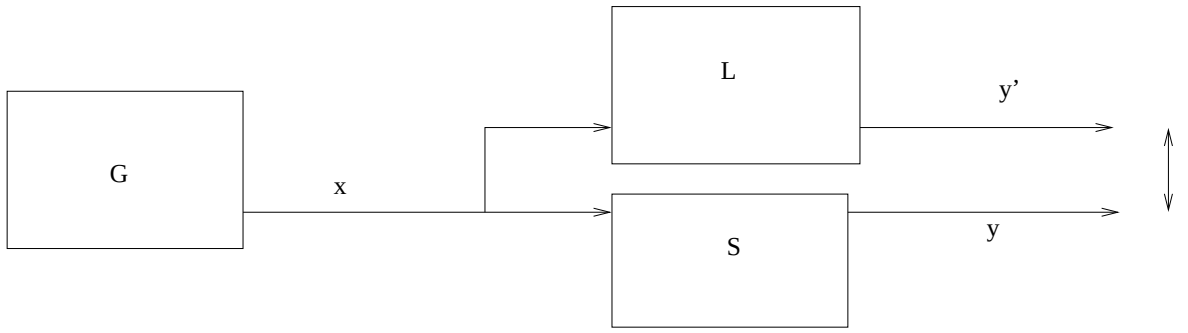

Figure 2.1: A typical machine learning system using observations of the system to predict the outputs

A fundamental hypothesis of statistical learning theory is that all of the examples are generated independently using a probability distribution $\mathbb{P}$. In other words, it can be said that the examples are i.i.d. (independent and identically distributed) as per $\mathbb{P}$.

Another very important concept is that of error, also known as cost or loss. Given a prediction function f, the loss finds the accord between the prediction $f(x)$ and the target output y. In the case of classification, a commonly used loss function is the $0-1$ loss [2], which is either 0 (correct classification) or 1 (wrong classification):

$$L(f(x), y) = \begin{cases} 1 & \text{if } f(x) = y \\ \\ 0 & \text{otherwise} \end{cases}$$

The error in the case of regression is the square of the difference between the actual output and the anticipated one (target output) [81]:

$$L\left(f(x), y\right) = \left(f(x) - y\right)^2$$

With this, the risk associated with the prediction function $f(x)$ can be calculated for all of the examples (x, y). This loss is also known as the *generalization risk* and is defined as the *expectation* of the loss function:

$$\mathbb{R}_{\text{gen}}(f) = E\left[L\left(f(x), y\right)\right] = \int L\left(f(x), y\right) d\mathbb{P}(x, y)$$

where $\mathbb{P}$ represents the probability distribution described earlier. In general, the risk $\mathbb{R}_{\text{gen}}(f)$ cannot be computed since the probability distribution is not known to the learning algorithm. Nevertheless, an approximation for the generalization risk can be calculated by averaging the loss function over the training set. This approximation is termed *empirical risk* and is given by:

$$\mathbb{R}_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} L\left(f(x^{(i)}), y^{(i)}\right)$$

where n stands for the number of examples in the training set.

[2] $0-1$ loss is also known as the leave-one-out error

2.2.1 Supervised vs Unsupervised Learning

Machine learning algorithms could be broadly classified into two main categories: supervised and unsupervised learning algorithms. Supervised learning is based on learning a function from a set of training samples in the form of pairs. Each pair is made up of input objects (usually vectors) and desired output values also known as target values. The function learned can have different types of outputs: continuous values (regression) or a predicted class label for the input object (also referred to as classification). The aim is to predict the value of the function once the learner has encountered a sufficient number of examples (training phase) in order to classify unseen examples (test phase). The accuracy of the learned function strongly depends on the quality of the object representation. The input objects are, oftenly, described in the form of feature vectors. The number of features must be chosen in a way that they can predict the output accurately. Some of the key supervised algorithms include perceptron algorithm, support vector machines (SVM) etc.

On the other hand, a model is fitted to observations (unlabeled examples) in the case of unsupervised learning [3]. In many real world applications, the labels are not present. The unsupervised learning methods work without a teacher as opposed to supervised learning. It does not have a priori output as opposed to supervised learning and helps to learn larger and more complex models than with supervised learning. The reason is that in supervised learning, the aim is to find the connection between two sets of data but the difficulty of the learning task increases exponentially in the number of steps required in finding the relation between the two data sets. On the contrary, unsupervised learning can proceed hierarchically from the observations to more abstract levels of representation. Some examples of unsupervised algorithms are clustering, self-organizing maps (SOM) etc.

Clustering is based on organizing the given examples into different clusters in such a way that the similar examples are put into the same cluster while different examples appear in different clusters. In general, clustering offers a way to know the implicit structure of the dataset.

Apart from the major categorization of the machine learning algorithms (supervised and unsupervised), there is another way in which a machine learning algorithm could be classified: online or batch learning.

2.2.2 Online Learning vs Batch Learning

Learning can be batch or online depending on the targeted task. Batch learning or offline learning deals with all of the available examples in one-go. In general, the learned parameters cannot be updated once the learning is complete. It is assumed that a probability distribution over the product space $\mathcal{X}$x$\mathcal{Y}$ exists, where $\mathcal{X}$ is an instance space while $\mathcal{Y}$ is a label set as explained in section 1.1. Moreover, it is also assumed that there is access to a training set drawn i.i.d. from this distribution. The aim is to generate an *output* hypothesis from the training set. Furthermore, the batch algorithm should have the ability to *generalize* well beyond the training set and accurately predict the labels for unseen test examples sampled from the distribution.

[3] The frontier between supervised and unsupervised learning is blurred: e.g. semi-supervised learning in which case the classifier can be initialized based on the labeled examples which then learns without supervision on the rest of the unlabeled examples, transductive learning etc.

Examples of batch algorithms include linear discriminant analysis (a model employing stochastic dependence between terms that relies on the covariance matrices of different categories), Rocchio classifier etc.

Most of the Machine Learning models are designed for the batch case. However, another type of learning is oftenly used nowadays. It is called online learning (also known as incremental learning, instantaneous learning or on-the-fly learning) and uses the examples one-by-one to learn the parameters for the employed algorithm. In other words, the instances are obtained in a sequential manner. It starts building the classifier once it has examined the very first training example. After receiving an instance, the online algorithm makes a prediction using a default hypothesis h_1, the type of which depends on the problem being treated e.g. in the case of binary classification, it is a $+ive/-ive$ decision [23]. Upon making a prediction ($\hat{y}$), the algorithm receives a feedback in the form of correct prediction (y). Based on the true label, the algorithm can suffer from an *instantaneous* loss. The *cumulative* loss on a sequence of rounds is the sum of instantaneous losses suffered on each of the rounds in the sequence. The *cumulative* loss or the *empirical* loss is the sum of hinge losses for the entire training set. The instance-label pair together enables the online algorithm to modify its prediction mechanism and eventually helps in making accurate predictions over the rest of the instances. An online algorithm is defined by its default hypothesis and the update rule to define new hypothesis. In general, an example is used only once by the online algorithm. However, the algorithm could be run more than once to optimize its performance.

Online learning is usually simpler to implement, memory efficient and faster as compared to the batch learning [30] and is preferred in the environments where the best model changes gradually over the passage of time or when the storage space is limited. Apart from these practical advantages, online algorithms often have formal guarantees in the form of worse case bounds on their performance. Furthermore, sometimes there is a scenario e.g. text or information filtering where the examples are provided in a sequential manner and the predictions must be made on-the-fly.

In case, there is no loss for an online algorithm, the current hypothesis h_τ is left unchanged. On the contrary, if there is some loss, two goals must be balanced:

1. Change the current hypothesis h_τ as it has encountered a certain loss for the current example. However, the change must be enough so that the current error is not repeated in the future.

2. Do not change h_τ too much, since h_τ was able to correctly classify the last encountered example. If the current hypothesis is changed excessively, then one cannot be sure that the new hypothesis would be able to cover the previously seen examples.

Suppose that the changes in h_τ are measured by taking into account the Euclidean distance between the updated hypothesis $h_{\tau+1}$ and the old one h_τ. This case, where the first goal is enforced while the the second one is minimized, corresponds to the classical gradient descent update rule.

In order to satisfy the two major goals (given above) of an online algorithm, a *passive* and *aggressive* strategy is required. It should be *aggressive* enough to avoid the repeat of errors,

while *passive* at the same time so that a new hypothesis classifies correctly the examples already encountered by the algorithm.

The Passive Aggressive Family of Online Algorithms

Crammer et al. [23] have defined a family of online algorithms termed as *passive aggressive* algorithms. The basic idea is the same as that of the goals mentioned earlier. However, instead of simply ensuring that a correct classification is made with the help of rule 1, it is ensured that the correct classification is made with a margin of at least 1. The examples are considered in the form of instance-label pairs i.e. (x^τ, y^τ) where $x^\tau \in \mathbb{R}^n$, $y^\tau \in \{+1, -1\}$ and τ represents the current iteration or round. The predictions are made based on a classification function of the form: $sign(w \,.\, x)$ where $w \in \mathbb{R}^n$ represents the vector of weights. The aim of the algorithm is to learn the vector of weights in an incremental fashion. The margin on the round τ can be given by $y^\tau(w_\tau \,.\, x^\tau)$. In case the margin is positive ($sign(w_\tau \,.\, x^\tau) = y^\tau$), it can be stated that the algorithm has made a correct decision. However, the aim is to predict with higher confidence and to achieve a margin of at least 1 in as many rounds as possible. Whenever the margin is less than 1, the algorithm suffers from a hinge loss which can be given as:

$$l_\tau(w; (x^\tau, y^\tau)) = \begin{cases} 0 & \text{if } y^\tau(w \,.\, x^\tau) \geq 1 \\ \\ 1 - y^\tau(w \,.\, x^\tau) & \text{otherwise} \end{cases}$$

Hence, the loss is zero whenever the margin is greater than 1. On the contrary, the loss is equal to the difference between 1 and the margin value if the margin is less than 1. For regression, the choice of the margin can be defined by the user as well. It has been further shown that the algorithms have a small *cumulative square loss* over the set of T examples ($\sum_{\tau=1}^{T} l_\tau^2$).

The initial weight vector w_1 is initialized with all zeros for all of the variants of the passive aggressive algorithm for binary classification. However, the update rule for the weight vector differs for each of the three variants. The simplest and the strongest of the rules requires the new weight $w_{\tau+1}$ to be the solution of the following constraint optimization problem:

$$w_{\tau+1} = \arg\min_{w \in \mathbb{R}^n} \frac{1}{2}\|w - w_\tau\|^2 \quad \text{subject to} \quad l_\tau(w; (x^\tau, y^\tau)) = 0$$

which has a closed form solution:

$$w_{\tau+1} = w_\tau + \delta_\tau y^\tau x^\tau \text{ where } \delta_\tau = \frac{l_\tau}{\|x^\tau\|^2}$$

Here $\delta_\tau \geq 0$ and is a Lagrange multiplier. Moreover, $w_{\tau+1}$ is the projection of w_τ onto the space where the hinge loss on the current example is zero. Whenever the loss is zero, $w_{\tau+1} = w_\tau$ and the algorithm is said to be *passive*. However, if the loss is positive (it cannot be negative), the algorithm *aggressively* forces the update $w_{\tau+1}$ to satisfy the constraint $l(w_{\tau+1}; (x^\tau, y^\tau)) = 0$ imposed by the current example, while remaining as close as possible to w_τ. That is the reason these algorithms have been termed as *passive aggressive*. The passive approach is for the retention of the information gathered during the earlier iterations while the aggressive nature is useful whenever there is a misclassification.

Another related work is that of Helmbold et al. [50] who showed the relationship between the amount of progress made at each iteration and the amount of information retained from the previous ones. Here, the update requires $w_{\tau+1}$ to correctly classify the current example x^τ with a high margin and in this way, the progress is made (aggressiveness). Similarly $w_{\tau+1}$ should stay close to w_τ which enables the algorithm to retain the information learned from the previous iterations (passiveness).

In order to reduce the *aggressiveness* of *Passive Aggressive* algorithms, two more update rules have been introduced, which employ gentler updates and use a non-negative slack variable ξ to redefine the optimization problem:

$$w_{\tau+1} = \arg\min_{w \in \mathbb{R}^n} \frac{1}{2}\|w - w_\tau\|^2 + C\xi \quad \text{subject to} \quad l(w; (x^\tau, y^\tau)) \leq \xi \ \wedge \ \xi \geq 0$$

Here the objective function is directly proportional to the slack variable ξ and C. C is a positive *aggressiveness* parameter that controls the impact of the slack term on the objective function. More precisely, C controls the trade off between two objectives: remaining close to the previous weights w_τ and minimizing the loss on the current example. It has been shown that the larger values of C indicate a more aggressive update. The resulting algorithm has been termed as *PA-I*. This update is termed *gentler* as it is no longer required that the loss must be equal to zero and in this way, the loss constraint is relaxed.

In another variation (named as *PA-II*), an objective function has been defined which scales quadratically with ξ:

$$w_{\tau+1} = \arg\min_{w \in \mathbb{R}^n} \frac{1}{2}\|w - w_\tau\|^2 + C\xi^2 \quad \text{where} \quad l(w; (x^\tau, y^\tau)) \leq \xi$$

The variants *PA-I* and *PA-II* have the same closed form solution as that of *PA-I* except the value of δ_τ:

$$\delta_\tau = \min\left\{C, \frac{l_\tau}{\|x^\tau\|^2}\right\} \quad \text{(PA-I)}$$

$$\delta_\tau = \frac{l_\tau}{\|x^\tau\|^2 + \frac{1}{2C}} \quad \text{(PA-II)}$$

It is important to mention here that the Passive Aggressive family of algorithms learn only a vector of weights and are not interested in learning a complete matrix.

Dredze et al. [33] have developed confidence-weighted (CW) linear classifiers which also belong to the family of Passive Aggressive algorithms. The main characteristic of these classifiers is that they maintain a probabilistic measure of confidence in each of the attributes. The less confident parameters are updated more *aggressively* than more confident ones. In CW learning methods (Dredze et al. [33], Crammer et al. [24]) second-order information is used to represent the uncertainty about the linear classifier's feature weight estimates. This second-order information could be modeled as a Gaussian distribution over the classifier's weight vector. In these cases, the mean of the weight vector is used for classification, whereas the covariance matrix is used to modulate the learning rate over different features [67]. However, the CW learning methods use diagonal approximations for the full covariance matrix, and hence lose the information regarding cross-feature correlations which can help towards faster convergence. Ma et al. [67] show in which cases it is advantageous to use a full matrix rather than using the diagonal one.

Online to Batch Conversion

Sometimes, a batch algorithm must be developed that not only is computationally efficient and easier to implement than an online algorithm but also has the good generalization properties of batch algorithms. A simple way to develop such an algorithm is to use *online to batch conversion*. Many people have described such conversion e.g. Gallant [40] has developed a *Pocket* algorithm which is basically a conversion of online perceptron algorithm to a batch one. This method retains the longest surviving hypothesis i.e. which has made the fewest number of mistakes during the training phase.

Littlestone et al. [65] have described a *cross-validation* technique where the training set is presented to the online algorithm. After running the algorithm for T rounds, a sequence of hypothesis $h_0, \cdots, h_T$ is collected where h_0 is the default hypothesis. This is followed by selecting h (the output of the batch algorithm) to be one of the $T+1$ hypothesis which converts the online algorithm to a batch one.

Helmbold and Warmuth [51] have argued that rather than selecting only a single hypothesis from the set of hypothesis, it is better to consider h to be some combination of the entire set of hypothesis. The different hypothesis could be combined by taking a *majority* or by *averaging*. In this way, the information retained by each and every hypothesis is used to define h and ultimately promotes robustness and stability. Furthermore, the training data plays absolutely no role in the process of combining different hypothesis which gives these methods the name *data independent* methods.

Dekel and Singer [30] have shown that the matrices (or vectors) learned during the earlier iterations of an online algorithm can be discarded as the online algorithm makes more mistakes in the beginning as compared to the end (e.g. h_0 is determined without observing any training example). This means that, in a sequence of p matrices learned $(A_1, \cdots, A_p)$, one can rely on the last q one and use the average over these q hypothesis (*suffix averaging* conversion). One extreme of this approach is to use all of the hypothesis while the other extreme is to retain only the last hypothesis or matrix and is also known as *last-hypothesis* technique [29]. *Suffix averaging* finds the best trade off between these two extremes. However, all of the hypothesis must be stored in memory as it finds the optimal suffix length only once the entire hypothesis sequence has been formed. Moreover, the required memory space grows linearly with the training set size.

Dekel [29] has addressed the problem faced by the *suffix averaging* technique and developed a method called *cutoff averaging*. One extreme of this method is just like the simple *averaging* method. However the other extreme converts this method to the *longest survivor* technique. In this way, there is no need to store all of the online hypothesis in the memory unlike the *suffix averaging* method and the memory space scales with square-root of the number of training examples in the worst case scenario. In a typical case, the required memory is much less than that of the worst case. A *cutoff parameter* k is used to get the online hypothesis sequence. It represents the minimum number of rounds during which the online algorithm must not suffer any loss. This is followed by finding a weighted averaging of the hypothesis selected, where the weight represents the additional number of iterations a hypothesis has survived once selected. It may be noted that in order to find the best value of k, the entire training data must be processed. However there is no need to store the entire sequence of hypothesis. The only requirement is to group together the hypothesis by their survival times, and store the average hypothesis for each

group along with the cumulative loss in each group.

2.2.3 Some Key Machine Learning Methods

Perceptron Algorithm

The perceptron algorithm was developed by Franck Rosenblatt [93]. It is a linear classifier used for binary classification and can be regarded as the simplest form of feed-forward neural network. It separates the objects using a linear hyperplane as shown in Figure 2.2. It is a very simple algorithm and it has been proved by Novikoff [77] that it converges after a finite number of epochs (iterations) if the data is linearly separable.

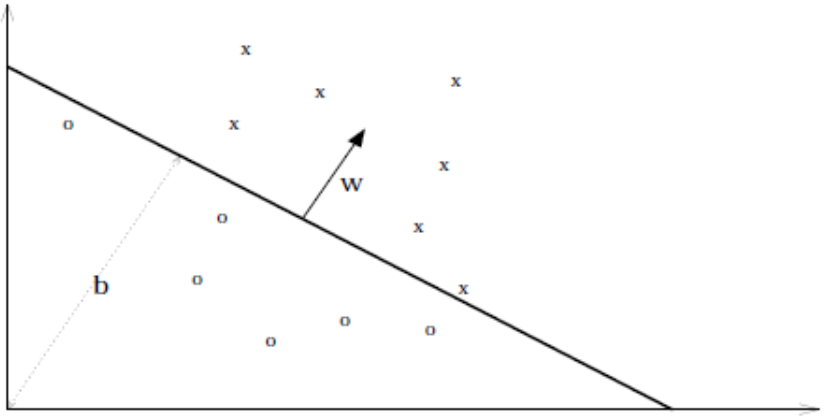

Figure 2.2: A hyperplane separating the two classes

The perceptron algorithm is an online supervised algorithm and the learning takes place in rounds or iterations. At each round, a new hypothesis is estimated based on the previous one. The algorithm starts with a hypothesis initialized with zero $w_1 = 0$. At each step, an instance x^τ is presented to the perceptron algorithm which makes a prediction $\hat{y}$ using the current hypothesis w_τ. This is followed by the revelation of the actual label y^τ. In case the actual label is different from the predicted one, the hypothesis is updated as $w_{\tau+1} = w_\tau + y^\tau x^\tau$. On the contrary if the actual label matches with that of the predicted one, the current hypothesis is left unchanged. The process is repeated for all of the training examples.

Voted perceptron of Freund and Schapire

Freund and Schapire [37] have introduced a variant of the perceptron algorithm for linear classification while attaining large margin, and have termed it as the voted-perceptron algorithm. Weights have been added to the prediction vectors which justifies the name *weighted perceptron*. Moreover, the standard perceptron algorithm (online) has also been converted to a batch one,

followed by an in-depth discussion on the online (incremental) to batch conversion. It can also classify instances having a relational representation (e.g. trees, graphs, or sequences). The proofs of convergence have been provided for both the separable as well as non-separable data.

It has been further suggested that the "kernel trick" can also be applied to the voted-perceptron algorithm [96]. The kernel trick is basically a method in which a linear classifier is converted to a non-linear one by mapping the original observations (e.g. x and x') to a higher dimensional space ($\phi(x)$ and $\phi(x')$) and then taking their inner product. This is equivalent to using the kernel function which is a function of two variables $K(x, x')$ and can be represented as an inner product $\phi(x).\phi(x')$ for some function ϕ. This implies replacing each inner product $x.x'$ with a kernel function computation $K(x, x')$. Kernel functions have also been used with support vector machines (SVMs).

The voted perceptron algorithm, being a supervised algorithm is composed of two steps: training and prediction. The initial prediction vector v_1 is set to zero just like the original perceptron algorithm. The prediction vector is used to predict the label of the new instance x. In the case of a wrong prediction $\hat{y} \neq y$, the prediction vector is updated while in the case of correct classification, it remains unchanged. The update is similar to that of the perceptron except the fact that the weight related with the current prediction vector i.e. w_τ is also updated. The weight is increased by one in case of correct classification. However, for misclassification, the weight related to the new prediction vector $w_{\tau+1}$ is initialized with 1. This process is then continued with the next example and is repeated for T epochs. Once the training is complete, a set of prediction vectors have been generated after each and every mistake. The weights related to the prediction vectors correspond to the number of examples they have *survived* until the next wrong classification. The weighted perceptrons can then be used to classify unseen test examples.

The Voted-Perceptron Algorithm

Training

Input: a labeled training set $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(n)}, y^{(n)})\}$, number of epochs T
Output: a list of weighted perceptrons $\{(v_1, w_1), \cdots, (v_k, w_k)\}$
Initialize: $k = 1, v_1 = 0, w_1 = 0$
  Repeat T times:
    For $i = 1, \cdots, n$
      Compute predictions: $\hat{y} = sign(v_k \,.\, x^{(i)})$
      If $\hat{y} = y_i$ then $w_k = w_k + 1$
      else $v_{k+1} = v_k + y^{(i)} x^{(i)}$
           $w_{k+1} = 1$
           $k = k + 1$

Prediction

Given: the list of weighted perceptron: $\{(v_1, w_1), ..., (v_k, w_k)\}$, an unlabeled instance: x
Compute a predicted label $\hat{y}$ as follows:
$s = \sum_{i=1}^{k} w_i \text{sign}(v_i.x)$; $\hat{y} = \text{sign}(s)$

During prediction, the votes are taken from all of the weighted perceptrons. As T approaches ∞ for linearly separable data, the voted perceptron converges to the original perceptron algorithm where the prediction is made using the last prediction vector.

The online to batch conversion can be called as a *voting* conversion as each online hypothesis $(v_1, \cdots, v_k)$ casts a classification vote for an unseen example x; and x gets the label that receives the highest number of votes.

Li et Long [64] have proposed an online algorithm called as *ROMMA* (Relaxed Online Maximum Margin Algorithm) for classification using a linear threshold function. The algorithm has been compared against the perceptron algorithm and the voted perceptron algorithm of Freund and Schapire, and it has been found that *ROMMA* performed better than the perceptron algorithm, and an aggressive version of *ROMMA* performed even better than the voted perceptron.

Collins extension of voted perceptron

Michael Collins [20] has used a variant of the perceptron algorithm for the part-of-speech tagging and base noun phrase recognition, related to the domain of Natural Language Processing. In this work, the voted or averaged version of the perceptron algorithm has been extended, originally introduced by Freund and Schapire. In addition, a parameter vector α (also referred to as the weights) is also introduced, which is trained on a set of training examples. This vector is then used for part-of-speech tagging or base noun phrase recognition. The proofs of convergence have been provided for the separable as well as for the non-separable data. Furthermore, it has been shown that the number of errors made by the algorithm is bounded not only on the training examples but also on unseen examples. The algorithm proposed by Collins can be applied to different other domains as well.

The parameter is considered to be associated with a trigram (x, y, z) as $\alpha_{x,y,z}$ and the one associated with a tag/ word pair (t, w) as $\alpha_{t,w}$. Moreover, a sequence of words $(w_1, \cdots, w_n)$ is represented as $w_{[1:n]}$ while $t_{[1,n]}$ is used to describe a tag sequence $(t_1, \cdots, t_n)$. The training set is made up of n tagged sentences where the length of ith sentence is n_i. This helps to write the examples as $(w^i_{[1:n_i]}, t^i_{[1:n_i]})$ where $i = 1, \cdots, n$. Furthermore, *Viterbi* algorithm is used in order to find the best tagged sequence for the sentence $w^i_{[1:n_i]}$ which is denoted by $z_{[1:n_i]}$. For every tag trigram (x, y, z) seen c_1 times in $t^i_{[1:n_i]}$ and c_2 times in $z_{[1:n_i]}$ with the condition that $c_1 \neq c_2$, the parameter associated with a trigram (x, y, z) can be expressed as:

$$\alpha_{x,y,z} = \alpha_{x,y,z} + c_1 - c_2$$

Similarly for each tag/word pair (t, w) seen c_1 times in $(w^i_{[1:n_i]}, t^i_{[1:n_i]})$ and c_2 times in $(w^i_{[1:n_i]}, z_{[1:n_i]}$ (with $c_1 \neq c_2$), $\alpha_{t,w}$ can be written as:

$$\alpha_{t,w} = \alpha_{t,w} + c_1 - c_2$$

Input: Training examples $(x^{(i)}, y^{(i)})$
Output: Parameters α
Initialization: Set $\alpha_1 = 0$
Algorithm:
For T iterations, $i = 1, \cdots, n$
Calculate $z^{(i)} = \arg\max_{z \in \text{GEN}(x^{(i)})} \phi(x^{(i)}, z).\alpha$
If $(z^{(i)} \neq y^{(i)})$ then
$\alpha_{l+1} = \alpha_l + \phi(x^{(i)}, y^{(i)}) - \phi(x^{(i)}, z^{(i)})$

where n represents the number of examples. If the highest scoring sequence under the current model $z^{(i)}$ is not correct $(z^{(i)} \neq y^{(i)})$, the parameter α is updated in a simple additive manner. It has been shown experimentally that instead of using only the final parameter α, it is better to use averaged parameters over T passes and n examples i.e. the averaged parameter γ can be written as:

$$\gamma = \sum_{t=1,\cdots,T;\ i=1,\cdots,n} \frac{\alpha^{t,i}}{nT}$$

The task in this algorithm is to learn a mapping from inputs $(x \in \mathcal{X})$ to outputs $(y \in \mathcal{Y})$. The parameter vector $\alpha \in \mathbb{R}^d$ is initialized with zero which is subsequently optimized over the training data. The function GEN lists a set of candidates $GEN(x)$ for an input x.

Support Vector Machines

Support vector machines (SVMs) are no doubt the most popular classification algorithms these days, mainly due to their results [26], [103]. We first discuss here the binary classification problem. The input space is denoted by $\mathcal{X} \subseteq \mathbb{R}^d$ where the value of d is fixed. A linear classifier is a function of $\mathbb{R}$ in $-1, 1$ having the form:

$$f(x) = sign(b^t x + b_0)$$

where $b \in \mathbb{R}^d$, while $b_0 \in \mathbb{R}$. The sign$(t) = 1$ if and only if $t > 0$, otherwise is equal to 0. It can be noted that the classifier $f(x) = b^t x + b$ divides $\mathcal{X}$ into two sub-spaces: $\{x \in \mathcal{X} \,|\, b^t x + b_0 < 0\}$ and $\{x \in \mathcal{X} \,|\, b^t x + b_0 > 0\}$

Here, a classifier $f(x) = b^t x + b_0$ having zero empirical loss is considered. This means that this classifier classifies correctly all of the examples in S. Since it is supposed that S is linearly separable, hence there exists a scalar such that the examples $(x^{(i)}, y^{(i)}$ which are nearest to the hyperplane satisfy $|b^t x + b_0| = 1$. Two examples $x^{(1)}$ and $x^{(2)}$ are further considered belonging to opposite classes, such that $b^t x^{(1)} + b_0 = 1$ and $b^t x^{(2)} + b_0 = -1$. The margin can be defined as the distance between these two points, where the margin is calculated perpendicular to the hyperplane. The margin (given in the figure 2.3) can also be represented by:

$$\frac{b}{\|b\|}(x^{(1)} - x^{(2)}) = \frac{2}{\|b\|}$$

It can be seen that in order to increase the margin, $\|b\|$ must be decreased. This can eventually help in order to have a hyperplane with a maximum margin.

SVM with a hard margin: The constraints $|b^t x + b_0| = 1$ can be written as $y(b^t x + b_0) = 1$ for the examples which are near to the hyperplane. The overall aim thus, is to resolve the following optimization problem:

$$\begin{aligned} \min_{b \in \mathbb{R}^d,\, b_0 \in \mathbb{R}} \quad & \frac{1}{2}\|b\|^2 \\ \text{such that} \quad & \forall i,\, y^{(i)}(b^t x^{(i)} + b_0) \geq 1 \end{aligned}$$

It can be observed that a quadratic optimization problem is being solved along with the linear constraints. A work around is to solve a dual problem in the following manner:

$$\begin{aligned} \max_{(\alpha_1, \cdots, \alpha_n) \in \mathbb{R}^d} \quad & \sum_{i=1}^{d} \alpha_i - \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} y^{(i)} y^{(j)} \alpha_i \alpha_j {x^{(i)}}^t x^{(j)} \\ \text{such that} \quad & \sum_{i=1}^{d} y^{(i)} \alpha_i = 0 \\ & \forall i, \alpha_i \geq 0 \end{aligned}$$

An advantage of the above formulation lies in the fact that b (the solution of the initial optimization problem) can be written as:

$$b = \sum_{i=1}^{d} y_i \alpha_i x^{(i)}$$

where $(\alpha_1, \cdots, \alpha_n)$ accounts for the optimal solution of the dual problem. One can also show that $\alpha_i > 0$ if and only if $y_i(b^t x^{(i)} + b_0) = 1$. The maximal margin hyperplane depends only on a subset of the examples. These examples lie exactly on the margin and are called the support vectors. The rest of the examples can lie anywhere outside the margin. In other words, one gets exactly a similar solution even if the training set S contains only the support vectors.

SVM with a soft margin: The SVM described earlier cannot deal with inseparable data and is therefore termed as having a *hard* margin. In actual practice, the data is rarely separable. One of the reasons is the presence of noise in the data. In case of non-separable data, SVM must live with wrongly classified examples. A simple way is to introduce slack variables, in which case a slack variable is associated with each examples. The use of slack variables allows to calculate a loss each time an example is misclassified. The resulting algorithm is said to have a *soft* margin. This also changes the aim and the new objective is to maximize the margin and minimize the number of examples violating the constraint on the margin. In other words, the norm of b and the overall loss associated with the slack variables is minimized. This new optimization problem can be written as:

$$\begin{aligned} \min_{b \in \mathbb{R}^d,\, b_0 \in \mathbb{R}} \quad & \frac{1}{2}\|b\|^2 + C \textstyle\sum_{i=1}^{n} \eta_i \\ \text{such that} \quad & \forall i,\, y^{(i)}(b^t x^{(i)} + b_0) \geq 1 - \eta_i \\ & \forall i,\, \eta_i \geq 0 \end{aligned}$$

where η_i stand for the slack variables while C is a positive real number which must be tuned. Whenever η_i is positive, this means that the margin constraint is not obeyed. The loss associated

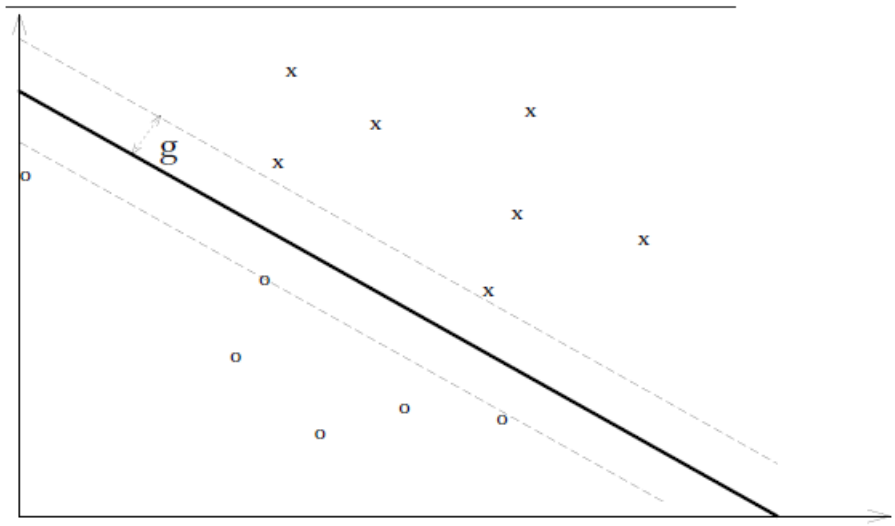

Figure 2.3: Maximum margin for support vector machines (SVM)

with this can be written as $C\eta_i$ which can be compensated while decreasing the norm of b. In case C is large, even a little violation of the constraint would be costly. Hence hyperplanes with small margins would be chosen with less number of errors. On the other hand, if C is small, the margin would be large and so do the number of errors. One way to tune C is to use cross-validation.

k Nearest Neighbor Algorithm

The k Nearest-Neighbor (kNN) algorithm [21], developed by Fix and Hodges [34], has been studied by many researchers, from many different communities. In the database community, for example, it is used to determine the instances closest to a given query point. In case-based reasoning, pattern recognition and machine learning, the kNN rule, because of its simplicity and good performance, is still heavily used for classification purposes e.g. image and text classification, web site classification [62] etc. This method is categorized as a non-parametric supervised learning algorithm and classifies instances based on the closest training examples in the feature space. In this method, all of the training points together with their class labels are kept in memory (hence referred to as memory-based method) and the computation is deferred until classification. Hence it is also known as a *lazy* method which belongs to the instance-based learning (IBL) methods. Nearest-neighbor learning has been shown to be the algorithmic parallel of the exemplar model of human learning [43]. Normalization of feature vectors may be required in some cases.

During the classification phase, when a query point is given, the classification of that point is made keeping in view the k nearest points. First of all, same features as for the training examples are computed for the query point, which is followed by the calculation of distance/similarity to all of the stored feature vectors. A metric is required for calculating the distance or the similarity between the query point and the instances from the training data in order to make predictions.

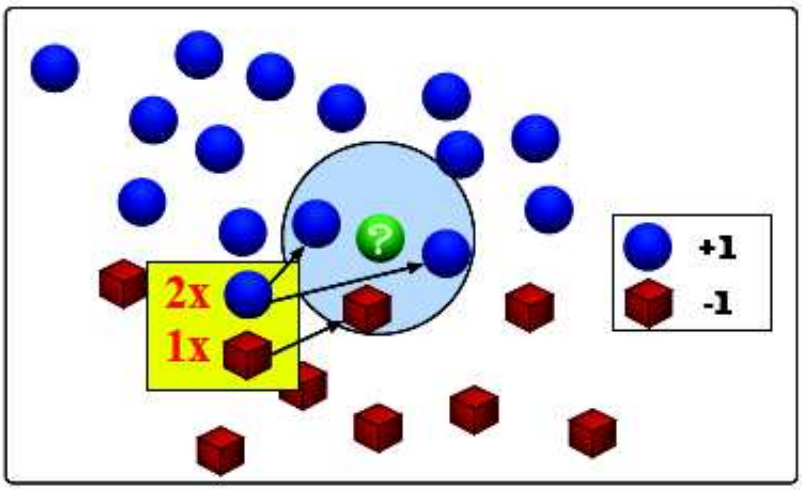

Figure 2.4: An example of a 3 nearest neighbor classification [108]

Some popular choices for the metric are the Euclidean distance and the cosine similarity. Some people use the term *metric* in order to signify distance or similarity, and sometimes this term is used to refer to distance only. However, the choice mainly depends on the problem domain. The distances and similarities are arranged in ascending and descending order respectively. This is followed by the selection of the the top k values in the sorted list. In the standard version, the query point is assigned the class that appears most frequently within the k nearest examples. Figure 2.4 shows the 3 nearest neighbors classification for an example represented by the symbol ?. This method is often successful when the decision boundary is very irregular [49]. In order to classify a new example x, the distances $d_i(x, x^{(i)}), \quad i = 1, \cdots, k$ between the new example and the k nearest neighbors are calculated. The smaller the distance, the greater is the similarity between two examples. Furthermore, the classes for the k nearest neighbors are also found. This is followed by assigning the new example x to the majority class g among the k nearest neighbors:

$$C(x) = \text{argmax}_{g \in \mathcal{G}} \sum_{i=1}^{k} k_i | k_i = \omega_g$$

where $C(x)$ represents the class of x and $\mathcal{G}$ is the set of all possible classes.

An important factor in this algorithm is the right choice of k which can strongly influence the quality of classifications assigned. The value of k can be determined from a validation set of examples. A smaller value of k leads to large variance in predictions for a given problem. On the other hand, larger values of k reduce the effect of noise on classification. Hence, k should be chosen in such a way that the value is large enough to minimize the probability of misclassification. Many experiments have shown that increasing the value of k does not significantly degrade the performance [98].

Another important issue is breaking ties among the k nearest neighbors. A tie occurs when two or more classes become the majority class. This can happen when k is even or odd in a multiclass problem. In the case of a binary problem, a tie can occur only when the value of k is even. A naïve approach to break ties is to pick any random majority class, but is definitely not

logical. Another type of a tie is the distance tie, which occurs when two or more neighbors are at the same distance from an example. Devroye et al. [31] have described a strategy where the ties are broken by indices i.e. if $x^{(i)}$ and $x^{(j)}$ are equidistant from x, then $x^{(i)}$ is declared closer to x if $i < j$.

Like any smoothing parameter, there is an optimal value of k for every problem. One possible method to find this optimal value is to use cross-validation. The simplest or the degenerate case is when the value of $k = 1$ and the algorithm is known as nearest neighbor (1NN) algorithm or sometimes as first nearest neighbor rule (FNN). It has been also shown that the FNN rule has an asymptotic error rate that is at most twice the Bayes error rate, independent of the distance function used.

Baoli et al. [2] have argued that having a fixed value for k results in a bias on large classes. This is specially true when the distribution of different classes in the training set is uneven. After finding the original k nearest neighbors, the probability that an example belongs to a certain class is computed using only some top p nearest neighbors for that class, where p is extracted from k based on the size of the class c_m. Generally speaking, different number of nearest neighbors are used for different classes. In order to make the comparison between different classes reasonable, the probabilities are derived from the proportion of the similarity sum of examples belonging to a class to the total sum of similarities for all of the selected neighbors for that class. The decision function can be given as follows:

$$y(d_i) = \arg\max_m \frac{\sum_{x^{(j)} \in top-p-kNN(c_m)} sim(d_i, x^{(j)})\, y(x^{(j)}, c_m)}{\sum_{x^{(j)} \in top-p-kNN(c_m)} sim(d_i, x^{(j)})}$$

where $top - p - kNN(c_m)$ represents the top p neighbors in the original k nearest neighbors. p can be calculated in the following manner:

$$p = \frac{k\, N(c_m)}{\max\{N(c_j) \,|\, j = 1, \cdots, N_c\}}$$

Here $N(c_m)$ represents the size of the class c_m while $\max\{N(c_j) \,|\, j = 1, \cdots, N_c\}$ is the size of the largest class in the training set.

The advantage of this algorithm lies in the fact that it is easier to implement and has good accuracy but, on the other hand, as it performs all of the computations at run time, it is a computationally intensive algorithm. Another possible approach for kNN is adding a threshold for each class, which may be learned using a validation set of examples [119]. In this case, the kNN method is not lazy any longer and a real training is performed. But at the same time, there is a loss in incremental behavior.

The nearest neighbor algorithm is less appealing with limited training examples because of the curse of dimensionality. Support vector machines have also been used along with *kNN* to increase the margin between the positive and the negative examples in the weighted space in which the classification is performed. Nock and Sebban [74] have developed a non-linear hyperplane with a large margin by computing the weights of the reference examples.

Another variant of kNN is the *Weighted* kNN [18], [79] where an ith neighbor ($i = 1, \cdots, k$) is assigned a weight w_i. The test sample x is classified as the class $\hat{y}$ that is assigned the maximum

weight:

$$\hat{y} = \arg\max_{g \in \mathcal{G}} \sum_{i=1}^{k} w_i I_{\{y^{(i)}=g\}}$$

Here $\mathcal{G}$ represents the set of classes while I is the indicator function having the value 0 or 1.

Distance-weighted nearest neighbor rule allows all of the training samples to cast votes where the votes for the closest samples have greater weight than the samples further away. The intuition behind this idea is that the nearer neighbors should provide more information than the distant ones. The weight for a vote decreases with the increase in distance from the query point. Another variation is the *rank-weighted* nearest neighbor technique, in which the closest neighbors can cast more votes as compared to the far-off neighbors.

Bay [4] has developed a technique *MFS* (Multiple Feature Subsets) which combines multiple nearest neighbor classifiers each using only a subset of features.

2.3 Metric Learning

Metric has always been a very important and decisive ingredient of many machine learning problems. Among these, the performance of k-nearest algorithm heavily depends on whether the metric chosen takes into account the underlying geometry of the space in which the examples lie or not. Metric learning can be further subdivided into two different types: distance metric learning and similarity metric learning.

2.3.1 Distance Metric Learning

Distance measures the dissimilarity in a given data set. A value of 0 indicates the examples to be totally similar while a value of 1 means that the examples are completely distinct. There are many different possibilities for distance functions like the Euclidean distance, the City-Block distance, the Mahalanobis distance etc.

Definition of a Distance: The distance over a set $\mathcal{X}$ is defined as a function d (also known as the distance function) such that:

$$d : \mathcal{X}\mathrm{x}\mathcal{X} \Rightarrow \mathbb{R},$$

$\forall x, x', x'' \in \mathbb{R}$, this function needs to satisfy the following four conditions:

1. $d(x, x') \geq 0$ (also known as non-negativity)
2. $d(x, x') = 0$ iff $x = x'$ (distinguishability)
3. $d(x, x') = d(x', x)$ (symmetry)
4. $d(x, x') + d(x', x'') \leq d(x, x'')$ (triangle inequality)

The first and second conditions together produce the positive semi-definitiveness [82]. A pseudo-metric satisfies all of the requirements for a metric, except the second one. This means that one may have $d(x, x') = 0$ for even distinct values $x \neq x'$.

Various distances are defined hereafter:

For two examples, $x(x_1, x_2, \cdots, x_d)$ and $x'(x'_1, x'_2, \cdots, x'_d)$, the Euclidean distance function (also known as L2 norm) can be written as:

$$d_2(x, x') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}$$

A generalization of the Euclidean distance is the *Minkowski* function which can be written as:

$$d_t(x, x') = \sqrt[t]{\sum_{i=1}^{d} w_i (x_i - x'_i)^t}$$

Here w_i represents the weight corresponding to the ith feature of x and x'. The Euclidean distance can be obtained by setting t to 2 and each weight, w_i, to 1 in the above equation. Setting t and all of the weights w_i to 1 results in the L1 norm (also known as Manhattan or City Block distance). It represents the distance between two points in a city road grid and examines the absolute differences between coordinates of a pair of points:

$$d_1(x, x') = \sum_{i=1}^{d} |x_i - x'_i|$$

Setting t to ∞, gives the maximum value distance or Chebyshev distance:

$$d_\infty(x, x') = \max_{i=1}^{d} |x_i - x'_i|$$

A family of metrics over a vector space $\mathcal{X}$ could be obtained by computing Euclidean distances after performing a linear transformation $x'' = \mathcal{L}x$. These metrics compute square distances in the following manner:

$$d_L^2(x, x') = \|\mathcal{L}(x - x')\|_2^2 \tag{2.1}$$

where the linear transformation is parametrized by the matrix $\mathcal{L}$. The equation 2.1 can also be written in terms of a square matrix A:

$$A = \mathcal{L}^t \mathcal{L}$$

Any matrix A created from a matrix $\mathcal{L}$ in this manner is always positive, semi-definite (PSD) (written as $A \succeq 0$) which means that there are no negative eigenvalues [112]. The square distances can also be expressed in terms of the matrix A:

$$d_A^2(x, x') = (x - x')^t A (x - x') = \|x - x'\|_A^2 \tag{2.2}$$

where equation 2.2 defines the *Mahalanobis* distance [69]. The Mahalanobis distance [3, 10] is used, originally, to describe the quadratic forms in Gaussian distributions where it was the inverse of the covariance matrix used to incorporate the correlations of different feature dimensions [106]. It generalizes the Euclidean distance by admitting arbitrary linear scalings and rotations of the feature space [28]. Choosing A to be the identity matrix, the Mahalanobis distance reduces to the Euclidean distance. The Mahalanobis distance can either be parametrized in terms of the matrix $\mathcal{L}$ or in terms of A, which means that either a linear transformation $\mathcal{L}$ is estimated or a

PSD matrix A. The optimization is unconstrained in the case of the first approach while in the second approach it is mandatory to enforce the constraint that the matrix A must be positive, semi-definite.

Moreover, in case the matrix A is diagonal, the resulting distance is called the normalized Euclidean distance where the different axes are given different *weights*:

$$d_{NE}(x, x') = \sqrt{\sum_{i=1}^{d} \frac{(x_i - x'_i)^2}{\sigma_i^2}}$$

where σ_i is the standard deviation of x_i over the sample set.

Having introduced various distance metrics, the next question is how to learn these distance metrics in an effective manner [109]. Many state of the art metric learning algorithms are next presented and compared in detail.

Metric learning algorithms can be broadly classified into supervised metric learning algorithms and unsupervised learning algorithms (covering linear (Principal Component Analysis (*PCA*) [45], Multidimensional Scaling (*MDS*) [22]) and nonlinear embedding methods (e.g. Locally Linear Embedding (*LLE*) [94]) depending on the fact whether the label or side information has been used or not. Empirical studies have shown that, in general, supervised metric learning algorithms outperform unsupervised ones [107]. Unlike most supervised learning algorithms where each training example has been assigned a label, a supervised distance metric learning algorithm is generally based on two types of pairwise constraints: equivalence and inequivalence constraints. Equivalence constraints consider those examples which belong to the same classes where as inequivalence constraints deal with data points belonging to different classes [4].

Rather than using the *absolute* qualitative feedback (e.g. A and B are *similar* or A and C are *not similar*), some works like Schultz and Joachims [97] and Frome et al. [38] consider *relative* qualitative examples (e.g. A is more similar to B than A is to C). A practical example of this scenario is search-engine query logs, where the documents that are clicked can be considered to be semantically closer than the ones that the user observed but decided not to click.

Supervised metric learning algorithms could be further categorized into global metric learning algorithms, local metric ones or pseudo global/local ones. It is possible to formulate certain distance metric learning problems as convex optimizations over the cone of PSD matrices.

Global Distance Metric Learning

Global metric distance learning algorithms learn the distances in a global sense where the aim is to satisfy *all* of the pairwise constraints (equivalence as well as inequivalence) simultaneously. Such algorithms try to learn metrics in such a way that all of the examples belonging to the same classes are kept close while separating apart the examples from different classes. More oftenly, the distance function is explicitly learned in such a way that the distance between examples within the equivalence constraints is *minimized* while the distance between examples belonging to inequivalence constraints is *maximized* [113], [116], [117].

[4] Wang et al. [106] have termed the equivalence constraints as *must-link* constraints while inequivalence ones as *cannot-link* constraints.

Information Theoretic Distance Metric Learning

Davis et al. [28] have developed an Information-theoretic (Information-Theoretic Metric Learning - ITML) approach to learn (squared) Mahalanobis distances. This method does not require semi-definite programming and eigen-value decompositions which makes it faster and scalable. Two types of relationships between the examples are considered: similarity and dissimilarity. In this regard, two points x and x' are considered similar if the distance between them is less than a certain threshold u. Similarly, these points are dissimilar if the distance between them is greater than a sufficiently large threshold l.

The aim here is to learn the positive definite matrix A which parametrizes the Mahalanobis distance given in the equation 2.2. An input Mahalanobis matrix A_0 is also considered, which can be determined from the training data. For Gaussian data, A_0 can be initialized with the inverse of the sample covariance. Similarly A_0 can also be determined using the squared Euclidean distance. This is followed by bringing the matrix A (also known as the output matrix) as close as possible to the initial matrix A_0 using an *information theoretic* approach. The set of Mahalanobis distances are related to the set of multivariate Gaussian distributions [5] with an equal mean μ as follows:

$$p(x; A) = \frac{1}{Z}\exp(-\frac{1}{2}d_A(x, \mu))$$

where $p(x; A)$ is the multivariate Gaussian of the matrix A or the probability density function (pdf), Z is a normalizing constant and A^{-1} is the covariance matrix of the Gaussian distribution. The greater the distance d_A, the smaller the value of the probability. This helps to calculate the distance between the two Mahalanobis distance functions parametrized by A_0 and A i.e. $d(A_0\|A)$ using the relative entropy or the *Kullback-Leibler* divergence (KL divergence) [6] between their multivariate Gaussians:

$$d(A_0\|A) \;=\; \mathrm{KL}((p(x; A_0) \parallel (p(x; A)) \;=\; \int p(x; A_0)\;\log\frac{p(x; A_0)}{p(x; A)}dx$$

Thus, the distance metric learning problem can be written as:

$$\min_{\mathrm{A}}\;\mathrm{KL}\left((p(x; A_0)\|(p(x; A)\right)$$

$$\begin{aligned} \text{with} \quad & d_A(x, x') \leq u \qquad (x, x') \in S \\ & d_A(x, x') \geq l \qquad (x, x') \in D \end{aligned}$$

Here the aim is to minimize the KL divergence between the two Gaussians. Moreover, S represents the similar points whereas D is used to denote the dissimilar points. In order to use Bregman projections to learn the matrix A, it has been shown that the information theoretic objective can be described in terms of Bregman divergence. Considering the fact that the Log-Det (logarithm-determinant) divergence (D_{ld}) is actually a Bregman divergence defined over the cone of PSD matrices [60], [61]:

$$D_{ld}(A, A_0) = tr(AA_0^{-1}) - \mathrm{logdet}(AA_0^{-1}) - n$$

[5] Also known as multivariate normal distribution.

[6] KL divergence is also known as the information gain or information divergence.

Furthermore, Kulis et al. [60] have shown that the KL divergence between two multivariate Gaussian distributions can be written as the convex combination of Mahalanobis distance between mean vectors and the LogDet divergence between the covariance matrices. Considering the means of the two Gaussians to be the same, the KL divergence can be related to the Mahalanobis distance in the following manner:

$$\mathrm{KL}((p(x;A_0)\|(p(x;A)) = \frac{1}{2}D_{ld}(A_0^{-1}, A^{-1}) = \frac{1}{2}D_{ld}(A, A_0)$$

Moreover, the LogDet divergence is independent of the scaling of the feature space. With this, the distance metric learning problem can be written as a *LogDet* optimization problem:

$$\begin{array}{lll} \min\limits_{A\succeq 0} & D_{ld}(A, A_0) & \\ \text{such that} & tr(A(x^{(i)} - x^{(j)})(x^{(i)} - x^{(j)})^t) \leq u & (i,j) \in S, \\ & tr(A(x^{(i)} - x^{(j)})(x^{(i)} - x^{(j)})^t) \geq l & (i,j) \in D, \end{array}$$

The imposed constraints on the distances can be relaxed using slack variables to find an admissible solution.

It can be concluded that by using a LogDet divergence between two matrices along with an initial PSD matrix, all of the subsequent matrices are PSD as well and no projection is required [60]. However, a major shortcoming of this algorithm is its quadratic dependency on the dimensionality d.

Davis and Dhillon [27] learn low rank Mahalanobis distance metrics for high dimensional problems.

Pseudo-distance Online Learning Algorithm (POLA)

Shalev et al. [99] learn pseudo-distances parametrized by positive semi-definite matrices along with a scalar threshold in an online as well as batch setting. Convex optimizations over the cone of PSD matrices for distance metric learning have also been proposed. Like many other distance metric learning algorithms, the aim is to learn a metric that shrinks distances between similarly labeled examples while expanding distances between examples with different labels. The algorithm is termed as Pseudo-distance Online Learning Algorithm or *POLA*. Examples are composed of an instance pair and a label which can be $+1$ or -1 depending on the fact that the instances are similar or dissimilar. The algorithm is based on Mahalanobis distance d_M just like Xing et al. [114]. However this algorithm is online and comes with theoretical error guarantees.

Using a threshold $b \in \mathbb{R}$, the constraints for similar and dissimilar examples could be defined in the following manner:

$$\begin{aligned} \forall(x, x', y) : y = +1 \quad &\rightarrow (d(x, x'))^2 \leq b - 1, \\ \forall(x, x', y) : y = -1 \quad &\rightarrow (d(x, x'))^2 \geq b + 1, \end{aligned}$$

where the maximum distance in case of similar examples is $b - 1$. Consequently, the distance is at least equal to $b + 1$ for the dissimilar examples. These two inequalities can be combined to form a single linear constraint:

$$y(b - (d_A(x, x'))^2) \geq 1$$

The aim here is to learn the matrix A, where $A \succeq 0$ like many other distance metric learning methods. Being an online algorithm, the algorithm receives the examples in the form of tuples $(x_\tau, x'_\tau, y_\tau)$ in a sequential manner. A distance $d_M(x_\tau, x'_\tau)$ is calculated for each pair of examples at a time step τ. In case, the square of this distance is greater than the threshold b, the current pair is considered as dissimilar. On the contrary, it is considered as similar. Once the prediction has been given, the true label y_τ is received, based on which the algorithm may suffer from a loss:

$$l_\tau(A, b) = \max\left\{0,\ y_\tau((d_A(x_\tau, x'_\tau))^2 - b) + 1\right\}$$

It may be noted that this loss is a modified form of the hinge loss. The goal of the online algorithm is to reduce the *cumulative* loss. The matrix A and the threshold b are updated at each step upon receiving the feedback y_τ.

In order to define an online update rule for A and b, an orthogonal projection has been used. Suppose there is a vector $x \in \mathbb{R}^p$ along with a closed convex set $C \subset \mathbb{R}^p$. The orthogonal projection of x onto C can be given as:

$$\mathcal{P}_C(x) = \underset{x' \in C}{\operatorname{argmin}} \|x - x'\|_2^2$$

In order words, the aim is to find the closest point of x in the set C. Here $\mathcal{P}_C(x)$ is the vector in C that is closest to x. Moreover, (A, b) is considered both as a matrix-scalar pair and as a vector in $\mathbb{R}^{n^2+1}$ where the first n^2 elements represent the matrix A where as the last element stands for the threshold b. At each time step τ, the set $C_\tau \subset \mathbb{R}^{n^2+1}$ can be defined as follows:

$$C_\tau = \left\{(A, b) \in \mathbb{R}^{n^2+1} : l_\tau(A, b) = 0\right\}$$

where C_τ represents a set of all those matrix-threshold pairs that attain zero loss on the current example i.e. $(x_\tau, x'_\tau, y_\tau)$. Moreover, it is known that $A \succeq 0$ and the threshold must be greater than or equal to 1, since the loss between two similar points would be non-zero if $b < 1$. This allows to define another set C_a which is the set of all admissible matrix-threshold pairs:

$$C_a = \left\{(A, b) \in \mathbb{R}^{n^2+1} : A \succeq 0,\ b \geq 1\right\}$$

The update for the online algorithm consists of two projections: first onto C_τ and then onto C_a. The first projection onto C_τ gives $(A_{\hat{\tau}}, b_{\hat{\tau}})$ as the matrix-threshold pair. The aim is to keep $(A_{\hat{\tau}}, b_{\hat{\tau}})$ as close as possible to (A_τ, b_τ), while $(A_{\hat{\tau}}, b_{\hat{\tau}})$ is forced to attain a zero loss on the current example. The second projection onto C_a gives $(A_{\tau+1}, b_{\tau+1})$ which makes sure that the new matrix-threshold pair is admissible for deciding whether the current examples are similar or not.

Projection onto C_τ

In order to project (A, b) onto C_τ, $w \in \mathbb{R}^{n^2+1}$ is considered to be the vector representation of (A, b). Similarly w_τ, $w_{\hat{\tau}}$ and $w_{\tau+1}$ represent the vectors associated with (A_τ, b_τ), $(A_{\hat{\tau}}, b_{\hat{\tau}})$ and $(A_{\tau+1}, b_{\tau+1})$ respectively. Moreover, let $\mathcal{X}_\tau \in \mathbb{R}^{n^2+1}$ be the vector representation of the matrix scalar pair $(-y_\tau v_\tau v_\tau^t, y_\tau)$ where $v_\tau = x_\tau - x'_\tau$. It is further known that the projection onto C_τ

ensures zero loss which means that:

$$\begin{aligned} & y_\tau(b - d_A^2) && \geq 1 \\ \Rightarrow \quad & y_\tau b - y_\tau d_A^2 && \geq 1 \\ \Rightarrow \quad & y_\tau b - y_\tau (x - x')A(x - x')^t) && \geq 1 \end{aligned}$$

The definition of C_τ can be rewritten as:

$$C_\tau = \left\{ w \in \mathbb{R}^{n^2+1} : w.\mathcal{X}_\tau \geq 1 \right\}$$

The projection of w_τ onto C_τ can be given by:

$$\mathcal{P}_{C_\tau}(w_\tau) = w_\tau + \alpha_\tau \mathcal{X}_\tau$$

where $\alpha_\tau = 0$ iff $w.\mathcal{X}_\tau \geq 1$. Otherwise $\alpha_\tau = \dfrac{1 - w_\tau.\mathcal{X}_\tau}{\|\mathcal{X}_\tau\|_2^2}$. Furthermore, α_τ can be written as:

$$\alpha_\tau = \frac{l_\tau(A_\tau, b_\tau)}{\|\mathcal{X}_\tau\|_2^2} = \frac{l_\tau(A_\tau, b_\tau)}{\|v_\tau\|_2^4 + 1}$$

The updates for A_τ as well as for b_τ can now be written as:

$$A_{\hat{\tau}} = A_\tau - y_\tau \alpha_\tau v_\tau v_\tau^t \,, \quad b_{\hat{\tau}} = b_\tau + \alpha_\tau y_\tau$$

Projection onto C_a

After projecting (A_τ, b_τ) onto C_τ, $(A_{\hat{\tau}}, b_{\hat{\tau}})$ is projected onto C_a which can be written as:

$$(A_{\tau+1}, b_{\tau+1}) = \mathcal{P}_{C_a}(A_{\hat{\tau}}, b_{\hat{\tau}})$$

where $A_{\tau+1}$ is the projection of $A_{\hat{\tau}}$ onto the set of all positive semi-definite (PSD) matrices and $b_{\tau+1}$ is the projection of $b_{\hat{\tau}}$ onto the set $b \in \mathbb{R} : b \geq 1$. The projection of $b_{\hat{\tau}}$ onto the aforementioned set is maximum of 1 and $b_{\hat{\tau}}$. In order to project $A_{\hat{\tau}}$ onto the set of all PSD matrices, there are two possibilities: $y_\tau = +1$ or $y_\tau = -1$. In case where the current examples are dissimilar, the update would be $A_{\hat{\tau}} = A_\tau + \alpha_\tau v_\tau v_\tau^t$ where $\alpha_\tau \geq 0$. This implies that $A_{\hat{\tau}} \succeq 0$. Hence the projection of $A_{\hat{\tau}}$ onto the set of PSD matrices is $A_{\hat{\tau}}$. In case the current examples are similar, there is no surety that $A_{\hat{\tau}} \succeq 0$. Since $A_{\hat{\tau}}$ is symmetric, it can be rewritten in terms of its eigenvalues and eigenvectors:

$$A_{\hat{\tau}} = \sum_{i=1}^{n} \lambda_i u_i u_i^t$$

where λ_i stands for the i'th eigenvalue while u_i represents the i'th eigenvector of $A_{\hat{\tau}}$. Since the matrix $A_{\tau+1}$ is the projection of $A_{\hat{\tau}}$ onto the PSD cone, $A_{\tau+1}$ can be written in the following manner:

$$A_{\tau+1} = \sum_{i \,:\, \lambda_i > 0} \lambda_i u_i u_i^t$$

Here it can be seen that the interest lies only in the positive eigenvalues. Moreover, using the eigenvalue Interlacing Theorem, it is known that $A_{\hat{\tau}}$ can have at most a single negative eigenvalue. With this, the projection onto the PSD cone can be written as:

$$A_{\tau+1} = A_{\hat{\tau}} - \lambda_n u_n u_n^t$$

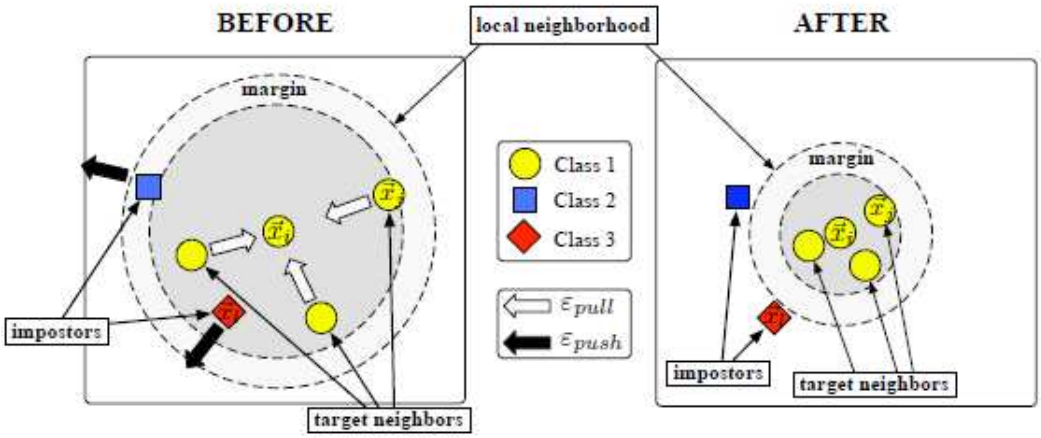

Figure 2.5: Neighbors of the instance $x^{(i)}$: before and after training [108]

where λ_n represents the minimal eigenvalue of $A_{\hat{\tau}}$ while u_n is its corresponding eigenvector.

Large scale classification using distance metric learning

Weinberger et al. [110], [112] have used the Mahalanobis distance for k nearest neighbor (kNN) using semi-definite programming. A semi-definite program, also known as *SDP*, is a linear program where the matrix whose elements are linear in the unknown variables must be positive, semi-definite having no negative eigenvalues. *SDPs* are convex which means that the global minimum can be computed easily.

The distance is optimized in such a way that the k nearest neighbors belonging to the same class (also called as the *target* neighbors) are attracted while examples belonging to different classes (called as the *impostors*) are separated by a large margin. In other words, the target neighbors define a perimeter around an example $x^{(i)}$, which the differently labeled inputs should not invade. Furthermore, the differently labeled examples that invade this perimeters are referred to as the *impostors*. The overall aim is to reduce the number of impostors. This is shown in figure 2.5. The distance is optimized with the view that the target neighbors (belonging to the same class) are located within a smaller radius after training; while differently labeled neighbors are located outside this radius, with a margin of at least one unit distance. This helps to maintain a large (finite) distance between the impostors and the perimeters established by target neighbors. The idea is to learn a linear transformation $\mathcal{L}$ where:

$$d(x, x') = \|\mathcal{L}(x - x')\|_2^2$$

In order to describe the impostors, $x^{(j)}$ is considered to be a target neighbor of an example $x^{(i)}$ with a label $y^{(i)}$. Then $x^{(l)}$ represents an impostor with the label $y^{(l)} \neq y^{(i)}$ such that:

$$\|\mathcal{L}(x^{(i)} - x^{(l)})\|_2^2 \leq \|\mathcal{L}(x^{(i)} - x^{(j)})\|_2^2 + 1 \tag{2.3}$$

The loss function is made up of two terms: the first one *pulls* the target neighbors closer and reduces the distances while the second one acts to *push* differently labeled examples further apart and hence increases the distances.

The first term in the loss function penalizes large distances between an input and its target neighbors. The sum of these squared distances can be given by:

$$\varepsilon_{\text{pull}}(\mathcal{L}) = \sum \|\mathcal{L}(x^{(i)} - x^{(j)})\|^2$$

where $x^{(j)}$ is a target neighbor of $x^{(i)}$. A good thing about this approach is that it only penalizes large distances between an input example and its target neighbors and not with all of the examples having similar class labels.

The second term in the loss function disfavors small distances between an input and all other examples that do not share the same class label. In order words, this terms penalizes the violators of the equation 2.3:

$$\varepsilon_{\text{push}}(\mathcal{L}) = \sum_{ij} \sum_{l} (1 - y^{(il)})[1 + \|\mathcal{L}(x^{(i)} - x^{(j)})\|^2 - \|\mathcal{L}(x^{(i)} - x^{(l)})\|^2]_+$$

where $y^{(il)} = 1$ if and only if $y^{(i)} = y^{(l)}$, and is 0 otherwise. Moreover, the terms $[z]_+ = \max(z, 0)$ and represents the standard hinge loss. It has been further suggested that the unit margin can be changed if desired.

With this, the two terms ($\varepsilon_{\text{pull}}$ and $\varepsilon_{\text{push}}$) can be combined to form the loss function. As the two terms have different aims: to attract the target neighbors and to repel the impostors; a weighting parameter $\mu \in [0, 1]$ is used:

$$\epsilon(\mathcal{L}) = (1 - \mu)\, \varepsilon_{\text{pull}}(\mathcal{L}) + \mu\, \varepsilon_{\text{push}}(\mathcal{L})$$

The loss function defined above is not convex. In order to reduce the loss, gradient descent algorithm could be used. However, this might result in local minima. A work around is to rewrite the loss function as an instance of semi-definite programming.

The algorithm has been tested on different datasets e.g. *Iris*, *Wine*, *Isolet* etc. The Principal Component Analysis (*PCA*) is used in order to reduce the number of dimensions. The results show significant improvement as compared to kNN algorithm employing Euclidean distance on all but the smallest data sets. The results are even comparable to the one using multi-class *SVM*s [25].

Xing's Distance Metric Learning Algorithm for Clustering

Xing et al. [114] were the people who first proposed a convex objective function. An algorithm was presented to learn the Mahalanobis distance for clustering based on similar and dissimilar pairs of points. Given a set of data points, the aim is to minimize the squared distance between similar examples or points while maximizing the distances between differently labeled examples. If two examples x and x' are similar, $(x, x') \in S$ where S represents all of the similar examples (also known as equivalence constraints) just like the *ITML* algorithm of Davis et al [28]. Similarly D represents the pairs which are dissimilar in case the information about the dissimilar pairs is available. On the contrary, all of the pairs which are not in S, can be added in the D set to form the inequivalence constraints. This can be expressed in the form of an optimization problem:

$$\begin{array}{ll} \min\limits_{\mathrm{A} \in \mathbb{R}^{d \mathrm{X} d}} & \sum_{(x,x') \in S} \|x - x'\|_A^2, \\ \text{such that} & \sum_{(x,x') \in D} \|x - x'\|_A \geq 1, \\ & A \succeq 0 \end{array}$$

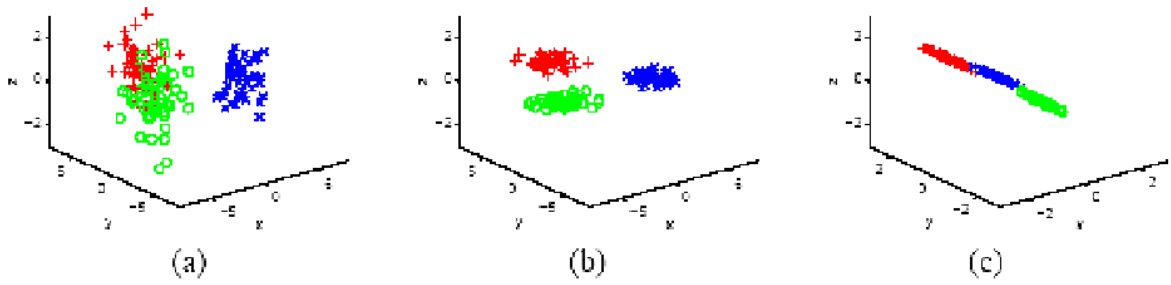

Figure 2.6: Xing's algorithm on 3 class data (a) Original data (b) Rescaling corresponding to learned diagonal matrix A (c) Rescaling corresponding to full A [114]

The constraint on D makes sure that the problem is feasible and bounded and A does not collapse the dataset into a single point in which case the distance between all similar points would become zero. Furthermore, it has been argued that if the squared distance is used for the dissimilar points as well, then the matrix A will always have rank 1 and the data would be projected on a line. Both of the constraints are convex which makes the optimization problem as *convex*. The algorithm is used to learn both diagonal A as well as full A. For diagonal A, the Newton-Raphson method has been used to learn A whereby $g(A)$ is minimized:

$$g(A) = \sum_{(x,x')\in S} \|x - x'\|_A^2 - \log\left(\sum_{(x,x')\in D} \|x - x'\|_A\right)$$

The first term or the distance between the similar points is reduced while the second term within the logarithm or the distance between dis-similar examples is increased.

In case where the full matrix is learned, the Newton-Raphson method cannot be used since it becomes way too expensive computationally. This is the reason why gradient descent is used like Weinberger's *LMNN* [112], along with the iterative projections to learn A. The resulting problem can be given as:

$$\begin{aligned} \max_{A} \quad & g(A) = \textstyle\sum_{(x,x')\in D} \|x, x'\|_A \\ \text{such that} \quad & f(A) = \textstyle\sum_{(x,x')\in S} \|x, x'\|_A^2 \leq 1, \\ & A \succeq 0 \end{aligned} \tag{2.4}$$

Here, the aim is *slightly* changed and the effort is made to *maximize* the distance between dis-similar points which belong to D whereas the original optimization problem was to *minimize* the distance between the similar points. Figure 2.6 shows a 3 class data in which case the centroids of the clusters differ only in x and y directions. As shown in figure 2.6(b), the learned diagonal metric correctly ignores the z direction. Furthermore, in the case of full A (figure 2.6(c)), the algorithm finds a projection of the data on a line that maintains the separation between the clusters.

A gradient ascent step is used to optimize equation 2.4 which can be given as $A = A + \alpha\nabla_A g(A)$. This is followed by repeatedly projecting the A matrix onto the sets $C_1 = \{A : \sum_{(x,x')\in S} \|x - x'\|_A^2 \leq 1\}$ and $C_2 = \{A : A \succeq 0\}$. The projection of A onto C_1 can be written as:

$$A = \arg\min_{A'}\{\|A' - A\|_F^2 : A' \in C_1\}$$

where $\|.\|_F$ represents the *Frobenius* norm, a type of the *entry-wise* norms. A *Frobenius* norm of a matrix P is the square root of the sum of the entries p_{ij} where i represents the rows whereas j stands for the columns. For the second projection onto C_2, the diagonalization of the matrix A is found:

$$A = X^t \Lambda X$$

where Λ is a diagonal matrix that is composed of the eigenvalues of the matrix A $(\lambda_1, \cdots, \lambda_n)$ and the columns of the matrix X make up the eigenvectors for A. In order to convert A into a positive semi-definite matrix, only the positive eigenvalues are taken into account and the negative ones are replaced with zeros. The following formula can then be used:

$$A' = X^t \Lambda' X$$

where Λ' is a diagonal matrix consisting of only positive eigenvalues.

Xing's algorithm is batch and does not has a computationally efficient online version like that of *POLA* [99]. Moreover, there are no theoretical error guarantees which means that there is no surety that the algorithm would make a *limited* number of mistakes on unseen examples. It is also implicitly assumed that the classes form a single compact connected set, which is detrimental in the case of highly multimodal class distributions.

Maximally Collapsing Metric Learning (*MCML*)

Another global distance metric learning approach is developed by Globerson et al. [41] where the aim is to collapse all of the examples belonging to the same class to a single point and push the examples from different classes infinitely apart. The goal is to learn a Mahalanobis distance metric. The objective function in this case is convex over the space of positive, semi-definite matrices, which in other words mean that there is a unique minimum. The goal is to have zero distance between the examples from the same class where as the distance between examples pertaining to different classes should be infinite. A conditional distribution has been defined for each of the training examples $x^{(i)}$ over other examples $x^{(j)}$ where $i \neq j$:

$$p_{ij} = \frac{exp(-\|Ax^{(i)} - Ax^{(j)}\|^2}{\sum_{j \neq i} exp(-\|Ax^{(i)} - Ax^{(j)}\|^2}, \;\; p_{ii} = 0$$

where p_{ij} represents the probability with which an example $x^{(i)}$ selects another example $x^{(j)}$ as its neighbor and $x^{(j)}$ share the class label with $x^{(i)}$. The ideal case where all the examples from the same class are mapped to a single point and infinitely apart from the examples belonging to different classes can be represented as:

$$p'_{ij} \propto \begin{cases} 1 & y^{(i)} = y^{(j)} \quad (d_{ij} = 0) \\ 0 & y^{(i)} \neq y^{(j)} \quad (d_{\infty} = 0) \end{cases}$$

The idea is to find a matrix A in such a way that p_{ij} is as close as possible to p'_{ij}. This can be achieved by minimizing the KL divergence between the two probability distributions:

$$min_A \sum_j KL[p'_{ij} \,|\, p_{ij}]$$

such that A is a PSD matrix. This optimization problem is convex over the space of PSD matrices and has a unique solution like many other approaches discussed earlier: POLA [99], Weinberger et al. [112], Xing et al. [114]. However, a disadvantage of this approach is that it assumes that the examples in each class have a unimodal distribution.

This method is based on Neighborhood Components Analysis (NCA) by Goldberger et al. [42] who also learn a Mahalanobis distance metric but especially for *kNN* classification. This algorithm finds the leave-one-out error or the $0-1$ loss from a stochastic variant of *kNN* classification. However, the objective function is not convex unlike *MCML* and can suffer from the problem of local minima.

Online Learning of Image Similarity - *OASIS*

Gal et al. [16] learn image (dis)similarity using an online algorithm called *OASIS* for Online Learning for Scalable Image Similarity learning. *OASIS* learns a bi-linear distance measure and belongs to the *Passive Aggressive* family of learning algorithms. The aim is to learn a pairwise similarity function S with large margin and an efficient hinge loss based on the relative similarity of pairs of images. It does not require the similarity measure to be PSD or even symmetric unlike many other works e.g. Weinberger et al. [112], Xing et al. [41] etc.

In order to dig deeper into the algorithm, consider $\mathcal{X}$ to be a set of images, and $r_{ij} = r(x^{(i)}, x^{(j)}) \in \mathcal{R}$ be a pairwise relevance measure which shows how strongly $x^{(i)}$ is related to $x^{(j)}$. Furthermore, an assumption is made that there is no full access to all the values of r. On the other hand, it is assumed that a comparison can be made between the available relevance values to determine which one is more relevant. Furthermore, if the relevance value is not available for a given pair of images then its value is considered as zero. The reason is that most of the images are not relevant to one another. The aim is to learn a Similarity function $S(x^{(i)}, x^{(j)})$ in such a manner that the pair having more relevant images are assigned higher scores:

$$\begin{aligned} & S\left(x^{(i)}, {x^{(j)}}^+\right) && > S\left((x^{(i)}, {x^{(j)}}^-\right), \quad \forall x^{(j)}, {x^{(j)}}^+, {x^{(j)}}^- \in \mathcal{R} \\ \text{such that } & r((x^{(i)}, {x^{(j)}}^+) && > r((x^{(i)}, {x^{(j)}}^-) \end{aligned}$$

A parametric similarity function S having a bi-linear form is considered as follows:

$$S_W(x^{(i)}, x^{(j)}) \equiv {x^{(i)}}^t W x^{(j)}$$

where $W \in \mathcal{R}^{d \times d}$. The idea is to find a function S in such a way that all of the triplets obey the following inequality:

$$S_W(x^{(i)}, {x^{(j)}}^+) > S_W(x^{(i)}, {x^{(j)}}^-) + 1$$

where 1 represents the value of the safety margin. The hinge loss for a triplet can be calculated in the following manner:

$$l_W(x^{(i)}, {x^{(j)}}^+, {x^{(j)}}^-) = \max\left\{0, 1 - S_W(x^{(i)}, {x^{(j)}}^+) + S_W(x^{(i)}, {x^{(j)}}^-)\right\}$$

The goal is to minimize the global or the cumulative loss L_W over all of the possible triplets:

$$L_W = \sum_{(x^{(j)}, {x^{(j)}}^+, {x^{(j)}}^-) \in \mathcal{R}} l_W(x^{(i)}, {x^{(j)}}^+, {x^{(j)}}^-)$$

Passive Aggressive algorithm [23] is applied in an iterative fashion to optimize W where W is initialized to $W_0 = I$. At each iteration i, a triplet is selected randomly before solving the following convex problem having a soft margin:

$$W_i = \arg\min_W \frac{1}{2}\|W - W_{i-1}\|_2^2 + C\xi$$

$$\text{such that} \quad l_W(x^{(i)}, x^{(j)^+}, x^{(j)^-}) \leq \xi \text{ and } \xi \geq 0$$

The online update for W closely resembles that of PA-I and can be written as:

$$W_i = W_{i-1} + \tau_i V^i$$

where

$$\tau_i = \min\left\{C, \frac{l_{W_{i-1}}(x^{(i)}, x^{(j)^+}, x^{(j)^-})}{\|V^i\|^2}\right\}$$

$$\text{and} \quad V^i = \left[x_1^{(i)}(x^{(k)^+} - x^{(j)^-}), \cdots, x_d^{(i)}(x^{(k)^+} - x^{(j)^-})\right]^t$$

Furthermore, loss bounds have been provided for *OASIS* based on the one given for the *passive aggressive* algorithms. This method is tested on Google proprietary data and found to be faster even than the fast implementation of LMNN by Weinberger et al. [111]. *OASIS* was also tested with symmetric as well as PSD matrices. In order to enforce symmetry, W is projected onto the set of symmetric matrices W' in the following manner:

$$W' = sym(W) = \frac{1}{2}(W^t + W)$$

However, adding symmetry did not improve the results. For the PSD projection, two different strategies were employed: projecting after every i iterations and projecting only once the training is completed. It was found out that the best performance can be achieved by projecting into PSD after learning.

Local Distance Metric Learning

As opposed to global distance metric learning algorithms where the aim is to optimize compactness and separability in a global fashion, local distance metric learning algorithms try to optimize local compactness and local separability. In general, most works in distance metric learning learn global distance functions which keep all points belonging to the same class nearer while the points pertaining to different classes are separated. In case the classes have multimodal distributions, it becomes very difficult to satisfy the two goals (within-class compactness and between-class separability) simultaneously as shown in figure 2.7 [118].

In local distance metric learning, the focus shifts on the *local* pairs where the pairs belonging to the same mode of a class are brought nearer while the nearby pairs from different classes are separated. Yang et al. [118] have presented a probabilistic framework in order to learn the local constraints.

Using the notations defined for global metric learning algorithms, the probability of making the right prediction for a test example x (denoted by $Pr(+|x)$) can be defined in the following

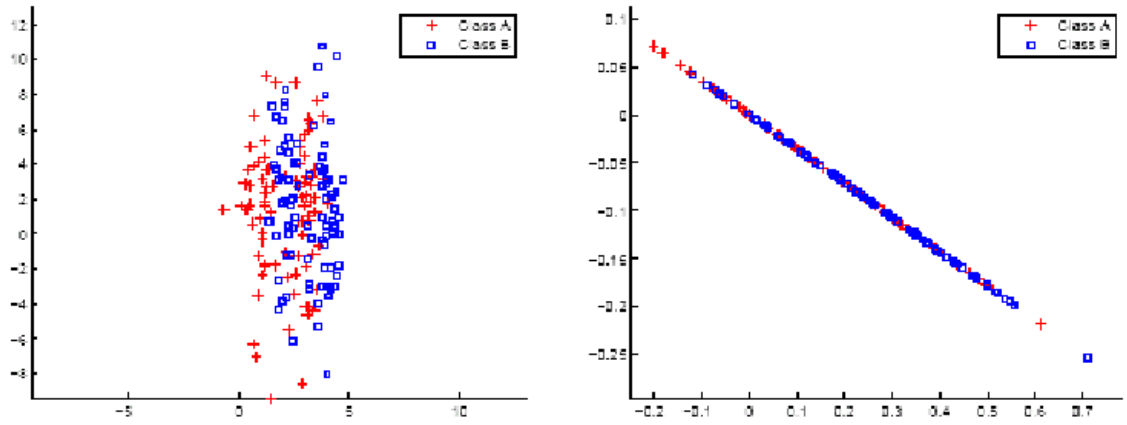

Figure 2.7: Original data distribution (left) and data distribution adjusted by a global distance function(right)

manner:

$$\Pr(+|x) = \frac{\sum_{x^{(i)} \in \phi_S(x)} f(x, x^{(i)})}{\sum_{x^{(i)} \in \phi_S(x)} f(x, x^{(i)}) + \sum_{x^{(j)} \in \phi_D(x)} f(x, x^{(j)})}$$

where S represents the equivalence constraints, D stands for the inequivalence ones and $f(x, x')$ is a kernel function which can be defined as:

$$f(x, x') = \exp(-\|x - x'\|_A^2)$$

The log likelihood for S as well as for D can be written as:

$$\mathcal{L}_l(A) = \sum_{x \in \mathcal{T}} \log \Pr(+|x)$$

where $\mathcal{T}$ represents all of the data points present in the sets S and D. Using the maximum likelihood principle, the local distance problem can be written in terms of the following optimization problem:

$$\max_{A \in \mathbb{R}^{d \mathrm{X} d}} \quad \mathcal{L}_l(A)$$
$$\text{such that} \quad A \succeq 0$$

It may be noted that when an example $x^{(i)}$ is relatively far from x compared to other examples in $\phi_S(x)$ and $\phi_D(x)$, the kernel value $f(x, x^{(i)})$ will be smaller than the kernel values for other examples (since the kernel value between two examples in inversely proportional to the distance between them). This explains the fact that the examples that are distant from each other would have a lesser impact on the objective function $\mathcal{L}_l$ as compared to the ones which are close to one another.

Another locally adaptive distance metric learning algorithm is used in Hastie and Tibshirani [48]. However, in this case, the locality must be specified in advance which is a difficult task.

2.3.2 Similarity Metric Learning

Similarity is a quantity that reflects the strength of relationship between two objects. It normally has the values in the range of either -1 to $+1$ or the values are normalized into 0 to 1. One of the widely used similarities is cosine similarity. The cosine similarity between term frequency-inverse document frequency (tf-idf) vectors is used in information retrieval and text mining for document classification. It has also been demonstrated to be a useful measure in gene express profiling. The similarity between two examples $x(x_1, x_2, \cdots, x_d)$ and $x'(x'_1, x'_2, \cdots, x'_d)$, with angle Θ can be calculated utilizing cosine function as given in the equation:

$$\text{sim}(x, x') = \cos\Theta = \frac{x^t x'}{\|x\|\|x'\|} = \frac{x_1 x'_1 + x_2 x'_2 + \cdots + x_d x'_d}{\sqrt{x_1^2 + x_2^2 + \cdots + x_d^2}\sqrt{x'^2_1 + x'^2_2 + \cdots + x'^2_d}}$$

This ratio defines the cosine angle between the two vectors where $\|.\|$ represents the Euclidean norm of an example. Furthermore, it can be noted that $sim(x, x') = 1$ if and only if $x = x'$, that means the x and x' refer to the same example. And $sim(x, x') = 0$ if and only if $x \perp x'$, that means the x and x' share nothing in common (in case of documents, this means that x and x' share no words at all). With the decrease in the angle between the vectors, the value of cosine approaches 1, meaning that the vectors are getting closer and the similarity is increasing. This ratio can be used as a similarity measure between any two vectors representing documents, queries, snippets, images or a combination of these. In Vector Space Model (VSM), x and x' can be replaced by a document $d^{(i)}$ and a query $q^{(j)}$ to calculate the similarity between a query $q^{(j)}$ and the list of documents ranked based on their similarity with the given query. A good thing about cosine similarity is that it is already normalized. Since the examples are already normalized to unit length, the cosine similarity degenerates to the inner product:

$$\text{sim}(x, x') = x^t x'$$

Threshold Learning

Yang et Liu [119] have proposed a variant of *kNN* algorithm, in which a class specific threshold $b^{(j)}$ is learned using a validation set of examples. Cosine similarity has been chosen and this method has been applied for text categorization in order to find the similarity between two documents. The decision rule for a test document x with respect to the category $c^{(j)}$ can be written as:

$$p(x, c^{(j)}) = \sum_{d^{(i)} \in kNN} \text{sim}(x, d^{(i)})\, p(d^{(i)}, c^{(j)}) - b^{(j)}$$

where $sim(x, d^{(i)})$ is the cosine similarity between a test document x and a training document $d^{(i)}$ (one of the k nearest neighbors of document x); $p(d^{(i)}, c^{(j)})$ is the classification for document $d^{(i)}$ with respect to category $c^{(j)}$ (1 if it belongs to the category or 0 otherwise). Apart from learning category specific thresholds, a similarity matrix is not learned and cosine is rather used

in its original setting. A cross-classifier comparison has also been performed between SVM, kNN, Linear Least Squares Fit (LLSF), Neural Network (NNet) and Naïve Bayes (NB) algorithms. The results show that the kNN performs better as compared to LLSF, NNet and NB but is outperformed by SVM for the micro-level analysis. On the other hand, the macro-level analysis indicate that the performance of SVM, kNN and LLSF are comparative and is better than NB and NNet approaches.

Neural Network Based Similarity Metric Learning

Artificial Neural networks (ANN) have been used both in supervised (e.g. classification) as well as unsupervised settings (self-organizing maps). Diligenti et al. [32] have tried to learn similarities based on a set of comparisons between pairs of examples while using multi-layer perceptron (*MLP*). The key idea is to have a mapping where the similar examples are closer in the output space while at the same time the dissimilar examples are far apart.

Mellaci et al. [72] as well as Maggini et al. [68] have learned similarities as opposed to distances using neural networks. More specifically, a feed-forward multi-layer perceptron (*MLP*) has been employed. A *MLP* is a modification of the linear perceptron with three or more layers (input, output and one or more hidden) of neurons or nodes. This technique is termed as a similarity neural network (*SNN*) whereby a non-negative and symmetric function is learned.

The training phase is based on dyadic supervisions (similar or dissimilar). The *SNN* is made up of a single hidden layer with all the hidden neurons fully connected with the input and output layers. Furthermore, backpropagation algorithm is used to finetune the system with the following properties:

1. The similarity (sim) or the output range is $[0, 1]$ guaranteed by the use of sigmoidal function,
2. The similarity between two examples $x^{(i)}$ and $x^{(j)}$ is symmetric i.e. $sim(x^{(i)}, x^{(j)}) = sim(x^{(j)}, x^{(i)})$,
3. Similarity is not a metric since $sim(x^{(i)}, x^{(i)}) = 1$ and the triangle inequality cannot be guaranteed.

SNN was evaluated on UCI datasets [36] (*Iris*, *Balance* and *Wine*) using similar pairs (pairs belonging to the same class) and dissimilar ones (pairs pertaining to different classes). It was compared with Euclidean and Mahalanobis distances using the cumulative neighbor purity index which measures the percentage of correct neighbors up to the k-th neighbor, averaged over the entire data set.

Similarity Based Classification

Bernal et al. [5] have developed a similarity based classification algorithm (*SBC*) in which the concept of maximal margin has been replaced, which is basically a binary concept, by a concept of robustness of the decision function that is independent of the number of classes. Effectively the replaced concept is equivalent to the maximal margin in the binary case.

Given a set of n class-labeled training objects $(x^{(i)}, y^{(i)})$, $i = 1, \cdots, n$, where $y^{(i)}$ represents the class of the example $x^{(i)}$, and for an unclassified object x', the class similarity of x' is defined with respect to a class C in the following manner:

$$S_C(x') = \sum_{x^{(k)} \in C} \alpha_k \; sim(x^{(k)}, x') \tag{2.5}$$

where $sim(,)$ is the similarity function and $\alpha_k \geq 0$ shows the relative importance given to each $x^{(k)}$ with respect to classification. Thus, the class of x' can be predicted using the following function:

$$C(x') = \arg_C\{\max(S_C(x'))\} \tag{2.6}$$

From equation 2.6, a stronger version can also be derived, which requires that not only x' is more similar to class C than any other class, but is also more similar to class C than it is to the union of *any* other collection of classes. The stronger rule can be written as:

$$C(x') = \arg_C\{\max(S_C(x') > \sum_{D \neq C} S_D(x'))\}$$

Moreover, in order to compare this algorithm with classical machine learning ones that deal with binary classification, the case of only two classes A and B is also considered. Thus the equation 2.5 can be written in another way:

$$\begin{aligned} S_A(x') - S_B(x') > 0 \quad &\Rightarrow C(x') = A \\ S_A(x') - S_B(x') < 0 \quad &\Rightarrow C(x') = B \\ S_A(x') - S_B(x') = 0 \quad &\Rightarrow C(x') \text{ is not defined} \end{aligned}$$

For the similarity measures, Radial Basis functions (RBF) and polynomial kernels have been selected. RBF calculates the distance between two points using the formula:

$$s(x, x') = exp(\frac{\|x - y\|^2}{2\sigma^2})$$

The similarity matrix is defined as:

$$S = [\delta \, s(x^{(i)}, x^{(j)})]$$

where $i, j \in n$, $\delta = 1$ iff $C(x^{(i)}) = C(x^{(j)})$, and $\delta = -1$ otherwise.

Some other Similarity Metric Learning methods

Grabowski et al. [46] have described a method for learning similarities on complex structures where similarity spaces are first learned on elementary domains like the domain of simple attributes etc. This is followed by learning these spaces on approximation spaces, which can be constructed from similarity spaces. The final goal in this case is to design similarities to be used for automated ontology extraction from rich, complex structures. Interestingly, the similarity measure considered is an asymmetric variant of the Jaccard coefficient. However, this approach in general is more inclined towards feature selection than the similarity metric learning.

Another interesting work is the one described by Hust [52], on Collaborative Information Retrieval (CIR), where individual users collaborate to improve the overall Information Retrieval system. Here, a variant of the cosine similarity is learned to re-rank the documents.

Peterson et al. [84] have shown that it is better to use weight-optimized cosine similarity instead of weighted Euclidean distance on UCI collections like *Pima*, *Ionosphere* etc. Genetic Algorithms are employed to improve the performance of *kNN* using weight and offset optimizations. In the case of Euclidean distance, each feature j of an example $x^{(i)}$ is transformed in the following manner:

$$x'^{(i)}_j = x^{(i)}_j * w_j$$

where w represents the weight vector. Euclidean distance is invariant to offset shifting.

Each feature is independently shifted positively or negatively for the cosine measure, thus changing the angular point of reference and ultimately the classification:

$$x'^{(i)}_j = (x^{(i)}_j - O_j) * w_j$$

where O stands for the optimization vector and w for the weight vector.

Pearson correlation is also used, which measures the strength of a linear relationship between two feature vectors $x^{(i)}$ and $x^{(k)}$ in the following manner:

$$\text{Pearson}(x^{(i)}, x^{(k)}) = \frac{\sum_{j=1}^{d}(x^{(i)}_j - \bar{x}^{(i)})(x^{(k)}_j - \bar{x}^{(k)})}{(d-1)SD_{x^{(i)}}SD_{x^{(j)}}}$$

where $\bar{x}$ is the mean value of the example x whereas SD_x is its standard deviation. The range of pearson correlation is $[-1, +1]$. $+1$ indicates a strong positive linear relationship while -1 represents strong inverse linear relationship. On the contrary, the cosine similarity is never negative.

Furthermore, Stahl et al. [100] have learned local similarity measures instead of global ones where the similarities are computed between individual attributes using an evolution program which is a special form of genetic algorithm. There are still some other approaches in which the terms distance and similarity are used in the same context (e.g. the work of Chen et al. [17]).

Mandl [70] use neural networks to learn a similarity matrix based on the similarity between documents and queries. Liu et al. [66] describe an algorithm whereby a similarity metric is learned in non-orthogonal space such that the similarity of features affect the similarity of objects, and vice versa.

2.4 How to use the best features for a dataset

In general, the features of a dataset are either reduced to a set of more meaningful ones or feature reweighting techniques are used. However, there are some other situations in which the different features of a dataset have different scales and the scale effects must be removed in order to use the attributes in an effective manner.

2.4.1 Dimensionality Reduction

In many practical cases, the number of features or the dimensions must be reduced to improve the performance of the classifier. This is particularly the case when many of the features are

irrelevant or redundant. In these cases, the aim is to reduce the dimensionality of the vector space from d to d' where $d' \ll d$. This can be exploited to vastly reduce the storage and search time requirements for *kNN* algorithm. Moreover, by choosing $d' = 2$ or $d' = 3$, one can compute low dimensional visualizations on labeled datasets using a linear projection [42]. The matrix $\mathcal{L}$ in equation 2.1 is considered to be non square of size $d' \mathrm{X} d$. It has been further argued that by using this matrix $\mathcal{L}$, the computational load of *kNN* can be reduced to quite a large extent by restricting the metrics to be those of rank at most d'. Figure 2.8 shows how Goldberger's *Neighborhood Component Analysis (NCA)* algorithm outperforms *PCA* (Principal Component Analysis) and *LDA* (Linear Discriminant Analysis) when the data is visualized in 2 dimensional space. There are two broad categories of feature selection methods: local dimension reduction and global dimension reduction. In local dimension reduction methods, the number of dimensions is reduced separately at each of the query points. On the other hand, in the case of global methods, the original feature space is converted into an optimally chosen subspace with lesser number of features [49].

Partridge and Calvo [80] have defined a fast and simple algorithm where they calculate the approximate principal components (PCs) of a dataset before reducing its dimensionality.

2.4.2 Feature Reweighting

The feature reweighting algorithms learn the weights of the attributes. *RELIEF* (originally proposed by Kira and Rendell [57]) is a simple yet an effective online feature reweighting algorithm. Unlike many other heuristic measures for estimating quality of the attributes, the conditional independence of the attributes is not assumed. Since its development, many people have modified and extended this algorithm (ReliefF, RReliefF, I-Relief etc.) It has been proven successful in many different settings. It learns a vector of weights (for each of the features) describing the importance or quality of different attributes or features.

It has been shown that it solves convex optimization problem while maximizing a margin-based objective function using k-NN algorithm. The weights are updated based on the nearest hit (nearest example belonging to the class under consideration or sometimes referred to as the nearest target neighbor) and the nearest miss (nearest example belonging to other classes).

RELIEF learns only a diagonal matrix in the original setting. However, Sun et al. [102] have extended *RELIEF* to learn a full distance matrix. It has been further proved that Relief is an online algorithm. *RELIEF* outperformed standard *kNN* algorithm on standard UCI collections like Banana, Splice, Waveform etc.

Let $x^{(i)}$ be a vector in R^d having $y^{(i)}$ as the class label with values $+1, -1$. Let w be a vector meant for iteratively estimating the qualities of attributes initialized with 0. The aim is to learn w on a set of training examples. Suppose an example $x^{(i)}$ is randomly selected. This is followed by finding two nearest neighbors of $x^{(i)}$: one from the same class (termed as *nearest hit* or H) and other from the different class than that of $x^{(i)}$ (termed as *nearest miss* or M). The update rule in case of Relief doesn't depend on any condition and can be represented as:

$$w_l = w_l - \frac{\mathrm{diff}(l, x^{(i)}, H(x^{(i)})}{J} + \frac{\mathrm{diff}(l, x^{(i)}, M(x^{(i)}))}{J} \tag{2.7}$$

where J represents the number of iterations, the algorithm has been run while *diff* is a function used to find the difference between the values of an attribute i for $x^{(i)}$ and the nearest hit or miss

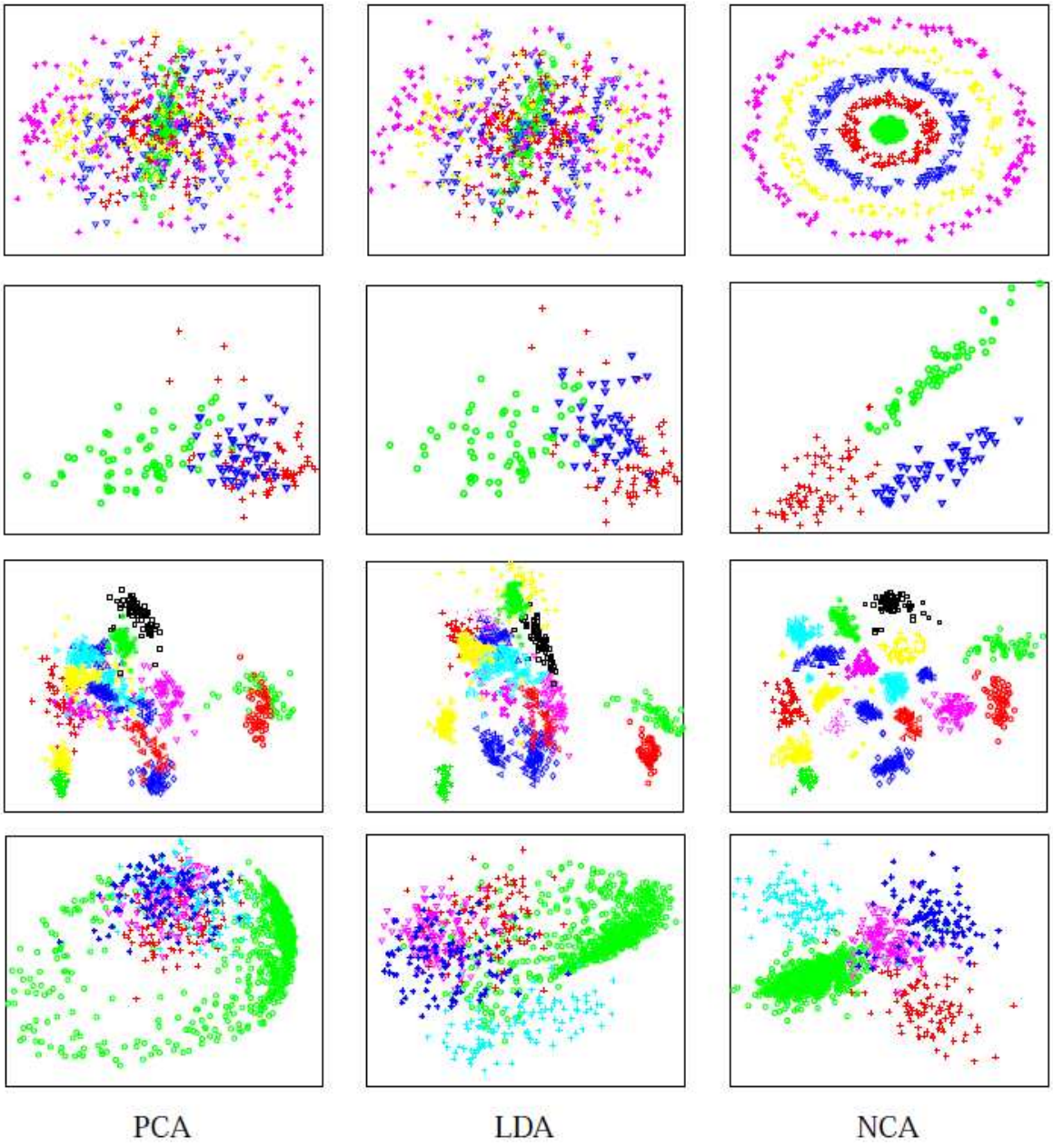

Figure 2.8: Dataset visualization results for PCA, LDA and NCA applied to *concentric rings*, *wine*, *faces* and *digits* (Top to bottom). The datasets are reduced to 2 dimensions in each case. [42]

represented by H or M. If the instances $x^{(i)}$ and H have different values for an attribute i then this means that it separates the two instances in the same class which is certainly not desirable, so the quality estimation w_l is decreased. Similarly if the instances $x^{(i)}$ and M have different values for an attribute i then this attribute separates two instances belonging to different classes which is desirable, so the quality estimation for i is increased. In the case of discrete attributes, the value of difference is either 1 (the values are different) or 0 (the values are the same). However, for continuous attributes, the difference is the actual difference normalized to the closed interval $[0, 1]$ which is given by:

$$\text{diff}(l, x, x') = \frac{|x_l - x'_l|}{max(l) - min(l)}$$

Furthermore, the same *diff* function is used to find the nearest hit and the nearest miss as well, where the total distance is the sum of differences for all of the attributes (Manhattan distance).

The overall aim is to learn the estimation of the qualities of attributes.

The complexity of Relief is $O(Jdn)$ where J is the number of iterations, d is the number of features, and n represents the total number of instances. However, the complexity is fixed for all of the scenarios.

In the original setting, *RELIEF* can only deal with binary class problems and cannot work with incomplete data. In order to cope with this problem, it was extended in the form of *RELIEFF* algorithm [58]. Instead of just finding the nearest hit and miss, it finds k nearest hits and the same number of nearest misses from each of the different classes.

Mathematical Interpretation for *RELIEF* algorithm

Sun and Wu [102] have provided a mathematical interpretation for the *RELIEF* algorithm. The margin for an instance $x^{(i)}$ can be defined as:

$$p_i = d(x^{(i)} - M(x^{(i)})) - d(x^{(i)} - H(x^{(i)}))$$

where $M(x^{(i)})$ and $H(x^{(i)})$ are the nearest miss and nearest hit for $x^{(i)}$ respectively, and $d(.)$ represents a distance function. $d(x) = \sum_l |x_l|$ is defined just like the one used in original *RELIEF* algorithm. The margin is positive only if $x^{(i)}$ is nearer to the nearest hit as compared to the nearest miss, or in other words, is classified correctly as per the *1NN* rule. The aim is to scale each feature in such a way that the leave-one-out error $\sum_{i=1}^{n} I(p_i(w) < 0)$ is minimized, where I(.) is the indicator function and $p_i(w)$ is the margin of $x^{(i)}$ with respect to w. As the indicator function is not differentiable, a linear utility function has been used so that the averaged margin in the weighted feature space is maximized:

$$\arg\max_{w} \sum_{i=1}^{n} p_i(w) = \sum_{i=1}^{n} \left\{ \sum_{l=1}^{d} w_l \left| x_l^{(i)} - M_l(x^{(i)}) \right| - \sum_{l=1}^{d} w_l \left| x_l^{(i)} - H_l(x^{(i)}) \right| \right\}, \tag{2.8}$$

$$\text{such that } \|w\|_2^2 = 1, \text{ and } w \geq 0,$$

where $w \geq 0$ makes sure that the learned weight vector induces a distance measure. The equation 2.8 can be simplified by defining:

$$z = \sum_{i=1}^{n} (|x^{(i)} - M(x^{(i)})| - |x^{(i)} - H(x^{(i)})|$$

and the simplified equation can be written as:

$$\max_{w} \; w^t z \;\text{ where }\; \|w\|_2^2 = 1, w \geq 0$$

The Lagrangian of the above equation can be written as:

$$L = -w^t z + \lambda(\|w\|_2^2 + 1) + \sum_{l=1}^{d} \theta_l(-w_l)$$

where both λ and $\theta \geq 0$ are Lagrangian multipliers. In order to show that the optimum solution can be calculated in a closed form, the following steps are performed: the derivative of L is taken with respect to w and is set to zero. This gives:

$$\frac{\partial L}{\partial w} = -z + 2\lambda w - \theta = 0 \;\text{ and }\; w = \frac{z+\theta}{2\lambda}$$

This is followed by deriving the closed form solution for w. In order to prove that $\lambda > 0$, it is supposed that $z_i > 0$. This implies that $z_i + \theta_i > 0$. In case $\lambda < 0$, then this means that w_i is negative, which contradicts the constraint $w \geq 0$. Therefore, it can be deduced that λ is always positive.

Different cases for z_i could be further verified using the *Karush-Kuhn-Tucker* condition ($\sum_i \theta_i w_i = 0$):

1. When $z_i = 0$, $\theta_i = 0$ and $w_i = 0$;
2. When $z_i > 0$, $z_i + \theta_i > 0 \;\Rightarrow\; w_i > 0 \Rightarrow \theta_i = 0$; and
3. When $z_i < 0$, $\theta_i < 0 \;\Rightarrow\; w_i = 0 \;\Rightarrow\; z_i = -\theta_i$

The optimum solution can be calculated in a closed form in the following manner:

$$w = \frac{(z)^+}{\|(z)^+\|_2} \tag{2.9}$$

where $(z)^+ = [max(z_1, 0), \cdots, max(z_d, 0)]^t$. While comparing the above equation with that of weight update rule for *RELIEF*, it can be noted that *RELIEF* is an online algorithm to solve the optimization problem given in equation 2.8. This is true except when $w_i = 0$ for $z_i \leq 0$ which is normally related to irrelevant features.

In the original setting, *RELIEF* algorithm uses only a diagonal matrix. Sun and Wu [102] have instead used a full distance matrix in which case the optimization problem can be written as:

$$\begin{aligned} &\max_{w} \textstyle\sum_{i=1}^{n} p_i(w) = \sum_{i=1}^{n} m_i^t W m_i - \sum_{i=1}^{n} h_i^t W h_i, \\ &\text{such that } \|W\|_F^2 = 1, \text{ and } W \geq 0, \end{aligned} \tag{2.10}$$

where $m_i = x^{(i)} - M(x^{(i)}), h_i = x^{(i)} - H(x^{(i)})$, and $\|W\|_F$ represents the Frobenius norm of W which can be written as:

$$\sqrt{\sum_{i,j} w_{i,j}^2} = \sqrt{\sum_i \lambda_i^2}$$

Here λ_i stands for the ith eigenvalue for W. It is to be noted that equation 2.8 and 2.10 have similar meanings. Furthermore, W, being a distance function is symmetric and positive, semi-definite.

The performance of a classifier can be enhanced using feature transformation mechanisms. Two commonly used ones are feature standardization and feature fuzzification.

2.4.3 Feature Standardization

It is a process used to remove the scale effects when different features have different measurement scales [83]. The raw feature values are transformed into z-scores using the mean and standard deviation of feature values over all of the samples. The z-score for ith sample and jth feature can be written as:

$$z_{ij} = \frac{x_j^{(i)} - \mu j}{\sigma_j}$$

where $x_j^{(i)}$ is the value for ith sample and jth feature or attribute, μ_j represents the average of all $x_j^{(i)}$ for feature j and σ_j stands for the standard deviation of all $x_j^{(i)}$ over all of the input examples. In case the feature values represent a Gaussian distribution, then the histogram for the z-scores represent a normal distribution having zero mean and the variance of unity. Once the standardization has been performed, the range and scale of the z-scores would be similar.

2.4.4 Feature Fuzzification

This technique exploits the uncertainty in feature values so as to increase the classifier performance [83]. The original feature values are replaced by a mapping into 3 fuzzy sets representing linguistic membership functions in order to facilitate the semantic interpretation of each fuzzy set. The fuzzification process starts by determining x_{min} and x_{max} as the minimum and maximum values of $x_j^{(i)}$ for feature j over all of the input samples i and q_1 and q_2 as the quantile values of $x_j^{(i)}$ at the 33rd and 66th percentile respectively. This is followed by computing the following averages:

$$\begin{aligned} \text{Avg}_1 &= \tfrac{x_{min}+q_1}{2} \\ \text{Avg}_2 &= \tfrac{q_1+q_2}{2} \\ \text{Avg}_3 &= \tfrac{q_2+x_{max}}{2} \end{aligned}$$

The next step is to translate each value of $x_j^{(i)}$ for feature j into 3 fuzzy membership values having the range $[0, 1]$ as $\mu_{low,i,j}$ $\mu_{med,i,j}$ $\mu_{high,i,j}$ using the following relationships:

$$\mu_{low,i,j} = \begin{cases} 1 & x < \text{Avg}_1 \\ \frac{q_2 - x}{q_2 - \text{Avg}_1} & \text{Avg}_1 \leq x < q_2 \\ 0 & x \geq q_2, \end{cases}$$

$$\mu_{med,i,j} = \begin{cases} 0 & x < q_1 \\ \frac{\text{Avg}_2 - x}{\text{Avg}_2 - q_1} & q_1 \leq x < \text{Avg}_2 \\ \frac{q_2 - x}{q_2 - \text{Avg}_2} & \text{Avg}_2 \leq x < q_2 \\ 0 & x \geq q_2, \end{cases}$$

$$\mu_{high,i,j} = \begin{cases} 0 & x < q_1 \\ \frac{x - q_1}{\text{Avg}_3 - q_1} & q_1 \leq x < \text{Avg}_3 \\ 1 & x \geq \text{Avg}_3. \end{cases}$$

The computations for $\mu_{low,i,j}, \mu_{med,i,j}$ and $\mu_{high,i,j}$ give 3 fuzzy sets or vectors $\mu_{low,j}$ $\mu_{med,j}$ $\mu_{high,j}$ of length n which replace the original input feature.

2.5 Classifier Comparison Techniques

The performance of different classifiers can be compared based on many different metrics. The most widely used criterion is accuracy which is the number of correct classifications to the total number of classifications made. Some of the other criterions are precision, which is the ratio of the number of relevant objects retrieved to the total number of objects retrieved, and recall, which is measured as the number of relevant objects retrieved, divided by the total number of relevant objects (whether retrieved or not):

$$\text{precision} = P = \frac{\text{Number of relevant objects (or documents) returned}}{\text{Total number of objects (or documents) returned}}$$

$$\text{recall} = R = \frac{\text{Number of relevant objects (or documents) returned}}{\text{Total number of relevant objects (or documents)}}$$

Another standard evolution measure is the F-measure which is a combination of precision and recall, and depends on a parameter α. It can be defined as:

$$\text{F-measure} = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

By choosing $\alpha = 0.5$, same importance is given to precision and recall. In this case, F-measure becomes the harmonic mean of the two values: P and R.

2.5.1 Cross Validation

Cross validation is basically a model evaluation method. There are many different types of cross validation techniques like holdout method, K fold cross validation, leave-one-out cross validation etc.

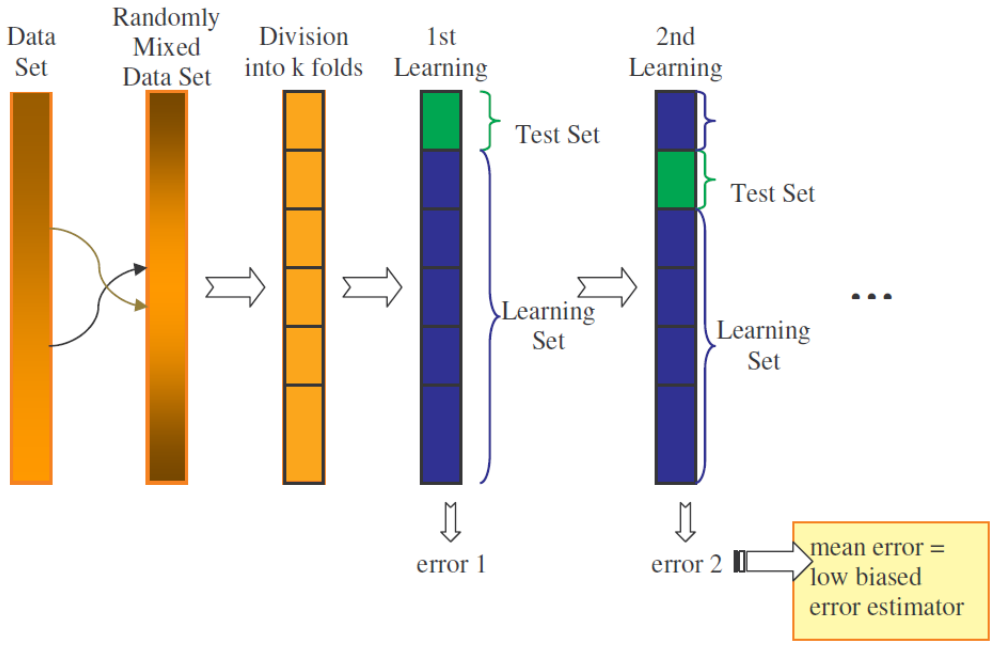

Figure 2.9: Cross validation [101]

The holdout method is the simplest of all cross validation methods. In this method, the data set is divided into training and test set. The algorithm is trained on the training set and the performance is assessed on the test set. The benefit of this method is that it requires much less time to execute. However, the evaluation is dependent on the distribution of examples into training set and the test set and it may have a high variance. In K fold cross validation, the dataset is presented K times to the classifier [120] as shown in the figure 2.9. The training is done on $\frac{K-1}{K}$ of the samples while the rest of $\frac{1}{K}$ samples are used as a test set. At the end, the average error across all K trials is found. One of the key advantages of this method is that it hardly matters how the data is divided. Every example is selected once in the test set while $K-1$ times for the training set. The disadvantage of this approach is that the training algorithm has to be executed for K times, consequently increasing the computation cost by K times.

Leave-one-out cross validation is equivalent to K fold cross validation with K chosen to be equal to n, the number of examples in the training set. This implies that the algorithm is run for n times, each time training on $n-1$ examples and testing on the only example which was left. In this approach also, the average error is found to evaluate the performance of the algorithm.

2.5.2 Significance Tests

Two systems or classifiers can be compared based on significance tests which can be broadly classified into two sub categories: micro level tests and macro level tests [119]. The micro level tests (e.g. s-test, p-test) are based on decisions on individual document/class pairs. On the other hand, macro level test (e.g. S-test, T-test etc.) is calculated from the performance scores for each category.

A micro sign test, s-test, compares two classifiers, A and B. This test is based on the binary

decisions for all document/class pairs. In order to explain this test, the following notation is used: n represents the total number of binary decisions made by each of the two classifiers, a_i measures the success of classifier A for ith decision ($i = 1, \cdots, n$). Similarly b_i is used to calculate the success for classifier B. The allowed values for a_i and b_i are 0 or 1. Furthermore, m is used to describe the number of times classifier A and classifier B have different classification. k describes the number of times the system A is better than system B i.e. a_i is larger than b_i. The null hypothesis is $k = 0.5m$ which means that 50% of the time classifier A is better than classifier B or in other words k has a binomial distribution $Bin(m, p)$ where $p = 0.5$. Consequently the alternate hypothesis says that k has a binomial distribution with $p > 0.5$. If $m \leq 12$ and $k \geq 0.5m$, the one sided P value can be computed using the binomial distribution:

$$P(Z \geq k) = \sum_{i=k}^{m} \begin{pmatrix} m \\ i \end{pmatrix} * 0.5^m$$

However, if $m \leq 12$ and $k < 0.5m$, P-value of the other extreme can be calculated as follows:

$$P(Z \leq k) = \sum_{i=0}^{k} \begin{pmatrix} m \\ i \end{pmatrix} * 0.5^m$$

The P-value shows the significance level of the observed evidence against the null hypothesis (whether classifier A is better or worse than classifier B).

If m is greater than 12, the P-value can be approximated using the normal distribution:

$$Z = \frac{k - 0.5m}{0.5\sqrt{m}}$$

Apart from micro level significance tests, there are also some macro levels tests e.g. S-test, T-test and T'-test etc. These tests evaluate the systems at a macro level; using the performance scores on each category as the unit measure. Furthermore, the authors have argued that the micro level tests are dominated by the performance of the classifiers on common categories. On the other hand, the macro level tests are more reflective of the performance of the classifiers on rare classes.

2.6 Conclusion

Machine learning studies the mechanisms and methods by which an entity constructs and uses knowledge, with the aim of improving its performance with experience. Machine learning algorithms can be classified into supervised (e.g. *kNN* algorithm, SVMs etc), unsupervised (e.g. clustering) or semi-supervised learning algorithms. The supervised learning is based on learning from labeled examples. On the other hand, unsupervised learning algorithms work without any sort of supervision. Semi-supervised learning lies in between supervised and unsupervised learning in which case the data consists of labeled as well as unlabeled data. There is yet another way in which machine learning algorithms can be distinguished: online vs batch learning. Many of the machine learning algorithms rely heavily on the metric employed. Among the most common ones are Euclidean distance and the cosine similarity. However both of these do not take into account the underlying geometry of the space in which the data lie and hence are not the best

options. This has paved the way for a new research theme known as *metric learning*. Metric learning can be divided into distance metric learning and similarity metric learning. Most of the distance metric learning algorithms are based on learning Mahalanobis distance metric, an extended form of the Euclidean distance e.g. Information Theoretic Metric Learning [28], Large Margin Nearest Neighbor classification [112] etc. However, people have showed that cosine similarity should be preferred over the Euclidean distance on datasets which are not necessarily text ones. In order to select the best features of a dataset for the learning process, various techniques like dimension reduction and feature reweighting techniques (e.g. *RELIEF* algorithm) could be employed. In order to evaluate an algorithm, cross validation techniques could be used. Furthermore, significance tests are used in order to show that a method is significantly better than its counterparts.

Chapter 3

Online and Batch Document Filtering Using An Adaptive Nearest Neighbor Algorithm

Contents

3.1 Introduction

In document filtering, a stream of documents is filtered as per the profiles of various topics. In the absence of any supervision, standard cosine can be found between a document d and the topics as $cos(d, t_i)$, before adding the document to the profile having the greatest cosine similarity. In case, there is some possibility of supervision, the standard cosine can be adapted to learn some parameters related with the cosine similarity. Apart from the similarity between documents and topics, another possible one is between different documents assigned to a particular topic and comes into action only in the presence of some sort of supervision.

In this chapter, a simple filtering method is described whereby the kNN algorithm is adapted to learn similarity thresholds. This represents the first step towards learning the complete similarity metric. The adaptive kNN algorithm is developed in the context of INFILE (INformation FILtering Evaluation) [9, 7] campaign and is based on strong constraints on the similarities between documents and topics and between different documents within a topic.

The INFILE campaign was run as a pilot track of CLEF (Cross Language Evaluation Forum) in 2008 and 2009. It was sponsored by the French National Research Agency (ANR) [7] and was co-organized by the CEA-LIST, ELDA and the University of Lille3-GERiiCO. It extended the *TREC* (Text REtrieval Conference) 2002 filtering track and was basically a cross-language adaptive filtering evaluation campaign where the aim was to successfully separate relevant and non-relevant documents with respect to a given profile, the document and the profile being possibly written in different languages. INFILE used 300,000 Agence France Presse (AFP) comparable newswires covering the years 2004 to 2006 in three languages (100,000 for each): Arabic, English and French. It also included a set of 50 topics in general and specific domain (scientific and technological information). The News articles written in different languages were not necessarily translation of each other, and were encoded in XML format and followed the News Markup Language (NewsML) specifications. NewsML is an XML standard designed to provide a media-independent, structural framework for multi-media news and is developed by International Press Telecommunications Council [8]. The competitors were asked to compare each topic in a source language to the documents in the target languages. Every possible source/target language pair was allowed. The participants had the possibility of participating in the monolingual filtering, cross-lingual filtering (e.g. source language is English and target language is French) or multi-language filtering (with a mixed set of documents from different target languages).

In this chapter, the participation in INFILE 2008 and 2009 is described in detail which covered only the monolingual participation using English language. The goal of the INFILE campaign was to filter 100,000 documents into 50 topics (plus a category 'other'). Out of these 50 topics, 30 were related to general news and events (e.g. national and international affairs, sports, politics etc.), whereas the rest concerned scientific and technical subjects. A document belonged to zero, one or more topics; each topic being described by a set of sentences. The topics or profiles have been created by competitive intelligence (CI) professionals from INIST [9], ARIST

[7] http://www.agence-nationale-recherche.fr/
[8] http://www.newsml.org
[9] The French Institute for Scientific and Technical Information Center, http://international.inist.fr

Nord Pas de Calais [10], Digiport [11] and OTO Research [12]. The profiles were defined with the following structure:

1. a unique identifier
2. a title describing the topic (maximum 6 words)
3. a sentence-long description of the topic (maximum 20 words)
4. a narrative describing which document should be considered as relevant and which should be termed as non-relevant (maximum 60 words)
5. Keywords (maximum 5)
6. an example of relevant text taken from a document not present in the collection (maximum 120 words)

Any of the possible combinations of these tags were allowed for filtering. An example of a topic is given below:

<**top**>
<**num**>110</**num**>
<**title**>The diversity in politics</**title**>
<**desc**>The profile relates to the diversity in politics, the existing provisions to ensure better representation of all social strata</**desc**>
<**narr**>The relevant document should describe the problem of cultural ethnic and social diversity in policy, the parity, lack of visibility of minorities in the political arena, the fight against discrimination, the various means for enabling this diversity, and the main obstacles encountered.</**narr**>
<**keywords**>
<**keyword**>Diversity in politics</**keyword**>
<**keyword**>Fight against discrimination</**keyword**>
<**keyword**>parity</**keyword**>
<**keyword**>visibility of minorities</**keyword**>
<**keyword**>Integration</**keyword**>
</**keywords**>
<**sample**>In the political arena, the term diversity (or diverse) is used to describe political entities (neighborhoods, cities, nations, student bodies, etc.) with members who have identifiable differences in their backgrounds or lifestyles. The use of the term diversity may encompass differences in racial or ethnic classifications, age, gender, religion, philosophy, physical abilities, socioeconomic background, sexual orientation, gender identity, intelligence, mental health, physical health, genetic attributes, behavior, attractiveness, place of origin, cultural values, or political view as well as other identifying features. Political creeds which support the idea that diversity

[10] Regional agency for strategic information and technology, http://www.aristnpdc.org
[11] http://www.digiport.org
[12] http://www.otoresearch.fr

is valuable and desirable hold that recognizing and promoting these diverse cultures may aid communication between people of different backgrounds and lifestyles, leading to greater knowledge, understanding, and peaceful coexistence.[citation needed] For example, "Respect for Diversity" is one of the six principles of the Global Greens Charter, a manifesto subscribed to by Green parties from all over the world. In contrast to diversity, some political creeds promote cultural assimilation as the process to lead to these ends </**sample**>
</**top**>

In comparison with INFILE 2008, where there was only an online task, an additional batch filtering task was added in the year 2009. As opposed to the online task, where the server provides the documents one by one to the user, all of the documents are provided beforehand in the batch task. This chapter describes the participation in the online task of 2008 [14], and the batch one of 2009 [88].

3.2 Document Filtering using An Adaptive Nearest Neighbor Algorithm

Many studies have shown that similarity measures are more appropriate for the kNN algorithm as compared to the distance ones, when dealing with texts (see e.g. [87]). This explains the fact that the cosine measure was chosen for document filtering rather than Euclidean distance.

In order to filter the documents into various topics, a similarity measure between the new documents and topics is employed, along with a set of thresholds on this similarity that evolves over time. The similarity between a new document d, to be filtered, and a topic t_i can be given as:

$$\text{sim}(t_i, d) = \alpha * \underbrace{cos(t_i, d)}_{s_1(t_i,d)} + (1-\alpha)\underbrace{\max_{(d' \neq d, d' \in t_i)} cos(d, d')}_{s_2(t_i,d)} \tag{3.1}$$

where $\alpha \in [0,1]$. The similarity given in equation 3.1 is based on two similarities: one based on a direct similarity between the new document and the topic (given by $s_1(t_i, d)$), and another one between the new document and the set of documents already assigned to the topic ($s_2(t_i, d)$). One might think that only the first similarity would suffice. However, this is not the case since the topics and the documents do not share the same kind of structure and content and hence the significance and interpretation of these two similarities is not the same.

Figure 3.1 [13] shows the range of cosine similarity values for all of the documents with respect to topic 1. It can be observed that most of the documents have the similarity even below 0.025. Furthermore, it was also observed that many of the documents have zero similarity with the topic (i.e. all of the words in the document and the topic are mutually exclusive). Similarly, the maximum value of cosine similarity is 0.487 shared by only two documents (document no. 13460 and 72687). The average similarity value is 0.019.

Nearly the same phenomenon is observed for topic 10 as shown in figure 3.2, except the fact that the maximum value of cosine similarity increases to 0.565 (for document number 48187) in

[13] The scale is different for the two figures since fewer documents have greater cosine similarity values. Hence, as the range of cosine similarity increases, the number of documents appearing in that particular range decreases.

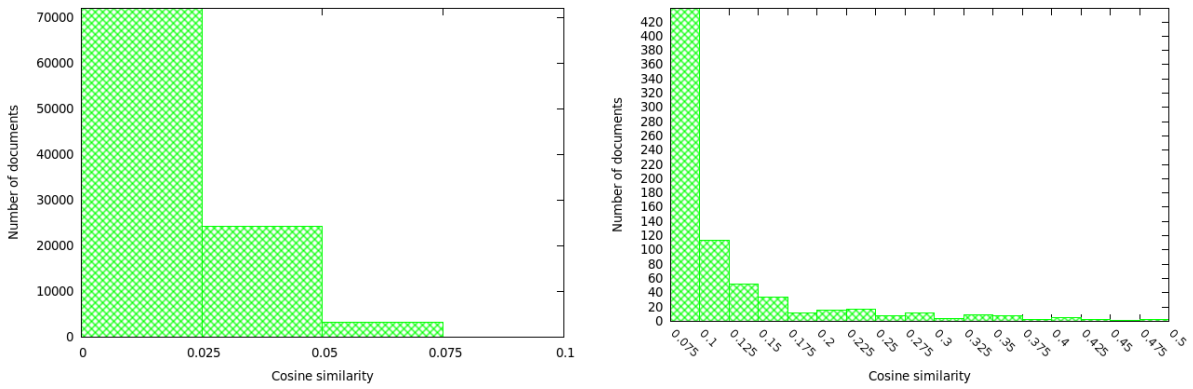

Figure 3.1: Cosine similarity for the 100,000 documents for Topic 1

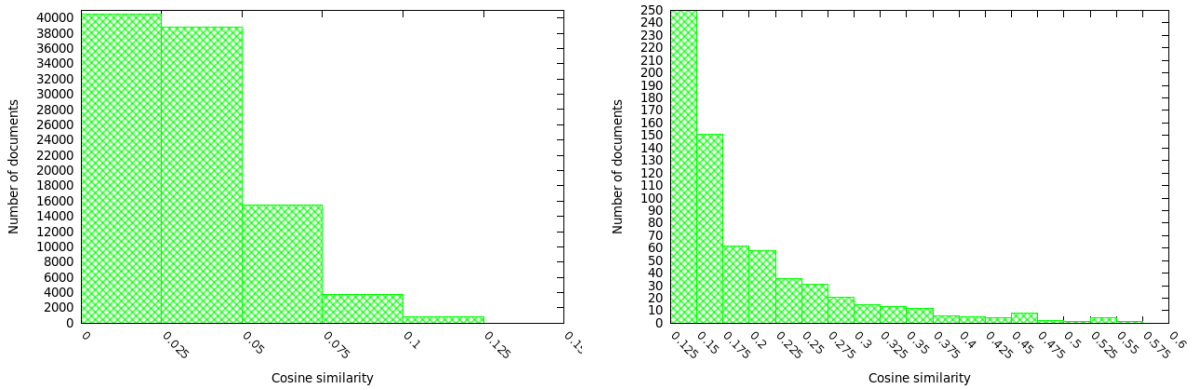

Figure 3.2: Cosine similarity for the 100,000 documents for Topic 10

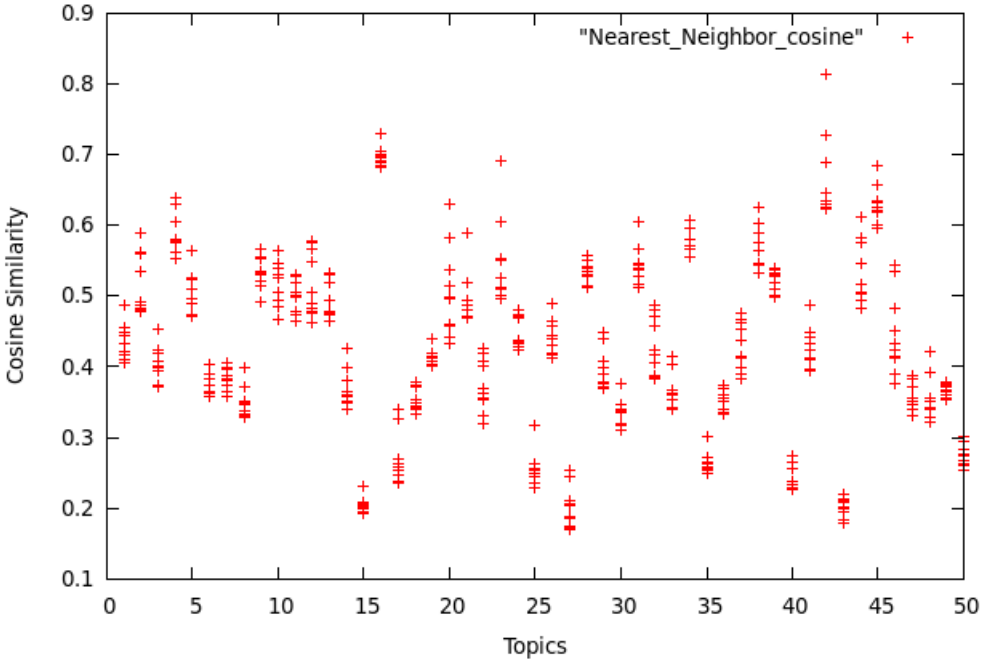

Figure 3.3: Cosine similarity for 10 Nearest Neighbors for all of the Topics

this case. The average similarity also increases to 0.034.

Figure 3.3 gives the values for the cosine similarity for the 10 nearest documents for each of the 50 topics. Most the values lie in the range $0.3-0.6$. The maximum value observed is 0.813 for 42nd topic whereas the minimum value (0.170) is for topic number 27. Here, the average cosine similarity is 0.43. It can also be observed that only a few documents have a cosine similarity less than 0.2, and even a fewer have got cosine similarity greater than 0.7.

The second similarity helps to find the documents which are closer to documents which had already been assigned to a topic. α is used to control the importance of the two similarities. In the beginning, when no documents are assigned to any topic, only the similarity between a topic and the new document, $s_1(t_i, d)$, is taken into account for computing the final similarity between the document and the topic.

The similarity in equation 3.1 can be used for document filtering in an online or batch setting. The two possibilities are discussed in detail.

3.2.1 Online Document Filtering

First, the online document filtering algorithms [15] based on the similarity given in equation 3.1 are described. Two thresholds were introduced for each of the topics, θ_i^1 and θ_i^2:

1. θ_i^1 allows filtering out documents in the early stages of the process (i.e. when only a few documents have been assigned to the topic) and operates only on $s_1(t_i, d)$. It helps to build an initial base of 10 documents per topic using the possible feedback from the server (50 in total for the whole collection of INFILE 2008). The use of feedback limits the assignment of non-relevant documents to the different topics. The threshold θ_i^1 is the value above

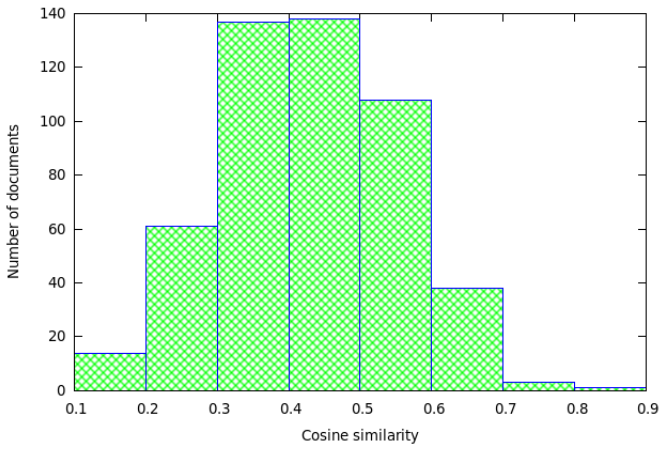

Figure 3.4: Range of cosine similarity between topics and their 10 nearest documents

which, the value of $s_1(t_i, d)$ is considered to be high enough to say that the document d is relevant to topic t_i.

2. θ_i^2 operates on the global similarity, after a certain number of documents have been assigned to the topic. It accounts for the fact that new information has been incorporated in the topic as explained in the algorithm.

The general algorithm for online filtering is summarized:

Online Algorithm (General)

Set α to α_0 and all θ_i^1 to θ_0^1
for each new document d
  for each topic i
    Construction of initial set:
    if ($l_i < NB$)
      if ($s_1(t_i, d) > \theta_i^1$)
        If feedback is possible: Ask for feedback
          $t_i \Leftarrow d$ (only if feedback positive)
        else $t_i \Leftarrow d$
    Assignment of remaining documents to topics:
    else if ($\text{sim}(t_i, d) > \theta_i^2$)
      $t_i \Leftarrow d$
    where $\theta_i^2 = \min_{d \in t_i} \text{sim}(t_i, d)$

where l_i represents the number of documents assigned to a topic i. The parameter α and the threshold θ_i^1 were tuned during the dry run phase which ran before the actual campaign. Two topics and ten documents were provided during the dry run phase. The value chosen for α_0 was 0.7 while that for θ_0^1 was 0.42. It can be recalled from figure 3.3 that the average cosine similarity between the 50 topics and their 10 nearest neighbors is 0.43 and thus very close to θ_0^1.

Once the initial set of documents has been constructed (maximum 10 per topic), the algorithm works to assign the remaining documents to different topics. For each topic i, its corresponding θ_i^2 is initialized with the cosine similarity between the topic and its least similar document. θ_i^2 is updated whenever a new document is added in the topic i.

Simplification of the general online algorithm

In addition to the general version of the online algorithm, a simplified version has also been investigated, which neither uses any feedback nor builds an initial set of documents. It does not update the threshold θ_i^2 unlike the general algorithm. In this version, a threshold θ is derived from θ_i^1 and θ_i^2 according to equation 3.1, which integrates the two similarities θ_i^1 and θ_i^2 operate upon:

$$\theta = \alpha * \theta_i^1 + (1 - \alpha) * \theta_i^2$$

Documents are then filtered according to the following, simple algorithm where the threshold θ replaces θ_i^2 of the online algorithm.

Online Algorithm (Simplified)

Set α to α_0
Assignment of documents to topics:
for each new document d
    for each topic i
        if $(\text{sim}(t_i, d) \geq \theta)$
            $t_i \Leftarrow d$

Here again, values for the different parameters were tuned during the dry run phase. This was followed by slight modifications of these values in the final experiments.

3.2.2 Batch Document Filtering

Here a batch algorithm [89] to filter the documents into various profiles/topics is described. It is also based on the equation 3.1 like the online algorithm. As for the online algorithm, when no documents are assigned to any topic, only the similarity between a topic and the new document, $s_1(t_i, d)$ is considered. This similarity is used to find a certain number of nearest neighbors for each of the document (10 in this case) which eventually helps to use the second similarity. A threshold was used for each of the 50 topics. Feedback is not possible in the case of batch filtering since the complete set of documents is transferred to the user in one go.

Batch Algorithm

Construction of initial set:
for each topic i
    find NB nearest neighbors based on $s_1 = cos(t_i, d)$
    for each nearest neighbor d found
        $t_i \Leftarrow d$

Assignment of remaining documents to topics:
Set α to α_0
for each topic i
    $\theta_i = \min_{d \in t_i} \text{sim}(t_i, d)$
for each document d
    for each topic i
        if $(\text{sim}(t_i, d) \geq \theta_i)$
            $t_i \Leftarrow d$
        $\theta_i = \min(\theta_i, \min_{d \in t_i} \text{sim}(t_i, d))$

Yang and Liu. [119] have described a similar method, whereby they learn category-specific thresholds based on a validation set. An example is assigned to a particular category only if its similarity with the category surpasses a certain learned threshold. In contrary, there is no validation set in this case to learn thresholds. Nevertheless, a simulated one is created by finding nearest neighbors for each of the 50 topics.

3.3 Comparison between Online and Batch Algorithms

A detailed comparison between the batch algorithm used in 2009 and the online algorithms developed for the online campaign in 2008 is discussed.

The main difference between the two algorithms (batch and general online) lies in the manner in which the initial set of documents relevant to the topics is created. In the batch algorithm, only 10 nearest neighbors are found for each topic, with the assumption that the nearest neighbors for a topic would, in general, belong to the topic under consideration. However, for the online algorithm, feedbacks were used (limited to 50) in order to add a document to a profile if the similarity between a topic t_i and a document d is greater than a certain threshold (θ^1). This procedure is repeated until either 10 documents have been added to each of the 50 topics or all of the 100,000 documents have been encountered. Hence it is possible that a certain topic has less than 10 documents after the construction of the initial set. On the contrary, the use of nearest neighbors in the batch algorithm ensures that each topic has exactly 10 documents after the buildup of the initial set.

Furthermore, as the online algorithm builds the initial set of documents based on the threshold θ^1, hence, it is very important that this threshold is chosen very carefully (a dry run was used to tune the value of θ^1 during the online campaign in 2008). On the other hand, the batch algorithm does not use any threshold during the construction of the initial set.

The second phase of the two algorithms, where the remaining documents are assigned to different topics, is similar except the fact that the threshold θ_i in the batch algorithm is updated, only if the current threshold is smaller than the previously stored one. However, the online algorithm does not make use of previously stored value of the threshold θ_i^2. This means that the batch algorithm is more lenient in assigning new documents to topics as compared to the online algorithm.

Comparing the simplified online algorithm with the rest of the two, it can be seen that as

the simplified algorithm does not build an initial set of documents, hence it cannot use $s_2(t_i, d)$ unless some document has been assigned to the topic t_i.

3.4 Evaluation Metrics

The results for the different runs were evaluated based on different measures, namely, precision, recall, F-measure, linear utility, anticipation (added in 2009) and detection cost (see [6] and [7]). The results indicating the association of a document with a profile were in the form of binary decisions. The results for a given profile can be categorized as per the contingency table 3.1. The different metrics can be defined in the following manner:

	Relevant	Not Relevant
Retrieved	a	b
Not Retrieved	c	d

Table 3.1: Contingency Table

Precision is defined as:

$$P = \frac{a}{a+b}$$

Recall is given by:

$$R = \frac{a}{a+c}$$

F-measure, which is a standard combination of precision and recall, and depends on a parameter α is defined as:

$$\text{F-measure} = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

By choosing $\alpha = 0.5$, same importance is given to precision and recall and the F-measure becomes the harmonic mean of the two values: P and R. This means that in order to have a good F-measure, both the precision as well as the recall must be high.

Detection cost was considered in 2008 but not in 2009 since the detection cost values were often low and were not discriminant between different participants. In order to define the detection cost, two measures are considered:

1. The estimated probability of missing a relevant document given by $P_{miss} = \frac{c}{a+c}$
2. The estimated probability of raising a false alarm on non-relevant document given by $P_{false} = \frac{b}{b+d}$

With this, the detection cost can be defined:

$$c_{det} = c_{miss} \times P_{miss} \times P_{topic} \times c_{false} \times P_{false} \times (1 - P_{topic})$$

where c_{miss} is the cost of a missed document, c_{false} is the cost of a false alarm while P_{topic} is the a priori probability that a document is relevant to a given profile. During the INFILE campaign

2008, c_{miss} was chosen to be 10, $c_{false} = 0.1$ while the value of P_{topic} was given the value 0.001 based on the average ratio of relevant documents.

Linear utility is based on two parameters: importance given to a relevant document retrieved (w_1) and the cost of a non-relevant document retrieved (w_2). Linear utility can be written as:

$$u = w_1 * a - w_2 * b$$

Filtering by linear utility is just like filtering by estimated probability of relevance. For example, with $w_1 = 2$ and $w_2 = 1$, it corresponds to the rule: retrieve if P(relevance) > 0.33. A problem with linear utility is that although it is bounded positively, it is unbounded negatively (negative values depend on the number of relevant documents for a profile). Thus, the average over all of the profiles would give much more importance to the few profiles on which the system has performed poorly. In order to average the value, the measure is scaled in the following manner:

$$u_n = \frac{\max(\frac{u}{u_{max}}, u_{min}) - u_{min}}{1 - u_{min}}$$

where u_{max} is the maximum value of the linear utility and u_{min} represents the minimum value below which a user does not consider the following documents for the profile. The values chosen for INFILE 2008 and INFILE 2009 were: $w_1 = 1$, $w_2 = 0.5$ and $u_{min} = -0.5$. The value of u_{min} was the same as that of TREC 2002 campaign.

Anticipation measure is designed to give more importance to systems that can find the first document in a given profile. The interest in this measure is motivated by the fact that in competitive intelligence, everyone wants to be at the cutting edge of the domain and does not want to miss the first information to be reactive. It is calculated by the inverse rank of the first relevant document detected in a list of relevant documents, averaged on all profiles.

3.5 Experiments

The algorithms have been run on the INFILE English corpus. For all of the documents, stemming was performed using Porter's algorithm [56]. This was followed by the removal of stop-words, XML tags skipping and the building of a document vector (which associates each term with its frequency) using the Rainbow package [71]. During the InFILE campaign, three runs were submitted during Online campaign of 2008 while a single run was submitted during the Batch campaign of 2009. All of the topics' fields were used for the filtering process. In the case of Batch algorithm, 10 nearest neighbors were found for each of the document based on the similarity $s_1(t_i, d)$ (between a document and the topic). These documents were subsequently used to compute $s_2(t_i, d)$. The experiment was divided into 4 sub-parts, each sub-part being run in parallel to increase the efficiency. However, this setting meant that the thresholds for the 50 topics were different for the different sub-parts.

There are 1597 documents relevant to one or more topics in the INFILE data. The average number of relevant documents per topic is 31.94 whereas the standard deviation on the number of relevant documents per topic comes out to be 28.45. The repartition of relevant documents across the 50 topics is shown in figure 3.5. The distribution of the relevant documents with respect to different topics is not uniform. On one hand, some topics have a very small number

	Name	Campaign	Algorithm	Doc. ret	Doc. ret - relevant
Run 1	run5G	Online 08	Online (with feedback)	7638	**601**
Run 2	run2G	Online 08	Online (w/o feedback)	1311	411
Run 3	runname	Online 08	Online (w/o feedback)	546	152
Run 4	IMAG_1	Batch 09	Batch (w/o feedback)	5513	413

Table 3.2: Detail about the different runs

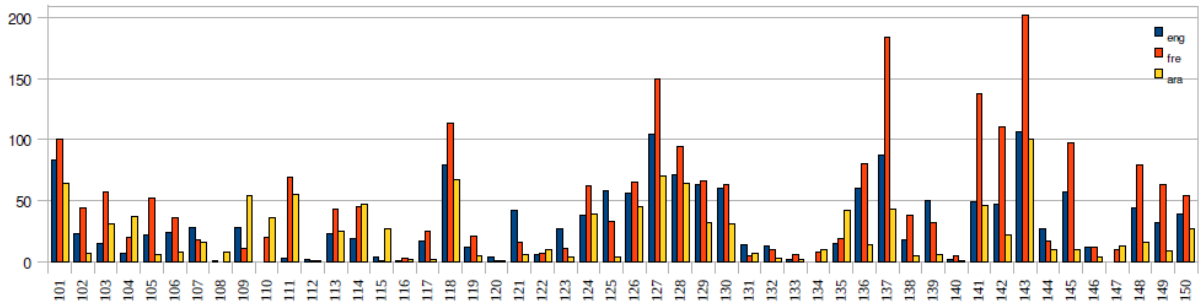

Figure 3.5: Number of relevant documents for each topic in the three languages (English, French and Arabic)

of relevant documents e.g. topic no. 108, 112, 116, 140 etc. On the other hand, some topics like topic no. 127 and 143 have got more than 100 relevant documents. Apart from these two topics, topic no. 101, 118, 125-130, 136, 137, 139, 141, 143 and 145 have got equal to or more than 50 relevant documents.

The general online algorithm and its simplified version developed in 2008 are compared with the batch algorithm of 2009. Table 3.2 describes the different runs along with the number of documents retrieved and the number of relevant documents found. Various measures could be computed like micro precision, micro recall etc. from table 3.2. Run 2 has the highest micro precision whereas Run 1 has got the highest micro recall. These values are computed on the entire corpus.

For Run 2 (run2G), θ^1 was chosen to be 0.45 while θ^2 was set to 0.8. Similarly for Run 3, the values for θ^1 and θ^2 were 0.4 and 0.7 respectively.

Figure 3.6, 3.7, 3.8 and 3.9 give an insight on the number of relevant documents retrieved during the different runs. *num_ret* stands for the number of documents retrieved, *num_rel_ret* describes the number of relevant documents retrieved while *num_rel* is used for the actual number of relevant documents. It is pertinent to mention that the number of relevant documents is not uniformly distributed among the 100,000 documents. Almost one fifth (approximately 300) of the relevant documents lie in the range 90,000-100,000. Another important thing is that the scale is not the same for the different runs. From these two figures, no significant difference can be noticed between Run 2 and Run 4, in terms of the number of documents retrieved during the entire process. However, Run 1 returns much more documents between 10,000-20,000 and 80,000-90,000 documents. Similarly Run 3 retrieves more documents between 10,000-40,000 and 50,000-70,000 documents.

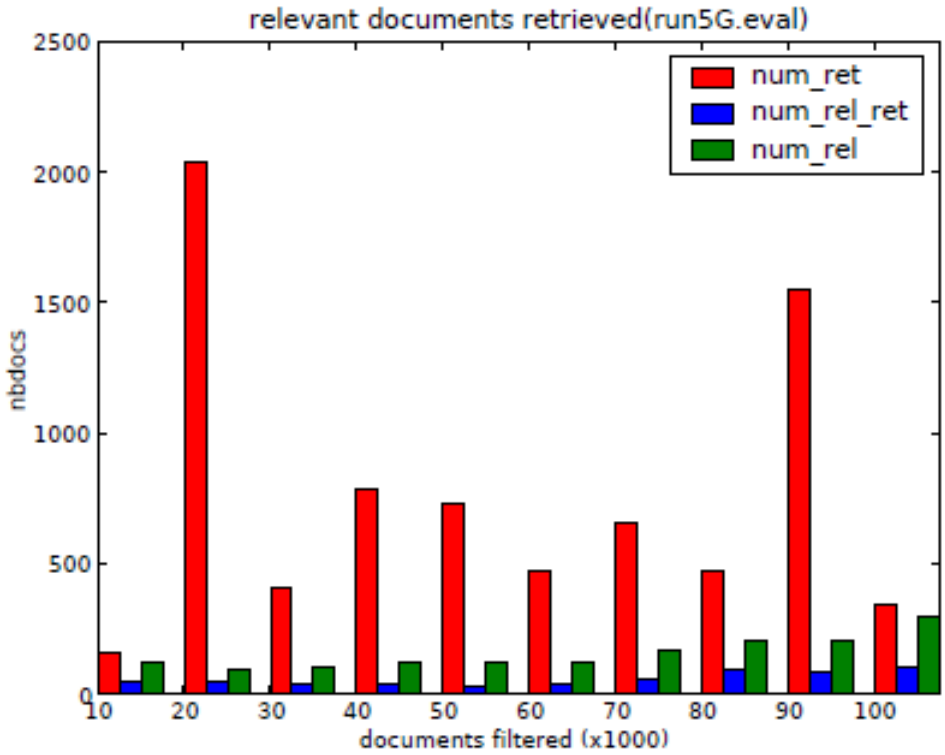

Figure 3.6: Number of documents retrieved for Run 1

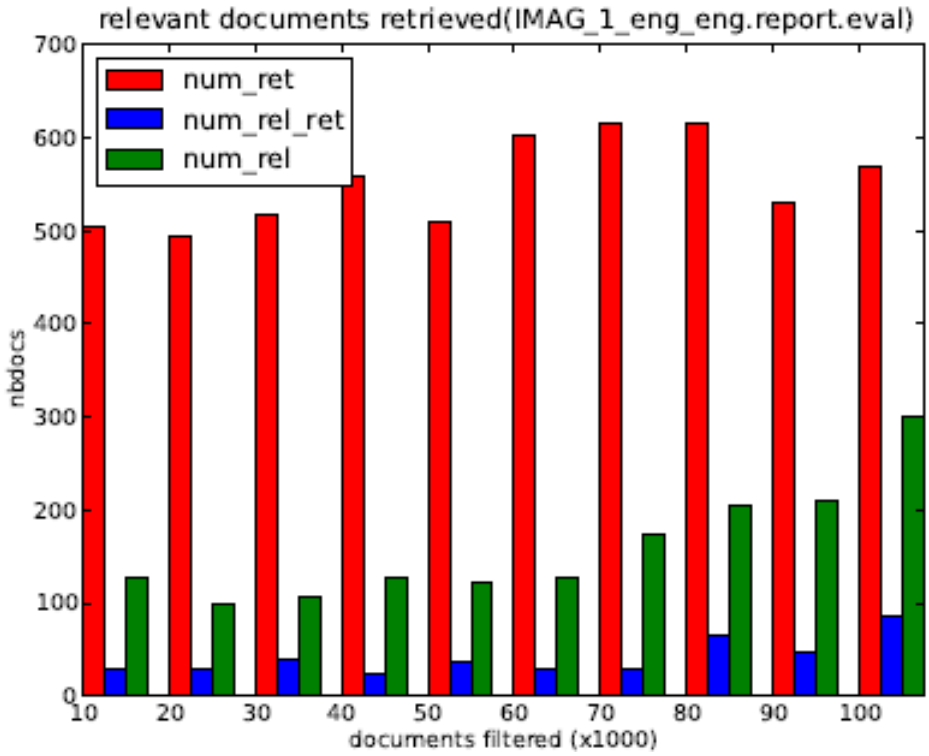

Figure 3.7: Number of documents retrieved for Run 4

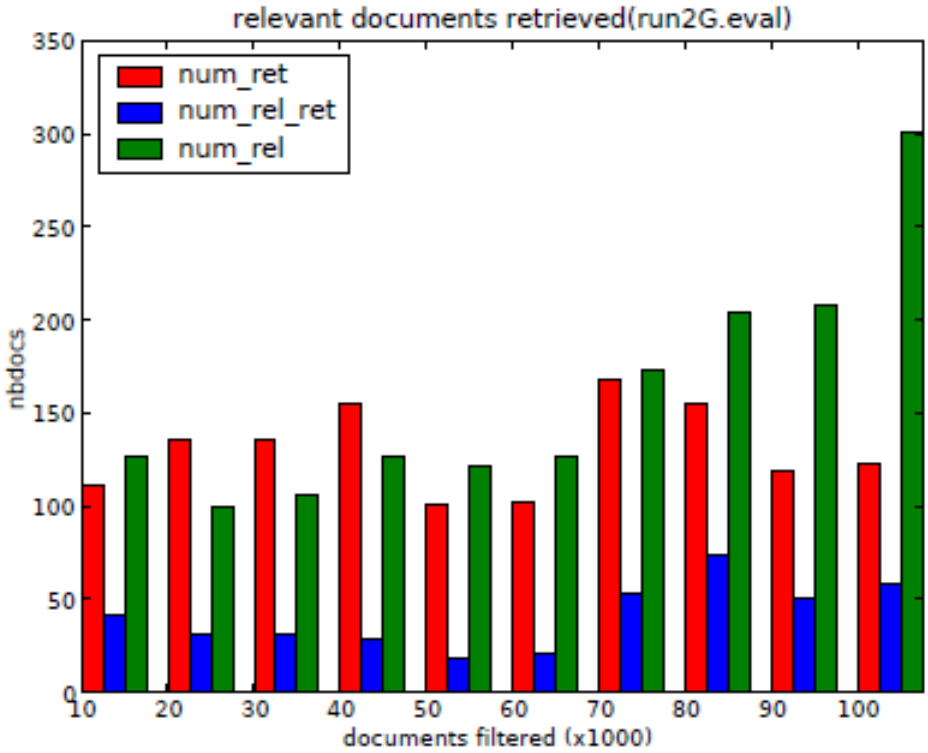

Figure 3.8: Number of documents retrieved for Run 2

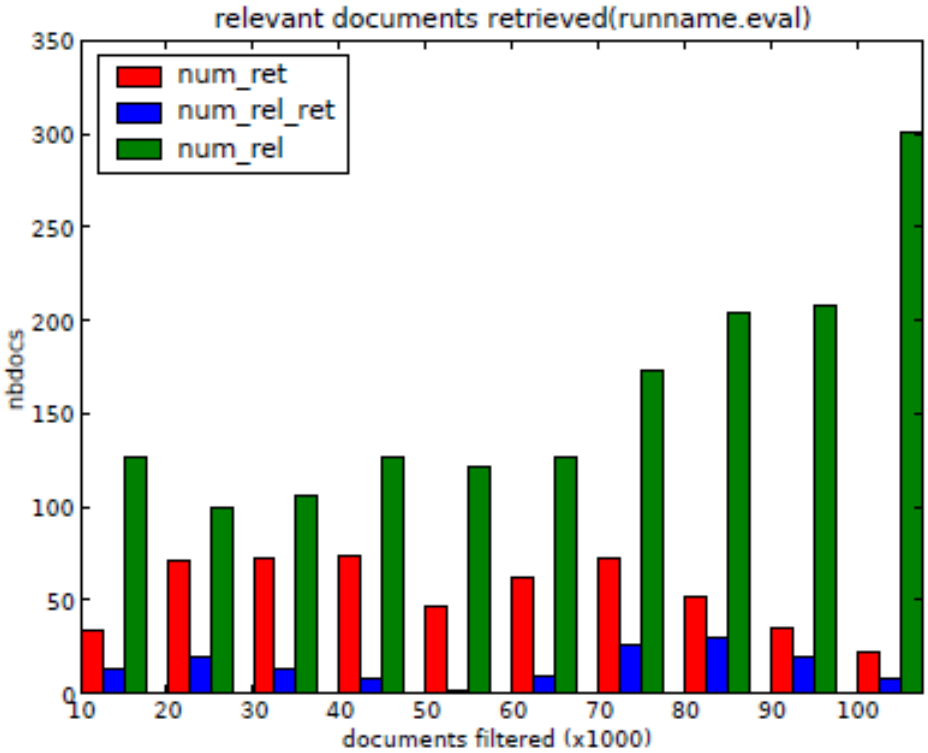

Figure 3.9: Number of documents retrieved for Run 3

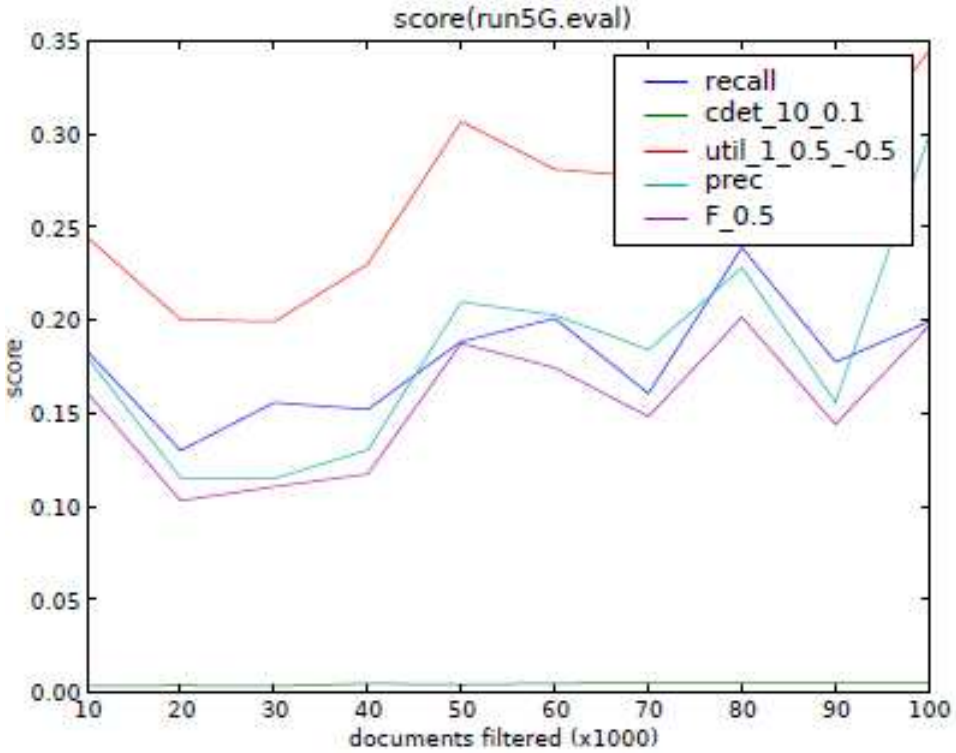

Figure 3.10: Score Evolution for Run 1

The evolution of different measures are computed at different times in the process, each time 10,000 documents have been processed.

For Run 1 (Figure 3.10), all of the measures, except utility and precision, randomly vary but remain approximately the same at the end. The curve at the bottom represents the detection cost for all of the runs. The evolution for different measures for Run1 is as follows: Precision changes from 0.18 in the beginning to 0.29 at the end, 0.18 vs 0.20 for Recall, 0.24 vs 0.34 for Utility, and 0.16 vs 0.20 for F-measure.

For Run 4 (Figure 3.11), the curve just above the one meant for detection cost, describes anticipation. For Run 4, all of the measures randomly vary but increase significantly as compared to the initial values (0.17 vs 0.30 for Precision, 0.15 vs 0.20 for Recall, 0.15 vs 0.25 for Utility, 0.12 to 0.19 for the F-measure, and 0.04 in the beginning vs 0.125 at the end for anticipation) during the course of the filtering process.

For Run2 (Figure 3.12), Precision decreases from 0.25 to 0.23 during the filtering of 100,000 documents, Recall's initial and final values are the same (0.14), Utility increases from the start value of 0.21 to 0.31 while F-measure increases from 0.15 to 0.19.

The different measures change in the following manner for Run3 (Figure 3.13): Precision decreases 0.18 to 0.08, Recall decreases from an already low value of 0.07 to 0.05, Utility increases a little bit from the initial value of 0.21 to the final value of 0.25, and F-measure reduces from 0.09 to 0.04.

Table 3.3 describes the macro values for the different runs. These values represent the average score over the complete set of 50 profiles. P represents precision, R represents recall, F represents

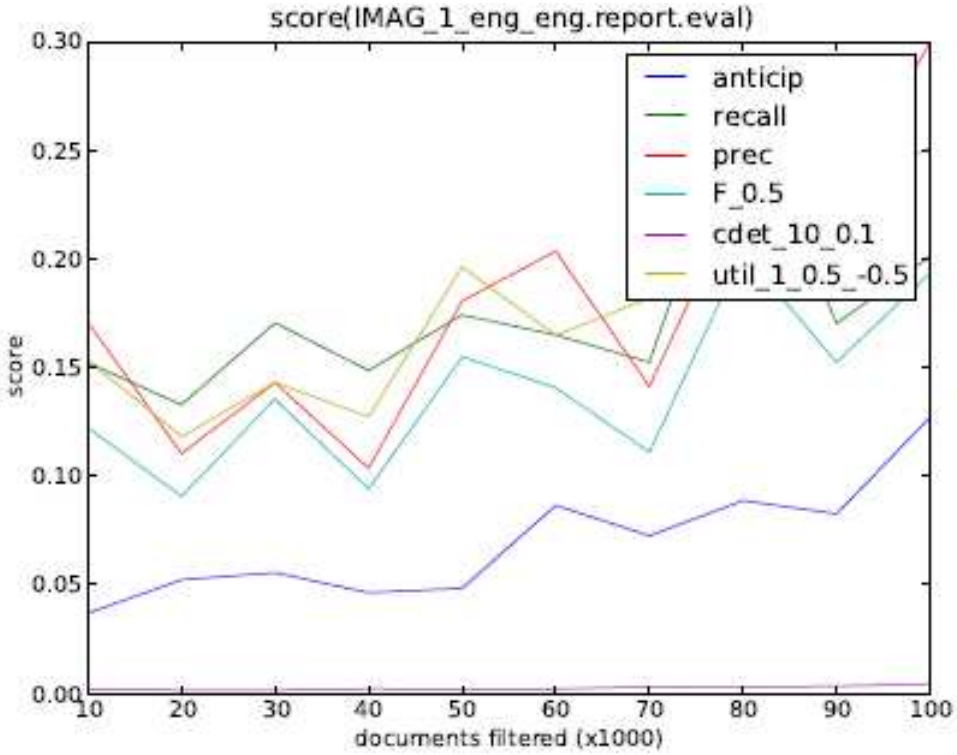

Figure 3.11: Score Evolution for Run 4

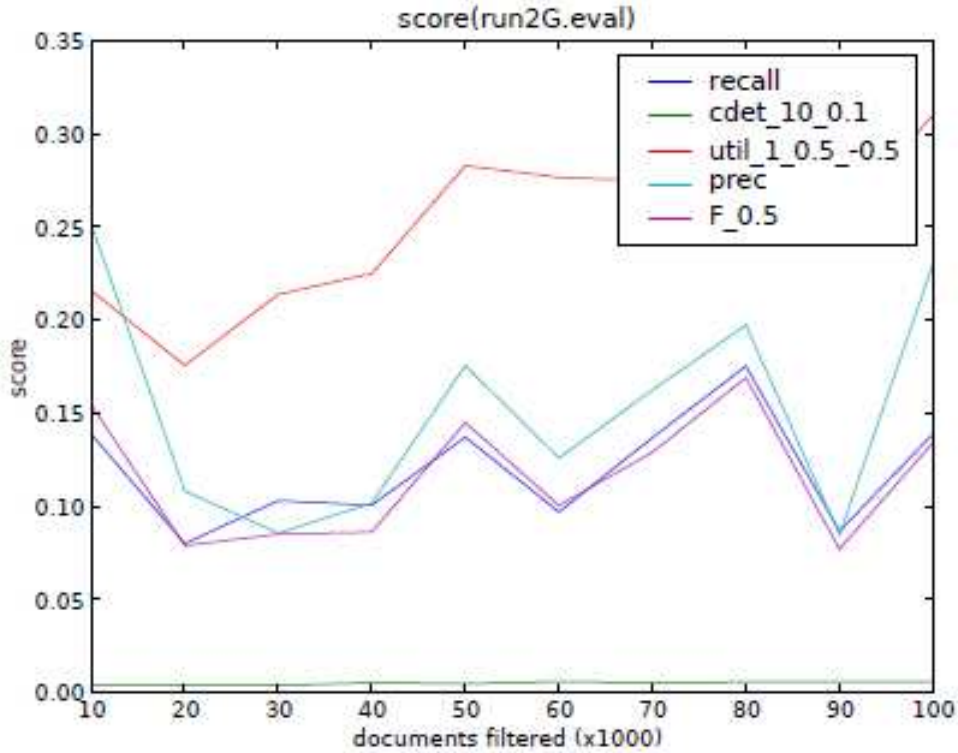

Figure 3.12: Score Evolution for Run 2

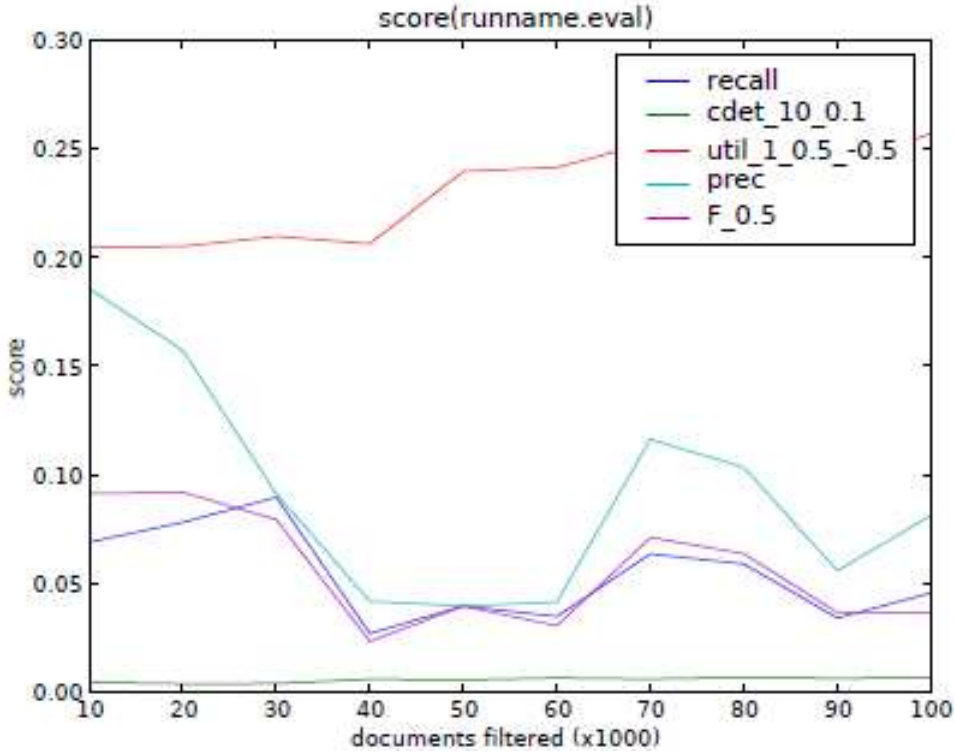

Figure 3.13: Score Evolution for Run 3

	Macro_P	Macro_R	Macro_F	Macro_LU	Macro_DC	Anticipation
Run 1	0.306	0.260	**0.209**	**0.351**	0.007	0.307
Run 2	0.357	0.165	0.165	0.335	0.008	0.317
Run 3	**0.366**	0.068	0.086	0.311	0.009	0.207
Run 4	0.256	**0.295**	0.206	0.205	**0.002**	**0.430**

Table 3.3: Run Scores

F-measure, *LU* represents linear utility while *DC* represents detection cost. The best results are given in bold. Run 4 has the best macro recall (0.295) as compared to all of the runs. It can be noted that Run 1, 2 and 3 are all precision-oriented since the precision values are clearly much better than the recall values. On the other hand, Run 4 is recall-oriented since it has got a better recall as compared to the precision value. The macro F-measure for the Run 1 and Run 4 are significantly greater than that of Run 2 and 3. However, Run 1 surpasses Run 4 in terms of macro precision. The overall macro detection cost is very low in all of these runs (less than 0.01), with Run 4, being the most economical. This is a strong point for these algorithms. The macro linear utility of Run 1 is greater than that of Run 4. On contrary, anticipation for Run 4 is significantly better than that for the other runs.

3.5.1 An Insight into the Micro scores

As far as the micro values for different topics are concerned (Reference Appendix), they differ a lot from topic to topic. For example, Run 1 has got a recall of 0.857 for topic no. 107, 0.962 for topic 118, 0.845 for topic 125 and a micro recall of 0.917 with topic 146. Among these, topic 120 and 146 have got very less number of relevant documents. However, among the topics considered above, only topic 125 and 146 have got a Micro F-score and Micro linear utility greater than 0.63.

Similarly for Run 2, topic 107, 118, 120, 125, 146 and 148 have got a Micro recall greater than 0.70. However only topic 107, 125 and 148 have got a Micro F-score and Micro linear utility greater than 0.635.

For Run 3, only topic no. 146 has got a Micro recall, F-score and linear utility greater than 0.63.

In the case of Run 4, only topic 107, 119, 120, 123, 132, 140 and 146 have got a Micro recall greater than 0.63. All of these topics except topic no. 123 contain fairly small number of relevant documents. As for majority of these topics, the Micro precision is quite low, the Micro F-score remains low as well (except topic no. 107 and 132) The micro linear utility for Run 4 is greater than 0.63 for topic no. 107 and 132. These figures indicate that a high Micro F-score indicates a high Micro linear utility. Similarly, in order to have a good F-score, both precision as well as recall must be good enough.

It can be easily concluded from these results, that the use of limited number of feedbacks (only 50 i.e. one per topic) did not help to get very good results, although it helped to increased the micro recall.

3.5.2 Comparison with other approaches

Table 3.4 shows the comparison between the two online algorithms employed at INFILE [8]. The other participant was from University of Wollongong, Dubai (UOWD). It can be observed that the best performance was from IMAG team while using the run *run5G*. It retrieved the highest number (601) of relevant documents out of a total of 1597 relevant documents. Consequently, it got the highest recall as well as the highest F-score among all of the different runs. The *run5G* was the most useful of all of the runs. *runname* got the best precision score whereas the highest anticipation was for *run2G*. The F-measure, precision and utility for *run5G* is the highest among

team	run	year	num_rel_ret	Pr	Re	F	LU	A
IMAG	run5G	2008	**601**	0.31	**0.26**	**0.21**	**0.35**	0.31
IMAG	run2G	2008	411	0.36	0.17	0.17	0.34	**0.32**
IMAG	runname	2008	152	**0.37**	0.07	0.09	0.31	0.21
UOWD	base	2009	20	0.00	0.01	0.01	0.03	0.05

Table 3.4: Comparison between different approaches for Online Filtering

team	run	num_rel_ret	Pr	Re	F	LU	A
IMAG	IMAG_1	413	**0.26**	0.30	**0.21**	**0.21**	0.43
UAIC	uaic_4	**1267**	0.09	**0.66**	0.13	0.054	**0.73**
SINAI	topics_1	940	0.02	0.50	0.04	0.00	0.57

Table 3.5: Comparison between different approaches for Batch Filtering

all of the different campaigns: monolingual french and cross-language french -> english.

Different batch algorithms are compared in the Table 3.5. Among the other participants were Universitatea Alexandru Ioan Cuza of IASI (UAIC), Romania and University of Jean (SINAI), Spain. Only the best runs for each of the three teams is provided. The best run in terms of precision and F-measure is IMAG_1. It has also got the highest utility among the 3 runs considered. Although the recall for uaic_4 is 0.66, yet the precision is only 0.09 which explains the reason for overall low F-score. However, the best anticipation (0.73) is for the run uaic_4. It is evident that both the runs uaic_4 and topics_1 are recall oriented since the recall values are much greater than the ones for precision.

3.6 Conclusion

A simple extension of the *kNN* algorithm using thresholds has been presented to define online and batch filtering algorithms. The results obtained can be deemed encouraging as the macro F-measure in the case of online algorithm as well as the batch one equals approximately 20%, for a collection of 100,000 documents and 50 topics, out of which only 1597 documents are relevant. While comparing the online results of 2008 with those for the batch campaign of 2009, it can be seen that the batch algorithm has a much better macro recall (almost 30% against 26% in 2008) along with a lower macro detection cost (0.002 vs 0.007) and a much better anticipation (0.430 vs 0.307). Considering the evolution of different measures, it can be observed that the values for all of the measures increase, with the increase in the number of documents filtered. The main difference between the batch and online algorithms lies in the way the initial set of documents is constructed. In batch algorithm, the initial set is built from finding the 10 nearest neighbors for each of the profile, whereas feedbacks are used in the online algorithm to construct the initial set of documents. It can be concluded from the results that the use of a limited number of feedbacks, in general, does not help to get very good results.

Furthermore, comparing the online results submitted by different participants, it can be seen

that IMAG team got the best results for all of the metrics. Moreover, the run *run5G* had got the highest recall and F-score and was the most useful of all of the runs. For Batch filtering, IMAG team got the highest precision, F-score and Utility among all of the submitted runs.

Chapter 4

Similarity Metric Learning in Nearest Neighbor Classification

Contents

4.1 Introduction

In Chapter 3, thresholds based on cosine similarity were learned. However, the approach followed is only interesting provided only a slight supervision is available. In case, complete supervision is available, it is better to learn the complete metric. An example is the case of classification problems where people prefer to learn the complete metric ([28, 99, 112]) which has proved to be a better choice as compared to only learning the thresholds.

Most works on metric learning for *kNN* classification have focused on distance metric learning (see for example [32, 99, 112]). However, in many practical situations, similarities may be preferred over distances. This is typically the case when one is working on texts, for which the cosine measure has been deemed more appropriate than the standard distance metrics like the Euclidean or the Mahalanobis ones. Furthermore, several experiments show that the use of the cosine similarity should be preferred over the Euclidean distance on several, non textual collections as well (see e.g. [18, 72, 84, 87]). Being able to efficiently learn appropriate similarity measures, as opposed to distances, for *kNN* classification is thus of high importance for various collections. If several works have partially addressed this problem (as for example [1, 46, 52]) for different applications, no previous work is known which has fully addressed it in the context of learning similarity metrics for *kNN* classification.

There is a wide range of options for selecting a similarity metric. However, the interest here lies in the scalar product of the form $x^t x'$ where x and x' are two examples and t represents the transpose.

A similarity metric between two examples x and x' can be defined in the following manner:

$$s_A(x, x') = \frac{x^t A x'}{N(x, x')} \tag{4.1}$$

where A is a $(p \times p)$ similarity matrix (diagonal or not) and $\mathrm{N}(x, x')$ is a normalization which depends on x and x' (this normalization is typically used to map the similarity function to a particular interval, as $[0, 1]$). Equation 4.1 represents an unconstrained similarity metric learning problem since the normalization is completely independent of the similarity matrix.

A generalized cosine similarity can also be defined from the equation 4.1 in which case the normalization is dependent on the similarity matrix and the similarity matrix is positive, semi-definite as described in the following equation:

$$s_A(x, x') = \frac{x^t A x'}{\sqrt{x^t A x}\sqrt{x'^t A x'}} \tag{4.2}$$

Here the normalization is dependent on the similarity matrix A and A is a PSD matrix.

As opposed to Passive Aggressive algorithms [23] which use diagonal approximations for a full covariance matrix, we are interested in learning complete similarity matrices.

The next section describes the unconstrained similarity metric learning followed by its extension based on PSD matrices in Section 4.3. The unconstrained similarity metric learning is compared with the *RELIEF* algorithm in Section 4.4. Section 4.4 also contains the description of a *RELIEF* based similarity learning algorithm (*RBS*) along with a stricter version of *RBS*, called *sRBS*. Generalized cosine similarity learning as well as its comparison with the unconstrained similarity learning is provided in Section 4.5.

4.2 Unconstrained Similarity Metric Learning

In this section, unconstrained similarity metric learning problem based on equation 4.1 is presented. Equation 4.1 generalizes several, standard similarity functions. For example:

1. Standard cosine measure, widely used in text retrieval, is obtained by setting A to the identity matrix I, and $\mathrm{N}(x, x')$ to the product of the L_2 norms of x and x'.
2. *Dice* coefficient is obtained, from presence/absence vectors (i.e. all coordinates of x and x' are either 0 or 1), by setting A to $2I$, and $\mathrm{N}(x, x')$ to the sum of the L_1 norms of x and x'.
3. Similarly, the *Jaccard* coefficient, again computed between presence/absence vectors, corresponds to $A = I$ and $\mathrm{N}(x, x') = |x| + |x'| - x^t x'$ (where $|x|$ denotes the L_1 norm).

Furthermore, the fact that no condition is imposed on A (apart from being square) allows to consider both symmetric as well as asymmetric similarity functions, depending on the targeted task. For example, Bao et al. [1], make use of two asymmetric similarity functions: the *Relative Frequency Model*, which is an asymmetric version of the cosine, and the *Inclusion Proportion Model*, which is an asymmetric version of the *Dice* coefficient, and show that these asymmetric measures are better than their symmetric counterparts in order to retrieve partial copies of text documents.

4.2.1 Problem Formulation

The problem addressed here is to learn a similarity function of the general form given in equation 4.1 from the training data, to be used in *kNN* classification. Let $(x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)})$ be a training set of n labeled examples with inputs $x^{(i)} \in \mathbb{R}^p$ and discrete (but not necessarily binary) class labels $c^{(i)}$ (where $c^{(i)}$ represents the class of the ith example). The aim is to learn a $(p \times p)$ similarity matrix A that aims at optimizing the *kNN* classification where the neighborhood function is given in equation 4.1. To do so, for each $x^{(i)}$, its k *target* neighbors are introduced as in Weinberger et al. [112], which are the k elements in $c^{(i)}$ closest to $x^{(i)}$, according to a base similarity measure. For example, one may be interested in learning a matrix A which generalizes the cosine similarity. In this case, the k target neighbors will be defined according to the standard cosine similarity, and will not change during the process of learning the similarity matrix A. The target neighbors of $x^{(i)}$ are denoted by: $y_l^{(i)}$, $1 \leq l \leq k$. Furthermore, for each $x^{(i)}$, its k nearest neighbors in $\bar{c}^{(i)}$ are found, also known as the *impostors* and represented as: $z_l^{(i)}$, $1 \leq l \leq k$.

A notion of separability can now be formalized, capturing the fact that any example should be closer to its k target neighbors than to any other set of k examples.

Definition 1 *Let $\mathcal{S} = (x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)})$ be a training sequence of n vectors in $\mathbb{R}^d$ and let k be an integer. Let $(y_1^{(i)}, \cdots y_k^{(i)})$ be the k target neighbors of $x^{(i)}$ in $c^{(i)}$. Lastly, let $\bar{c}^{(i)}$ denote the complement of $c^{(i)}$ in the category set. It can be said that $\mathcal{S}$ is separable with some margin $\gamma > 0$ iff there exists a $(p \times p)$ matrix A, with $\|A\| = 1$, such that:*

$$\forall i,\ \forall (z_1, \cdots, z_k) \in \bar{c}^{(i)},\ \sum_{l=1}^{k} \Big(s_A(x^{(i)}, y_l^{(i)}) - s_A(x^{(i)}, z_l) \Big) \geq \gamma$$

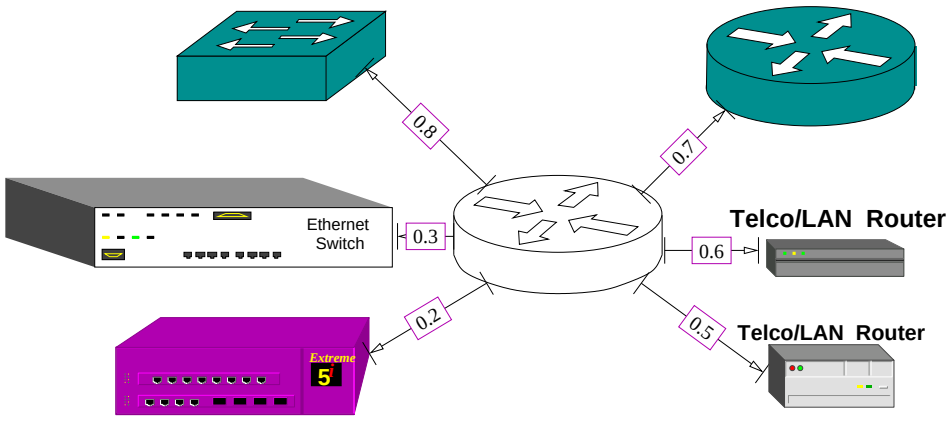

Figure 4.1: A classification scenario along with similarity values

where $\|A\|$ represents the Frobenius norm of the matrix A. Figure 4.1 depicts a scenario where a new object (in the center) has to be classified as a *router* or as a *switch* based on its similarity with the examples of these two classes. Here the examples belonging to the *router* class, also known as the *target* examples, can be represented as: y_1, y_2 and y_3, whereas the examples from the *switch* category, also known as the *impostors*, can be written as: z_1, z_2 and z_3. Furthermore, an assumption is made that the value of the threshold γ is 0.3. This sequence is separable since the difference of the sum of similarities between the new example and the examples belonging to the same class i.e. *router* and the sum of similarities between the new example and the examples from the *switch class* is greater than the threshold value i.e. $1.8 - 1.3 = 0.5$

Of course, in practice, the data is not likely to be separable in the above sense e.g. when the difference between the sum of similarities with the same class examples y_l and the examples from different classes z_l is less than the threshold γ. Nevertheless, a measure describing how close a matrix A is to separate the data with margin γ can be defined as follows:

Definition 2 *Let $\mathcal{S} = (x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)})$ be a training sequence of n vectors in $\mathbb{R}^p$, let A be a $(p \times p)$ matrix such that $\|A\| = 1$, and let $\gamma > 0$. The γ-related measure of example i is defined as:*

$$\epsilon_i = max(0, \gamma - m_i)$$

with

$$m_i = \sum_{l=1}^{k} s_A(x^{(i)}, y_l^{(i)}) - max_{(z_1,\cdots,z_k)\in\bar{c}^{(i)}} \sum_{l=1}^{k} s_A(x^{(i)}, z_l)$$

The overall separation measure $D_{A,\gamma}$ of $\mathcal{S}$ with respect to A and γ is defined as:

$$D_{A,\gamma} = \sqrt{\sum_{i=1}^{n} \epsilon_i^2}$$

If the data is separable with margin γ according to definition 1, then there exists A such that: $D_{A,\gamma} = 0$. Looking at the example discussed earlier, one can note that the value of m_i is

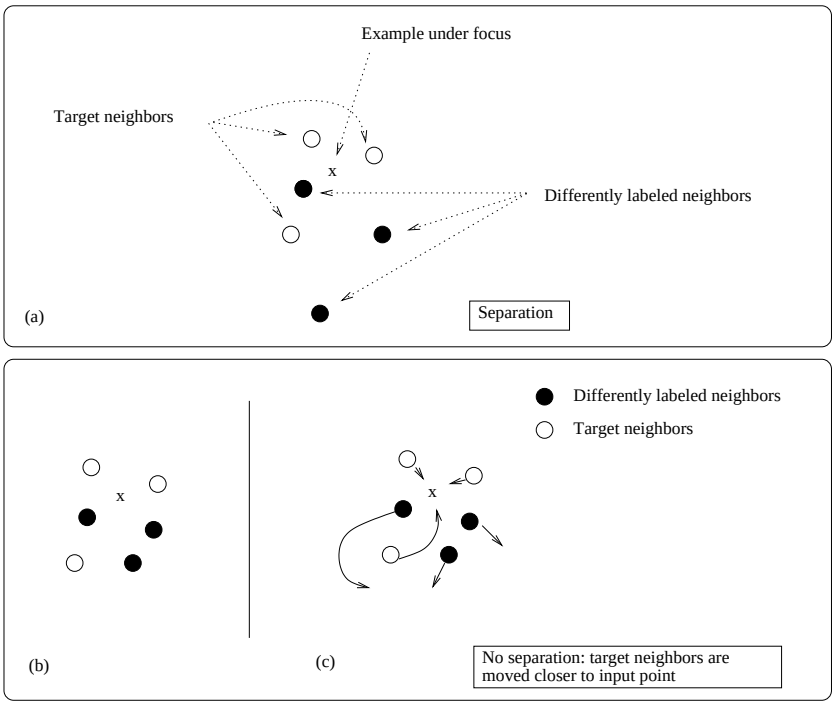

Figure 4.2: In (a) the input point is separated with $k = 3$, whereas it is not in (b). (c) illustrates the process being aimed at: moving target points closer to the input point, while pushing away differently labeled examples.

$1.8 - 0.8 = 1.0$ where 1.8 is the sum of similarities between the new example and the *target* neighbors whereas 0.8 is the maximum similarity value between the example to be classified and the *impostors*. As $\gamma - m_i = 0.3 - 1.0 = -0.7$ is less than zero, hence the γ-related measure ϵ_i and the overall separation measure $D_{A,\gamma}$ become zero.

If no example can be separated by A with margin γ, then $D_{A,\gamma} > 0$, with the property that the lower the $D_{A,\gamma}$, the higher the capacity of A to separate $\mathcal{S}$ with margin γ.

The notion of separation being considered here is relatively loose as there is no strict requirement that all target neighbors must be in the k nearest neighbors of an example. Rather, the aim is that any point be, globally, closer to k points from the same class than to k points from any other class. This simplification, also used in Weinberger et al. [112], allows one to avoid setting complex constraints on each target neighbor, while still retaining the idea behind *kNN* classification.

Figure 4.2 illustrates the notion of separability being considered here. In figure 4.2(a), the input point is separated, with $k = 3$ assuming that the difference between the sum of similarities between the example under focus and its *target* neighbors and the sum of similarities between the example under focus and the *impostors* is greater than the margin γ, whereas this is not the case in figure 4.2(b). The separation does not need to take place in the original input space, but rather on the space induced by the metric defined by A. Figure 4.2(c) illustrates what is being aimed at: moving the target points closer to the input point, while pushing away differently labeled examples (impostors). With an appropriate matrix A (which plays the role of a similarity metric), the target and negative neighbors of a given input point are separated, the former ones

being closer to the input point than the latter ones (note however that the separation is not necessarily linear when the number of neighbors, k, considered is greater than 1 - in this latter case, the linear separation is not obtained in the original input space when $A \neq I$). However, strictly speaking, the classification rule sustaining the above definitions of separation is: *for any example $x^{(i)}$, compute its k nearest neighbors in each class $c^{(i)}$ ($x_1^{(i)}, \cdots, x_k^{(i)}$); assign $x^{(i)}$ to the class $c^{(i)}$ for which $\sum_{l=1}^{k} s_A(x^{(i)}, x_l^{(i)})$ is maximum.* The goal here is to learn the similarity matrix A of equation 4.1 with guaranteed performance bounds with respect to the above classification rule and definitions of separation. As described in Chapter 6, by doing so, the standard kNN rule can be improved.

The matrix A in equation 4.1 can have many different variants: it can be symmetric or asymmetric or it can be chosen to be positive semi-definite as well.

4.2.2 An unconstrained Similarity metric Learning Algorithm - *SiLA*

An algorithm to learn unconstrained similarity metrics of the form given by equation 4.1 is presented here. This algorithm, called as *SiLA*, is based on the voted perceptron algorithm proposed in Freund and Schapire [37], and used in Collins [20]. It allows learning diagonal, symmetric or even asymmetric matrices, depending on the final form of the similarity function one is interested in.

The core of *SiLA* is an on-line update rule which iteratively improves the current estimate of the similarity matrix A. The overall goal is to move target examples closer to their input point whenever the input point is closer to a set of differently labeled examples. A theoretical motivation for *SiLA* is provided at the end of this section.

In the remainder of this section, $\text{kNN}(A, x, s)$ is used in order to denote the k nearest neighbors of example x in class s with the similarity function given by equation 4.1. For each example i, $T(i)$ will denote the set of target neighbors of $x^{(i)}$. The training algorithm is given below:

SiLA - Training

Input: training set $((x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)}))$ of n vectors in $\mathbb{R}^p$, number of epochs M; A^1_{ml} denotes the element of A^1 at row m and column l

Output: list of weighted $(p \times p)$ matrices $((A^1, w_1), \cdots, (A^q, w_q))$

Initialization $\tau = 1, A^1 = 0$ (null matrix), $w_1 = 0$

Repeat M times (epochs)

1. for $i = 1, \cdots, n$
   2. $B(i) = \text{kNN}(A^\tau, x^{(i)}, \bar{c}^{(i)})$
   3. if $\sum_{y \in T(i)} s_A(x^{(i)}, y) - \sum_{z \in B(i)} s_A(x^{(i)}, z) \leq 0$
      4. $\forall (m, l), 1 \leq m, l \leq p,$
         $A^{\tau+1}_{ml} = A^\tau_{ml} + \sum_{y \in T(i)} f_{ml}(x^{(i)}, y) - \sum_{z \in B(i)} f_{ml}(x^{(i)}, z)$
      5. $w_{\tau+1} = 1$
      6. $\tau = \tau + 1$
   7. else
      8. $w_\tau = w_\tau + 1$

When an input example $x^{(i)}$ is not separated from differently labeled examples, the current A matrix is updated by the difference between the coordinates of the target neighbors and the closest differently labeled examples also known as the *impostors* represented by the set $B(i)$ (line 4 of the algorithm), which corresponds to a standard perceptron update. When the current estimate of A correctly classifies the input example under focus, then A is left unchanged while its corresponding weight is increased by 1, so that the weights finally correspond to the number of examples correctly classified by A over the different epochs.

The functions f_{ml} allows to learn different types of matrices and hence different types of similarities:

1. For a diagonal matrix, $f_{ml}(x, y) = \frac{\delta(m,l)x_m^t y_l}{\mathrm{N}(x,y)}$ (with δ the Kronecker symbol),

2. For a symmetric matrix, $f_{ml}(x, y) = \frac{x_m^t y_l + x_l^t y_m}{\mathrm{N}(x,y)}$,

3. For a square matrix (and hence, potentially, an asymmetric similarity), $f_{ml}(x, y) = \frac{x_m^t y_l}{\mathrm{N}(x,y)}$.

It can be seen that the function f_{ml} is independent of the similarity matrix A. The weighted matrices provided by *SiLA* can be used to predict the class(es) to which a new example should be assigned. Two basic rules for prediction are considered: the first one corresponds to the standard *kNN* rule, whereas the second one directly corresponds to the notion of separation introduced earlier, and is based on the consideration of the same number of examples in the different classes. The new example is simply assigned to the closest class, the similarity with a class being defined as the sum of the similarities between the new example and its k nearest neighbors in that particular class. The second rule is called symmetric *kNN* rule and is denoted by *SkNN*.

In order to speed up the learning process, all of the training as well as the test examples are normalized before launching the algorithm. Furthermore, the sets $T(i)$ and $B(i)$ are also computed beforehand. Since the set $B(i)$ changes over the passage of time, a certain number of impostors (e.g. 100) could be found for each of the example before the algorithm has been launched.

The worst-time complexity of *SiLA* is $O(Mnp^2)$ where M represent the number of iterations, n is the number of train examples while p stands for the number of dimensions or attributes. The most costly steps consist of calculating the similarity s_A and f_{ml}.

4.2.3 Online to Batch Conversion

The core of *SiLA* is an update rule that is used incrementally, for each example. It is thus easy to extract from the description of *SiLA* a batch version of the algorithm. The way the matrices learned are used for prediction, corresponds to a transformation of an on-line algorithm to a batch one, following a methodology described in Helmbold and Warmuth [51].

SiLA - Prediction
Input: new example x in $\mathbb{R}^p$, list of weighted $(p \times p)$ matrices $((A^1, w_1), \cdots, (A^q, w_q))$; A is defined as: $A = \sum_{l=1}^{q} w_l A^l$
Output: list of classes

1. *Standard kNN rule*
   Compute the k nearest neighbors based on s_A; select the class with the highest weight (or the class the more represented in the nearest neighbor set)

2. *Symmetric classification rule - SkNN*
   Let $T(x, s) = kNN(A, x, s)$; assign x to the class for which $\sum_{z \in T(x,s)} s_A(x, z)$ is maximal [14].

The deterministic leave-one-out conversion of the training version of *SiLA* corresponds to the weighted sum ($A = \sum_{l=1}^{q} w_l A^l$) used in the prediction rules given above. One can find in Dekel et al. [30] a study of similar on-line to batch conversions, showing that it may be beneficial to weigh down (or even forget) the matrices (or vectors) learned in the first few iterations of the on-line algorithm. That is, instead of basing the prediction on the complete sequence $((A^1, w_1), \cdots, (A^q, w_q))$, base it instead on, say, the last r elements. This strategy is used in the experiments conducted.

SiLA could be used in either a binary or multi-class mode:

1. In the binary setting, the algorithm is run separately for each class, where the class under consideration is made as 1 while the rest of the classes are made 0.

2. However, in the multi-class mode, *SiLA* is run only once along with the original class labels. In this way, multi-class mode is much faster than the binary mode.

There is yet another method of converting the binary mode into a multi-class one. The similarity value for each of the test example which predicts a class label of 1 is stored. All of the examples for which a class label of 0 is predicted, are discarded since the exact class label cannot be determined. The similarity values are stored for each of the different classes. In order to determine the final classification, the class having the greatest similarity is chosen.

There are a certain number of advantages in the binary version. First, it allows using the two prediction rules given above. It also allows learning *local* matrices, which are more likely to capture the variety of the data. Finally, its application in prediction results in a multi-label decision.

4.2.4 Analysis of SiLA

Performance bounds for *SiLA* algorithm are provided in this subsection. These bounds, and the theorems they rely on, directly parallel the ones provided by Freund and Schapire [37], and used

[14] Nock et al. [76] have discussed another type of symmetric nearest neighbor rule in which a vote is made for some example x using the points which could belong to the k nearest neighbors of x, and the points for which x could be one of the k nearest neighbors.

in Collins [20]. To see the parallel between this work and the above-mentioned ones, first note that $\frac{x^t A x'}{\mathrm{N}(x,x')}$ can be rewritten as:

$$\frac{x^t A x'}{\mathrm{N}(x,x')} = \alpha \cdot \phi(x,x')$$

with:

$$\begin{cases} (\alpha, \phi(x,x')) \in \mathbb{R}^p \times \mathbb{R}^p & \text{when } A \text{ is diagonal,} \\ (\alpha, \phi(x,x')) \in \mathbb{R}^{p^2} \times \mathbb{R}^{p^2} & \text{otherwise.} \end{cases}$$

where α can be seen as the vector equivalent to matrix A. Different representations are possible with this transformation:

1. The cosine similarity is obtained, with this representation, by setting α to the unit vector $(\alpha_m = 1,\ 1 \leq m \leq p)$ and $\phi_m(x,x') = \frac{x_m^t x'_m}{\|x\|\|x'\|}$.

2. By setting ϕ to the *tensor product* between vectors x and x', one obtains a representation equivalent to the one with an unconstrained, square matrix A.

3. By setting ϕ to the *symmetric product*, i.e. $\phi_{ml}(x,x') = \frac{x_m^t x'_l + x_l^t x'_m}{\mathrm{N}(x,x')}$, one obtains a representation equivalent to the one with a symmetric matrix A.

The theorems justifying the use of the voted perceptron algorithm can be extended to *SiLA* as well, and are next presented. The justification of *SiLA* proceeds in three steps:

1. Theorem 1 justifies the core on-line update of *SiLA* in the separable case,

2. Theorem 2 provides a similar justification for the non-separable case, and

3. Theorem 3 provides the justification for the batch version used for prediction.

The proofs for Theorem 1 and 2 are given in the Appendix A.

Theorem 1 (separable case)**.** *For any training sequence* $\mathcal{S} = ((x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)}))$ *separable with margin* γ*, for one iteration (epoch) of the (on-line) update rule of SiLA*

$$\textit{Number of mistakes} \leq R^2/\gamma^2$$

where R *is a constant such that:*

$$\forall i, \forall (z_1, \cdots, z_k) \in \bar{c}^i, \left\| \sum_{y \in T(i)} \phi(x^{(i)}, y) - \sum_{n=1}^{k} \phi(x^{(i)}, z_n) \right\| \leq R$$

Theorem 1 implies that, if the data is separable, then the update rule of *SiLA* makes a number of mistakes bounded above by a quantity which depends on the margin (γ) of the data (the larger the margin, the lesser the number of mistakes made). The more general case where the data is not separable is covered by theorem 2, which makes use of the measure $D_{A,\gamma}$ (or equivalently $D_{\alpha,\gamma}$ with the new representation) introduced in definition 2.

Theorem 2 (non separable case). *For any training sequence* $\mathcal{S} = ((x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)}))$*, for one iteration (epoch) of the (on-line) update rule of SiLA*

$$Number\ of\ mistakes \leq min_{\alpha,\gamma} \frac{(R + D_{\alpha,\gamma})^2}{\gamma^2}$$

where R *is a constant such that*

$$\forall i, \forall (z_1, \cdots, z_k) \in \bar{c}^i, \left\| \sum_{y \in T(i)} \phi(x^{(i)}, y) - \sum_{n=1}^{k} \phi(x^{(i)}, z_n) \right\| \leq R,$$

and the min is taken over α *and* γ *such that* $\|\alpha\| = 1, \gamma > 0$.

This theorem implies that, provided the data is close to being separable, the update rule of *SiLA* converges in a finite number of steps, and has a number of mistakes bounded by a quantity which is smaller when the separation of the data is better (as measured by D). However, the interest is not only in the convergence of the update rule (which corresponds to an on-line version of the algorithm), but also on the convergence of the batch version used for prediction. The following theorem provides both a proof of this convergence and shows that the batch version is able to generalize well, i.e. behaves adequately on test (unseen) data. This theorem is based on the on-line to batch conversion studied in Helmbold et al. [51].

Theorem 3 (generalization). *Assume all examples are generated i.i.d. at random. Let* E *be the expected number of mistakes that the update rule of SiLA makes on a randomly generated sequence of* $m + 1$ *examples. Then given* m *random training examples, the expected probability that the deterministic leave-one-out conversion of this algorithm makes a mistake on a randomly generated test instance is at most:* $\frac{2E}{m+1}$.

4.3 *eSiLA* - An extension of *SiLA*

The similarity given in equation 4.1 does not guarantees that the form $x^t A x'$ corresponds to a symmetric bi-linear form, and hence a scalar product. In order to incorporate this guarantee, the similarity matrix A must be made a positive, semi-definite (PSD) one, which can be achieved by projecting A onto the set of positive, semi-definite matrices. The resulting algorithm is an extension of *SiLA* and is called *eSiLA* [85].

The projection onto the set of PSD matrices can be accomplished based on the fact that any matrix A can be represented in terms of its eigenvalues and its eigenvectors. In order to convert the matrix A^{t+1} into a PSD one, only its positive eigenvalues are selected whereas the non-negative eigenvalues are discarded. The projection can be written as:

$$\hat{A}^{t+1} = \sum_{j, \lambda_j > 0} \lambda_j u_j u_j^t$$

where λ and u represent the eigenvalues and eigenvectors of the matrix A^{t+1}. $\hat{A}^{t+1}$ stands for the matrix obtained after performing the projection and is a PSD (and symmetric) matrix .

This extension did not improve the performance of *SiLA* algorithm. Nevertheless, the technique used for projection was later used for *RELIEF* based algorithms (Section 4.4) as well as the generalized cosine similarity learning (Section 4.5).

4.4 Unconstrained Similarity Metric Learning and *RELIEF* Algorithm

As the reader may have noticed that learning the similarity matrix in *SiLA* bears resemblance with the feature reweighting procedures. Among such techniques, the *RELIEF* family of algorithms has received a lot of attention from many different communities in the recent years. In this section, unconstrained similarity metric learning is positioned with the *RELIEF* algorithm. It is important to mention that Sun and Wu [102] have shown that *RELIEF* is basically a distance metric learning algorithm which aims to optimize a linear utility function while maximizing the margin. After comparing *SiLA* with the *RELIEF* algorithm, a *RELIEF*-Based Similarity learning algorithm (*RBS*) is described together with its stricter version known as *sRBS*. Furthermore, the effect of positive, semi-definitiveness on the *RELIEF* based algorithms is also discussed.

4.4.1 *SiLA* and *RELIEF*

It has been shown that the *RELIEF* algorithm solves convex optimization problem while maximizing a margin-based objective function using *kNN* algorithm. The weights are updated based on the nearest *hit* (nearest example belonging to the class under consideration or sometimes referred to as the nearest target neighbor) and the nearest *miss* (nearest example belonging to other classes).

RELIEF learns only a diagonal matrix in the original setting. However, Sun and Wu [102] have extended *RELIEF* to learn a full distance metric matrix. They have further proved that *RELIEF* is an online algorithm and have shown that *RELIEF* outperforms standard kNN algorithm on many standard datasets.

Let $x^{(i)}$ be a vector in R^p having $y^{(i)}$ as the class label with values $+1, -1$. Let A be a vector meant for iteratively estimating the qualities of attributes initialized with 0. The aim is to learn A on a set of training examples. Suppose an example $x^{(i)}$ is randomly selected. This is followed by finding the two nearest neighbors of $x^{(i)}$: one from the same class (termed as the *nearest hit* or H) and other from the different class than that of $x^{(i)}$ (termed as the *nearest miss* or M). The update rule in case of *RELIEF* doesn't depend on any condition unlike *SiLA*.

The *RELIEF* algorithm is presented next:

RELIEF (k=1)

Input: training set $((x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)}))$ of n vectors in $\mathbb{R}^p$, number of epochs J;
Output: the vector A of estimations of the qualities of attributes

Initialization $\forall\, m\ 1 \leq m \leq p,\ A_m = 0$

Repeat J times (epochs)

1. randomly select an instance $x^{(i)}$
2. find nearest hit H and nearest miss M
3. for $l = 1, \cdots, p$
4. $A_l = A_l - \frac{\text{diff}(l, x^{(i)}, H)}{J} + \frac{\text{diff}(l, x^{(i)}, M)}{J}$

where J represents the number of iterations, the algorithm has been run while *diff* is a function used to find the difference between the values of an attribute l for $x^{(i)}$ and the nearest hit or miss represented by H or M.

4.4.2 Comparison between *SiLA* and *RELIEF*

While comparing the two algorithms *SiLA* and *RELIEF*, it can be noted that *RELIEF* learns a vector of weights while *SiLA* learns a sequence of vectors where each vector has got a corresponding weight which signifies the number of examples correctly classified while using that particular vector. Furthermore, the weight vector is updated systematically in case of *RELIEF* while a vector is updated for *SiLA* only if it has failed to correctly classify the current example $x^{(i)}$ (i.e. $s_A(x^{(i)}, y) - s_A(x^{(i)}, z) \leq 0$). In this case, a new vector A is created and its corresponding weight is initialized to 1. However, in the case of a correct classification for *SiLA*, the weight associated with the current vector A is increased by 1. Moreover, the two algorithms find the nearest hit and the nearest miss to update the vector A. *RELIEF* selects an instance randomly whereas *SiLA* uses the instances in a systematic way. Another difference between the two algorithms is that in case of *RELIEF*, the vector A is updated based on the difference (distance) while it is updated based on the similarity function for *SiLA*. This explains the fact that the impact of nearest hit is subtracted for *RELIEF* while the impact for nearest miss is added to the vector A. For *SiLA*, the impact of the nearest hit is added while that of the nearest miss is subtracted from the current vector A.

The worst time complexity of *SiLA* is $O(Mnp^2)$ whereas for *RELIEF*, it is $O(Mnp)$ and is thus lesser than that for *SiLA*. Here M represents the number of iterations, p is the number of features while n represents the total number of instances. Moreover, the complexity for *RELIEF* is fixed for all of the scenarios unlike *SiLA* where it depends on the number of mistakes made.

SiLA tries to directly reduce the leave-one-out error also known as the $0-1$ loss. However, *RELIEF* uses a linear utility function in such a way that the average margin is maximized.

4.4.3 *RELIEF*-Based Similarity Learning Algorithm - *RBS*

In this subsection, a *RELIEF*-Based Similarity learning algorithm (*RBS*) [90] is proposed which is based on *RELIEF* algorithm. However, the interest, here lies in similarities instead of distances like *SiLA*. The aim, just like that of *RELIEF*, is to maximize the margin $\mathcal{M}(A)$ between the *target* neighbors (represented by y) and the *impostors* (given by z). The margin, for $k = 1$ in *kNN* algorithm can be written as:

$$\begin{aligned}\mathcal{M}(A) &= \sum_{i=1}^{n} \left(s_A(x^{(i)}, y^{(i)}) - s_A(x^{(i)}, z^{(i)})\right) \\ &= \sum_{i=1}^{n} \left({x^{(i)}}^t A y^{(i)} - {x^{(i)}}^t A z^{(i)}\right) = \sum_{i=1}^{n} {x^{(i)}}^t A (y^{(i)} - z^{(i)})\end{aligned}$$

where A is the similarity matrix. The margin is maximized subject to the constraint $\|A\|_F^2 = 1$.

$$\begin{aligned}&\arg\max_{A} && \mathcal{M}(A) \\ &\text{subject to} && \|A\|_F^2 = 1,\end{aligned}$$

Taking the Lagrangian of the matrix A:

$$\mathcal{L}(A) = \sum_{i=1}^{n} x^{(i)^t} A(y^{(i)} - z^{(i)}) + \lambda(1 - \sum_{l=1}^{p}\sum_{m=1}^{p} a_{lm}^2)$$

where λ is a Lagrangian multiplier. Taking the derivative with respect to a_{lm} and setting it to zero yields:

$$\frac{\partial \mathcal{L}(A)}{\partial a_{lm}} = \sum_{i=1}^{n} x_l^{(i)}(y_m^{(i)} - z_m^{(i)}) - 2\lambda a_{lm} = 0$$

$$\Rightarrow a_{lm} = \frac{\sum_{i=1}^{n} x_l^{(i)}(y_m^{(i)} - z_m^{(i)})}{2\lambda}$$

Since the Frobenius norm of matrix A is 1:

$$\sum_{l=1}^{p}\sum_{m=1}^{p} a_{lm}^2 = 1$$

$$\Rightarrow \sum_{l=1}^{p}\sum_{m=1}^{p} a_{lm}^2 = \sum_{l=1}^{p}\sum_{m=1}^{p} \left(\frac{\sum_{i=1}^{n} x_l^{(i)}(y_m^{(i)} - z_m^{(i)})}{2\lambda} \right)^2$$

Now the value of 2λ can be computed in the following manner:

$$2\lambda = \sqrt{\sum_{l=1}^{p}\sum_{m=1}^{p} \left(\sum_{i=1}^{n} x_l^{(i)}(y_m^{(i)} - z_m^{(i)}) \right)}$$

In case of a diagonal matrix, m is replaced with l and 2λ becomes equal to:

$$2\lambda = \sqrt{\sum_{l=1}^{p} \left(\sum_{i=1}^{n} x_l^{(i)}(y_l^{(i)} - z_l^{(i)}) \right)}$$

Furthermore, the margin for $k > 1$ can be written as:

$$\mathcal{M}(A) = \sum_{i=1}^{n} \left(\sum_{q=1}^{k} s_A(x^{(i)}, y^{(i),q}) - \sum_{q=1}^{k} s_A(x^{(i)}, z^{(i),q}) \right)$$

$$= \sum_{i=1}^{n} \left(x^{(i)^t} A \sum_{q=1}^{k} (y^{(i),q} - z^{(i),q}) \right)$$

where $y^{(i),q}$ represents the qth nearest neighbor of $x^{(i)}$. Moreover, a_{lm} and 2λ can be written as:

$$a_{lm} = \frac{\sum_{i=1}^{n} x_l^{(i)} \sum_{q=1}^{k} (y_m{}^{(i),q} - z_m{}^{(i),q})}{2\lambda}$$

$$2\lambda = \sqrt{\sum_{l=1}^{p}\sum_{m=1}^{p} \left(\sum_{i=1}^{n} x_l^{(i)} \sum_{q=1}^{k} (y_m{}^{(i),q} - z_m{}^{(i),q}) \right)}$$

It can be further noted that a_{lm} is inversely proportional to the Lagrangian multiplier λ.

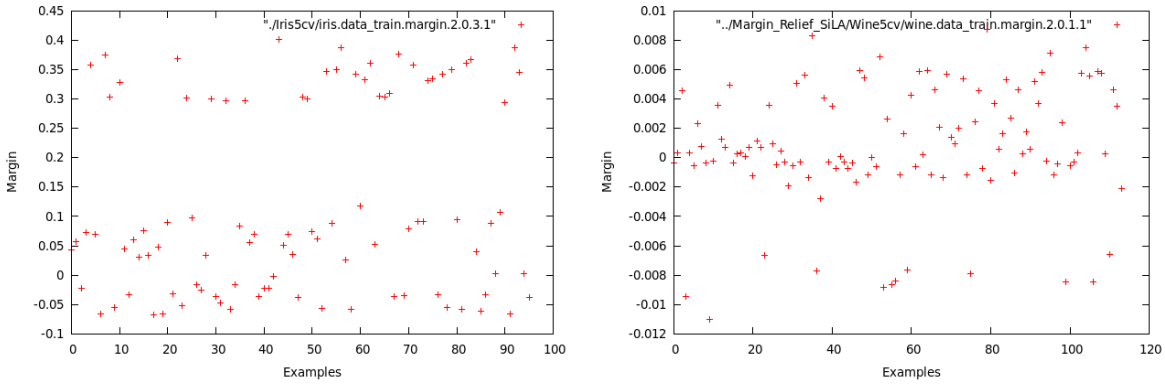

Figure 4.3: Margin for *RBS* on *Iris* (left) and *Wine* (right) datasets

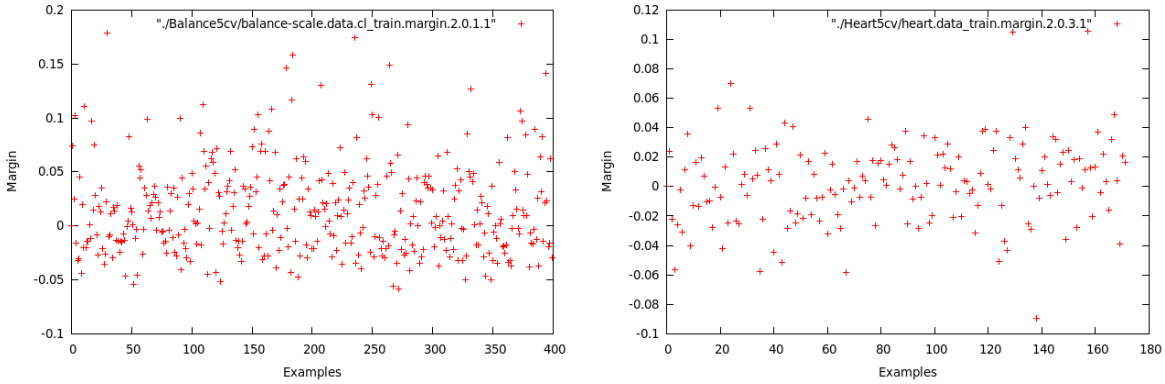

Figure 4.4: Margin for *RBS* on *Balance* (left) and *Heart* (right) datasets

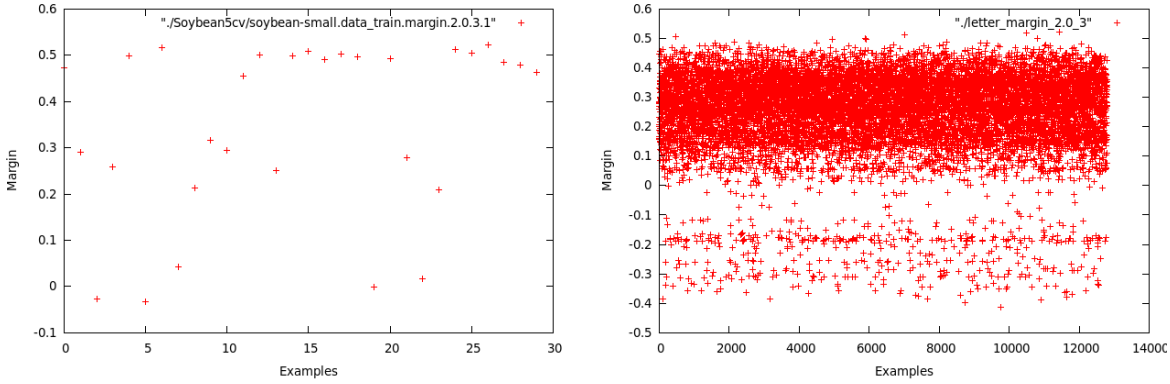

Figure 4.5: Margin for *RBS* on *Soybean* (left) and *Letter* (right) datasets

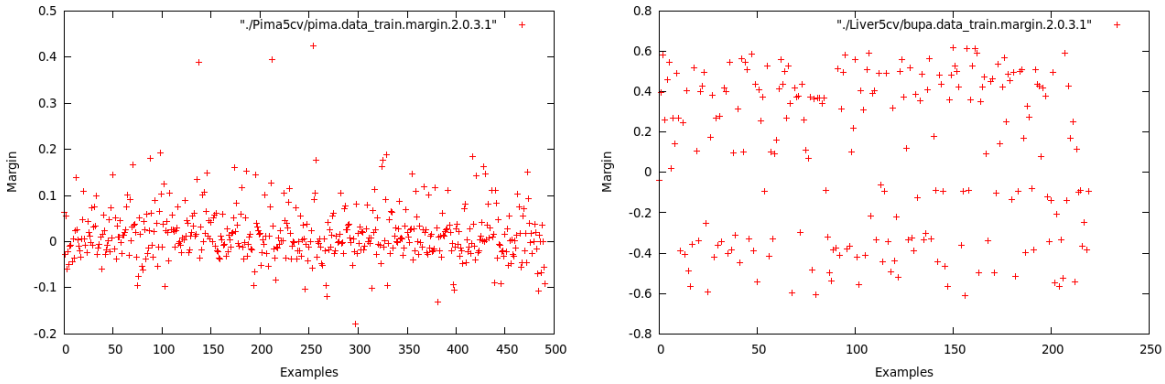

Figure 4.6: Margin for *RBS* on *Pima* (left) and *Liver* (right) datasets

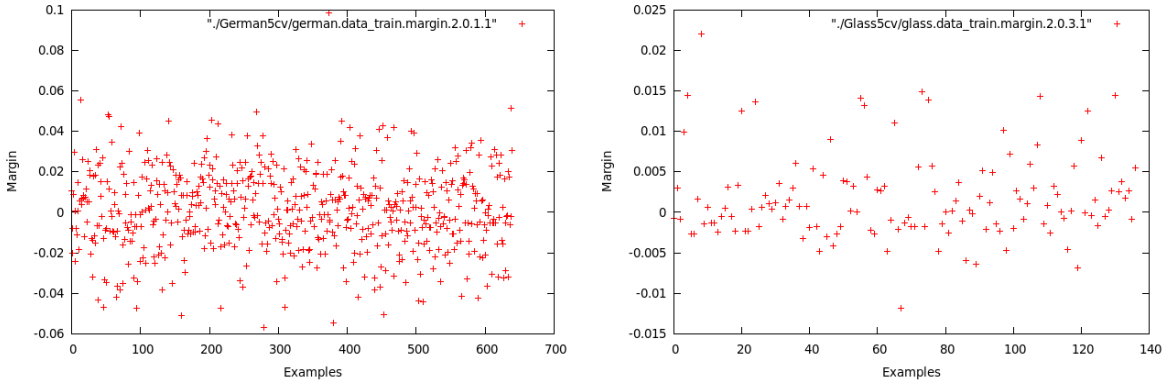

Figure 4.7: Margin for *RBS* on *German* (left) and *Glass* (right) datasets

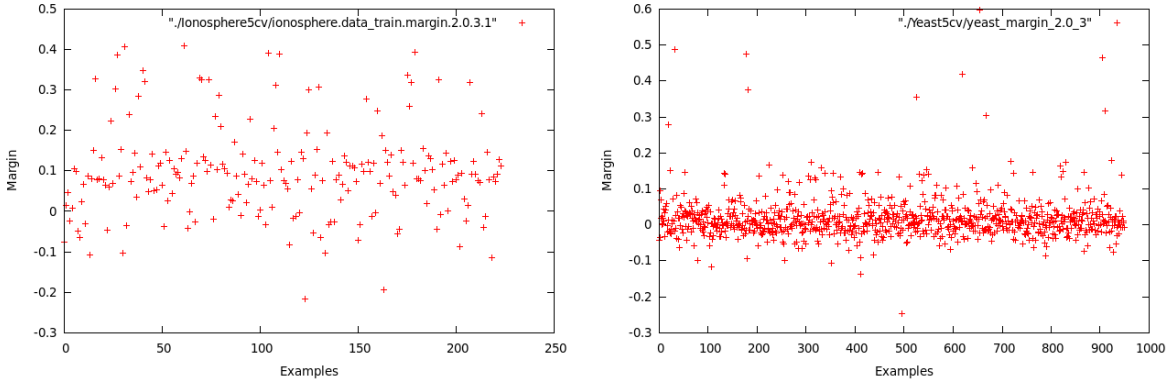

Figure 4.8: Margin for *RBS* on *Ionosphere* (left) and *Yeast* (right) datasets

4.4.4 Problems with *RELIEF* based techniques

The problem with the *RELIEF* based approaches (*RELIEF* and *RBS*) is that as one strives to maximize the margin, it is possible that the overall margin is quite large but in reality the algorithm has made a certain number of mistakes (characterized with negative margin). This concept was verified on a number of standard UCI datasets [36] *Iris*, *Wine*, *Balance*, *Heart*, *Soybean*, *Letter*, *Pima*, *Liver*, *German*, *Glass* , *Ionosphere* and *Yeast*, as can be seen from figures 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8. It can be observed that in most of these figures, the average margin remains positive despite the presence of a number of mistakes, since the positive margin is much greater than the negative one for the majority of the examples. For example, in figure 4.3, the values of negative margin for *Iris* are in between -0.10 and 0.0, whereas most of the positive margin values are greater than 0.25. Similarly, for *Wine* (figure 4.3), most of the negative margin values lie in the range between 0.0 and -0.002 while the positive margin values are mostly dispersed in the range $0 - 0.08$. Therefore, despite the fact that the overall margin is large, a lot of examples are misclassified. A similar story in portrayed in figure 4.4 for *Balance*, where most of the examples having negative margin values have a margin in between -0.05 and 0.0. On the other hand, the positive margin values are dispersed between 0.0 and 0.1. The positive as well as negative margin values for *Heart* (see figure 4.4), *Liver* (figure 4.6) and *German* 4.7) have the same range but the number of examples having positive margin values is greater than the ones having negative margin values.

This explains the fact that the algorithms *RELIEF* and *RBS* did not perform quite well on different standard test collections (see Chapter 6).

4.4.5 A stricter version: *sRBS*

A work around to improve the performance of *RELIEF* based methods is to directly use the leave-one-out error or $0-1$ loss like the original *SiLA* algorithm where the aim is to reduce the number of mistakes on unseen examples. The resulting algorithm is a stricter version of *RELIEF*-Based Similarity Learning Algorithm and is termed as *sRBS*. It is called as a *stricter* version as we do not try to maximize the overall margin but are interested in reducing the individual errors on the unseen examples.

The cost function for *sRBS* can be described in terms of a sigmoid function.

$$\sigma_A(x^{(i)}) = \frac{1}{1 + exp(\beta x^{(i)^t} A(y^{(i)} - z^{(i)}))}$$

As β approaches ∞, the sigmoid function represents the $0-1$ loss: it approaches 0 when the margin $x^{(i)}A(y^{(i)} - z^{(i)})$ is positive and approaches 1 in the case where the margin is negative. Let $g_A(i)$ represents $exp(\beta x^{(i)^t} A(y^{(i)} - z^{(i)}))$ while v represents $y - z$. The cost function being considered here is based on the above sigmoid function, regularized with the Frobenius norm of A:

$$\arg\min_A \varepsilon(A) = \sum_{i=1}^{n} \sigma_A(x^{(i)}) + \lambda \|A\|_2^2$$

Taking the derivative with respect to a_{lm}:

$$\frac{\partial \varepsilon(A)}{\partial a_{lm}} = -\beta \sum_{i=1}^{n} \frac{x_l^{(i)} v_m^{(i)} g_A(i)}{(1+g_A(i))^2} + 2\lambda a_{lm}$$

$\forall\, l, m, 1 \geq l \geq p, 1 \geq m \geq p,$

$$2\lambda a_{lm} = -\beta \sum_{i=1}^{n} \frac{x_l^{(i)} v_m^{(i)} g_A(i)}{(1+g_A(i))^2}$$

No closed form solution for this fixed point equation is already known. However, this equation can be solved with gradient descent methods. The cost function in the case of gradient descent can be written as:

$$\begin{aligned} \varepsilon(A) &= \sum_{i=1}^{n} \frac{1}{1+g_A(i)} + \lambda \textstyle\sum_{lm} a_{lm}^2 \\ &= \sum_{i=1}^{n} \left[\frac{1}{1+g_A(i)} + \frac{\lambda}{n} \textstyle\sum_{lm} a_{lm}^2 \right] \\ &= \sum_{i=1}^{n} Q_i(A) \end{aligned}$$

The derivative is taken with respect to a_{lm}:

$$(\nabla Q_i(A))_{lm} = \frac{\partial Q_i(A)}{\partial a_{lm}} = \frac{-\beta x_l^{(i)} v_m^{(i)} g_A(i)}{(1+g_A(i))^2} + \frac{2\lambda a_{lm}}{n}$$

With this, the update step for A_{lm}^t can be defined as:

$$A_{lm}^{t+1} = A_{lm}^t - \frac{\alpha^t}{n} \sum_{i=1}^{n} \frac{\partial Q_i(A^t)}{\partial a_{lm}}$$

where α^t stands for the learning rate and is given by: $\alpha^t = \frac{1}{t}$. The learning rate is inversely proportional to the number of iterations and decreases with the increase in the number of epochs. *sRBS* algorithm is next presented:

sRBS **- Training**
Input: training set $((x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)}))$ of n vectors in $\mathbb{R}^p$, A_{lm}^1 denotes the element of A^1 at row l and column m
Output: Matrix A
Initialization $t = 1, A^{(1)} = 1$ (Unity matrix)
Repeat J **times (epochs)**
1. For all of the features l, m
   2. $\text{Minus}_{lm} = 0$
3. for $i = 1, \cdots, n$
   4. For all of the features l, m
      5. $\text{Minus}_{lm} += \frac{\partial Q_i(A^t)}{\partial a_{lm}}$

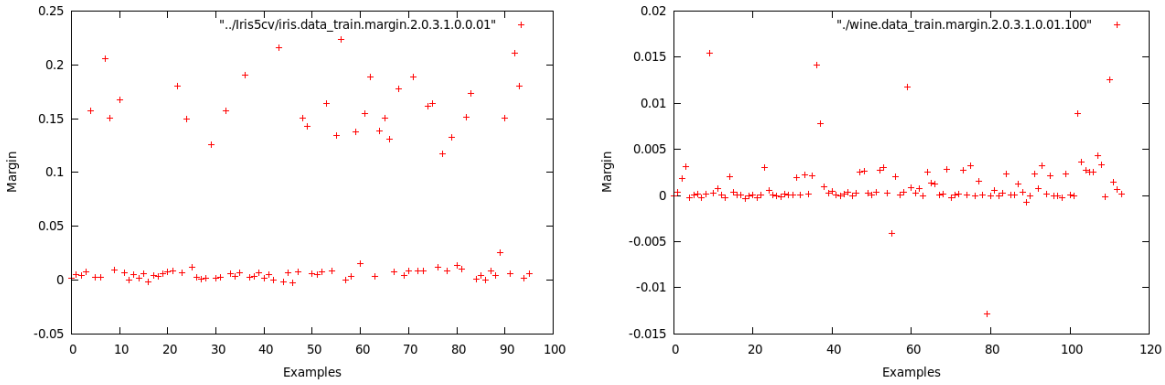

Figure 4.9: Margin for *sRBS* on *Iris* (left) and *Wine* (right) datasets

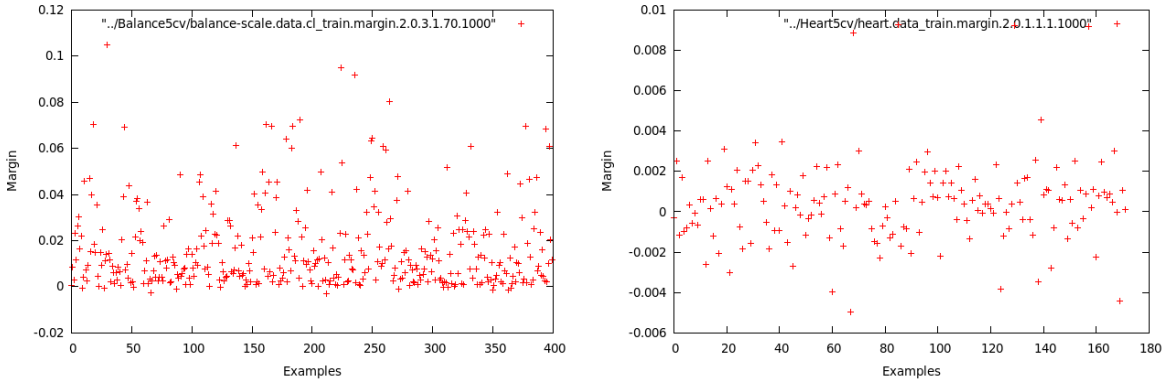

Figure 4.10: Margin for *sRBS* on *Balance* (left) and *Heart* (right) datasets

6. $A_{lm}^{t+1} = A_{lm}^{t} - \frac{\alpha^t}{n} * \text{Minus}_{lm}$
7. If $\sum_{lm} |A_{lm}^{t+1} - A_{lm}^{t}| \leq \gamma$
8. Stop

During each epoch, the difference between the new similarity matrix A_{lm}^{t+1} and the current one A_{lm}^{t} is computed. If the difference is less than a certain threshold (γ), the algorithm is stopped. The range of γ was between 10^{-3} and 10^{-4}.

Figures 4.9, 4.10, 4.11, 4.12, 4.13, 4.14 show the margin values for the training examples of different UCI datasets, once the training phase of *sRBS* algorithm has been completed. These figures can be compared with the earlier ones for *RBS* algorithm to observe that the training phase of *sRBS* is more effective than the one for *RBS* e.g. for *Iris* (figure 4.9), *Wine* (figure 4.9), *Balance* (figure 4.10), *Pima* (figure 4.12), *Glass* (figure 4.13), *Yeast* (figure 4.14), there are only a very few errors although a lot of examples have a margin close to 0.0. There are no errors (no example with a negative margin) for *Soybean* (figure 4.11). Moreover, the algorithm *sRBS* makes a lot of mistakes for *Letter* as depicted in figure 4.11.

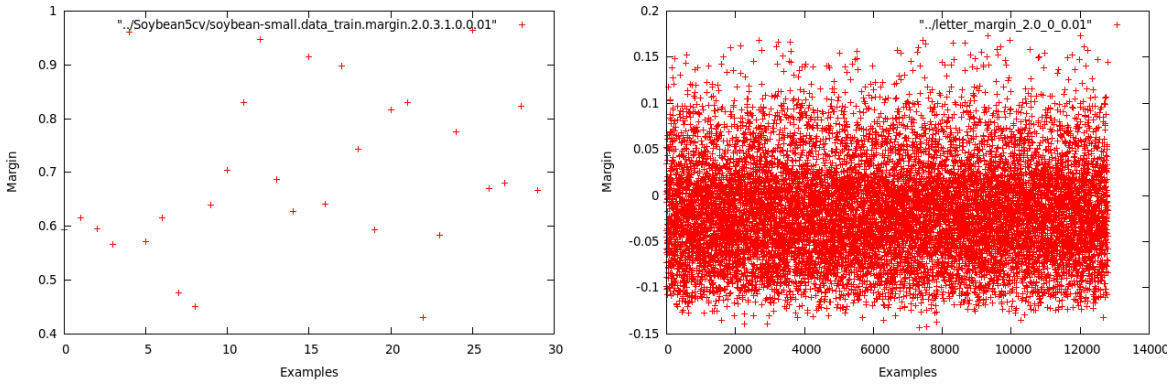

Figure 4.11: Margin for *sRBS* on *Soybean* (left) and *Letter* (right) datasets

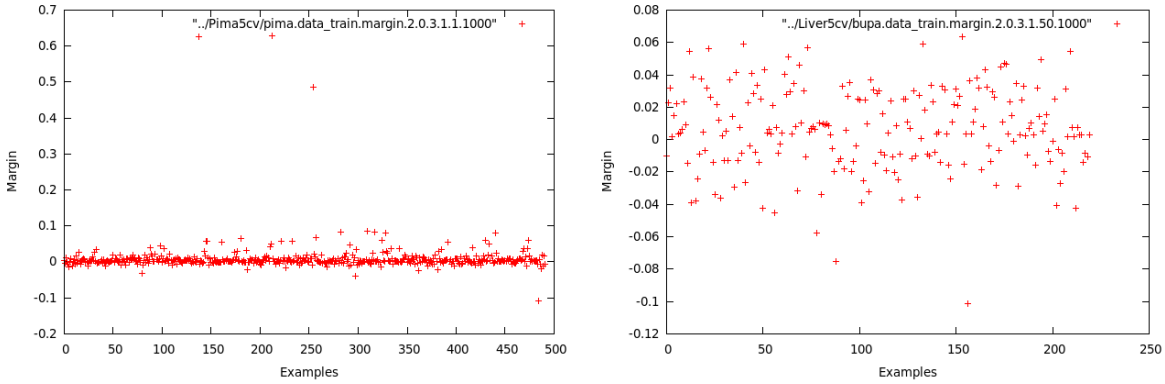

Figure 4.12: Margin for *sRBS* on *Pima* (left) and *Liver* (right) datasets

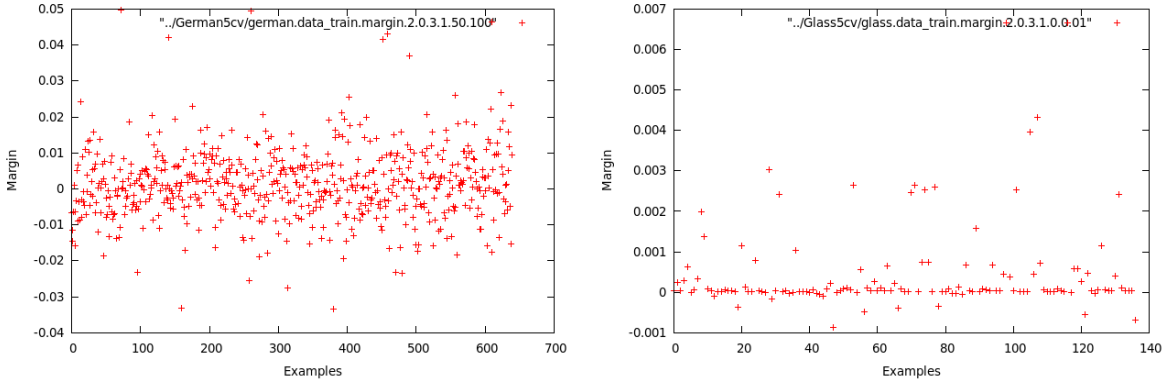

Figure 4.13: Margin for *sRBS* on *German* (left) and *Glass* (right) datasets

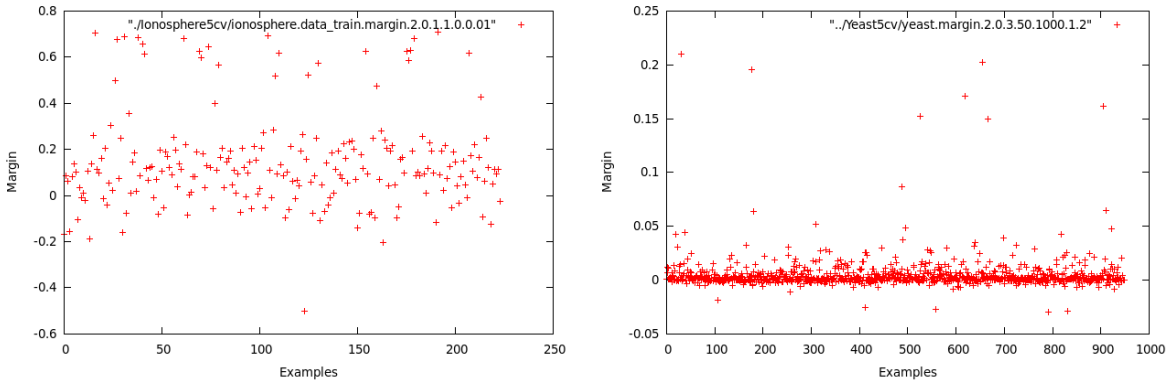

Figure 4.14: Margin for *sRBS* on *Ionosphere* (left) and *Yeast* (right) datasets

4.4.6 Effect of Positive, Semi-Definitiveness on *RELIEF* based algorithms

The similarity $x^t Ax$ in the case of *RELIEF* based algorithms does not correspond to a symmetric bi-linear form, and hence a scalar product. The work around lies in projecting the similarity matrix A onto the set of positive, semi-definite (PSD) matrices just like *eSiLA* (see section 4.3). A similarity matrix can be projected by finding an eigenvector decomposition followed by the selection of positive eigenvalues. A PSD matrix A is written as:

$$A \succeq 0$$

In case, where a diagonal matrix is learned by *RELIEF*, positive semi-definitiveness can be achieved by selecting only the positive entries of the diagonal. Moreover for learning a full matrix with *RELIEF*, the projection can be performed in the following manner:

$$A = \sum_{j,\lambda_j>0} \lambda_j u_j u_j^t$$

where λ_j and u_j are the eigenvalues and eigenvectors of A.

Similarly, *RBS* is transformed into *RBS-PSD* by incorporating an additional constraint that the similarity matrix A must be PSD, while maximizing the margin [91].

It is verified that despite the fact that the overall margin is quite large, *RBS-PSD* makes a number of mistakes characterized with negative margin. This concept was verified on a number of standard UCI datasets [36] i.e. *Iris*, *Wine*, *Balance*, *Heart*, *Soybean*, *Letter*, *Pima*, *Liver*, *Glass*, *Ionosphere* and *Yeast* as can be seen from figures 4.15, 4.16, 4.17, 4.18, 4.19, 4.20. It can be observed for all of the datasets that the average margin remains positive despite the presence of a number of mistakes, since the positive margin is much greater than the negative one for the majority of the test examples. For example, the values of negative margin in the case of *Iris* (see figure 4.15) is in the range of $-0.05 - 0.00$ whereas there are many positive margin values greater than 0.175. Similarly, for *Wine* (figure 4.16), most of the negative margin values lie in the range between -0.002 and 0 while most of the positive margin values are dispersed in the range $0 - 0.004$. In case of *Balance* (figure 4.16), the negative values are seen in the range of $-0.05 - 0.00$ whereas the positive margin values are mostly scattered between 0 and 0.1. While looking on the results for *Letter* (figure 4.17), one can note that while the negative margin values

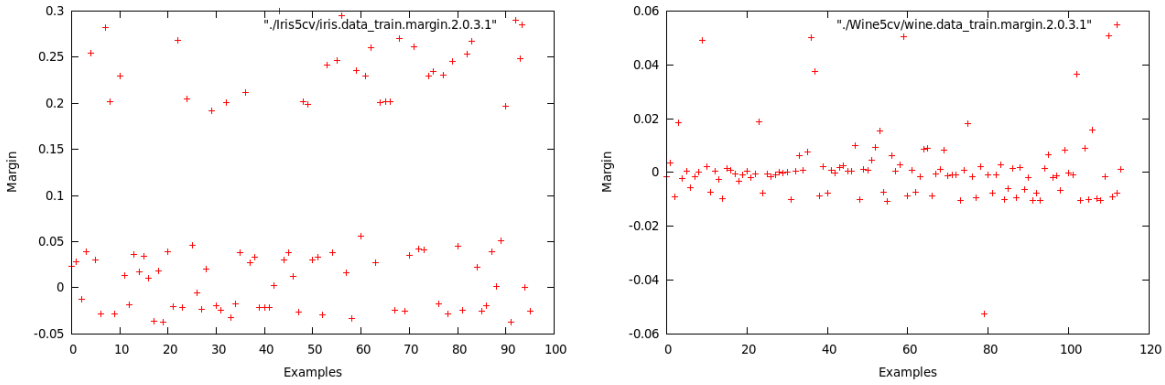

Figure 4.15: Margin for *RBS-PSD* on *Iris* (left) and *Wine* (right) datasets

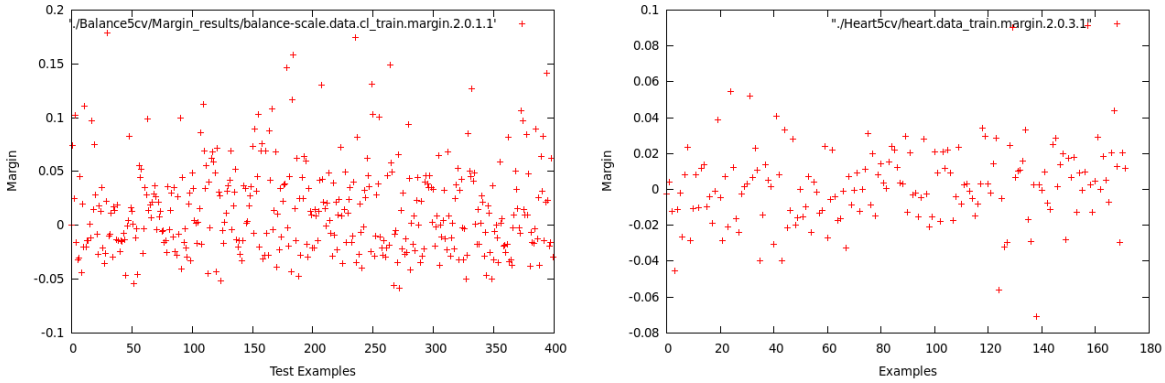

Figure 4.16: Margin for *RBS-PSD* on *Balance* (left) and *Heart* (right) datasets

lie between -0.1 and 0.0, the positive margin values are mostly seen between 0.0 and 0.15. So, despite the fact that the overall margin is large, a lot of examples are misclassified as was seen earlier for the *RBS* algorithm. Observing the figures for *RBS* and *RBS-PSD*, one can easily note that there are very few differences between the results for the two algorithms except *Ionosphere* in which case *RBS-PSD* performs better as compared to *RBS*.

However, for *Iris*, the range of negative margin values increases whereas the range for the positive margin values decreases for *RBS-PSD* as compared to *RBS*. Similar phenomenon is repeated for *Letter*, *Liver*, *Glass* and *Yeast*. This effectively means that *RBS* is better than its counterpart for these data sets as the overall margin decrease in all of these cases.

This explains the fact that the algorithms *RELIEF* and *RBS-PSD* did not perform quite well on different standard test collections as can be seen in Chapter 5.

Once the effect of PSD matrices on *RBS* has been covered in detail, the next obvious question is the effect of PSD matrices on *sRBS*. As seen from figures 4.21, 4.22, 4.23, 4.24, 4.25, 4.26 adding positive, semi-definite constraints in *sRBS* does not has any good effects except for *Ionosphere*. Similarly, *sRBS-PSD* performs better than *RBS-PSD* for *Iris*, *Wine*, *Balance*, *Soybean*, *Pima*, *Glass*, *Ionosphere* and *Yeast*.

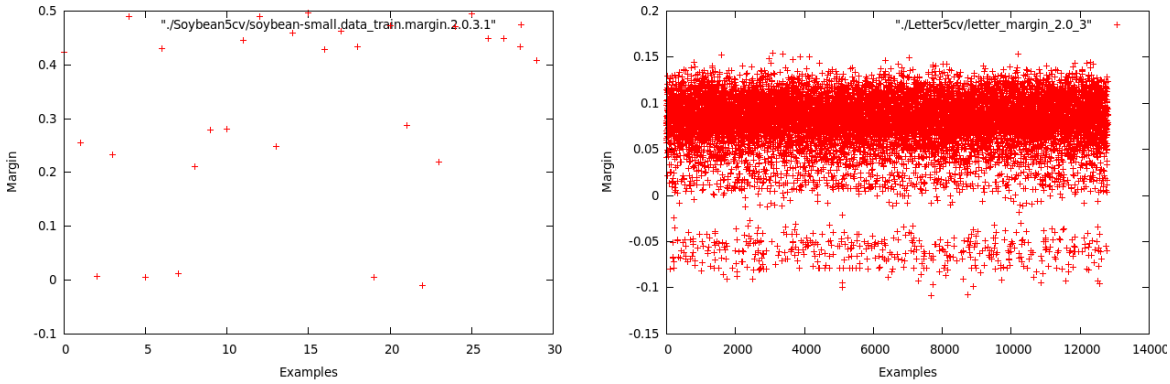

Figure 4.17: Margin for *RBS-PSD* on *Soybean* (left) and *Letter* (right) datasets

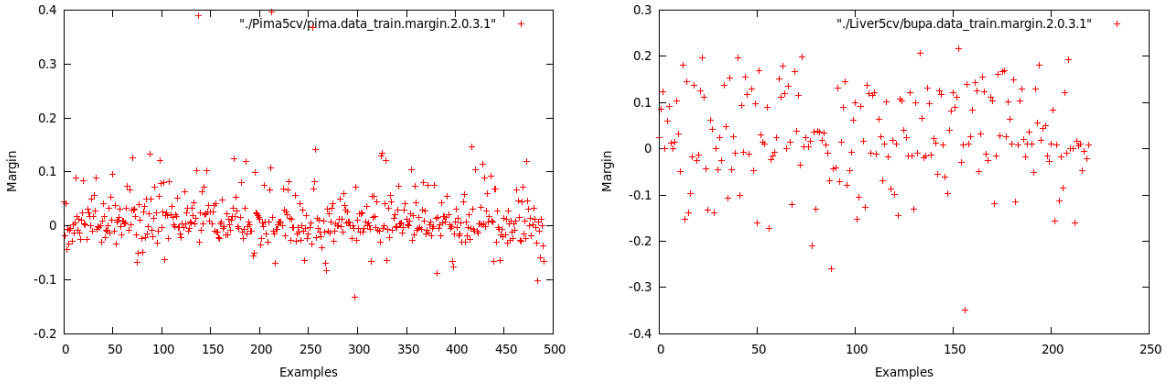

Figure 4.18: Margin for *RBS-PSD* on *Pima* (left) and *Liver* (right) datasets

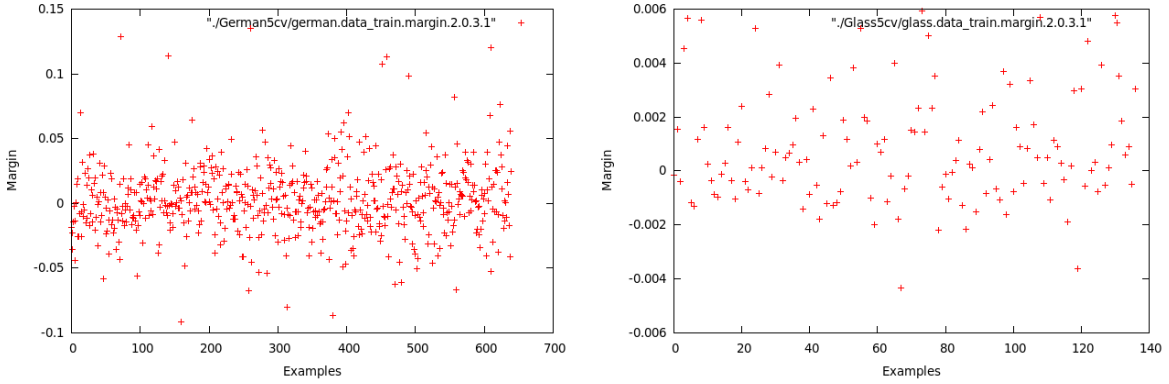

Figure 4.19: Margin for *RBS-PSD* on *German* (left) and *Glass* (right) datasets

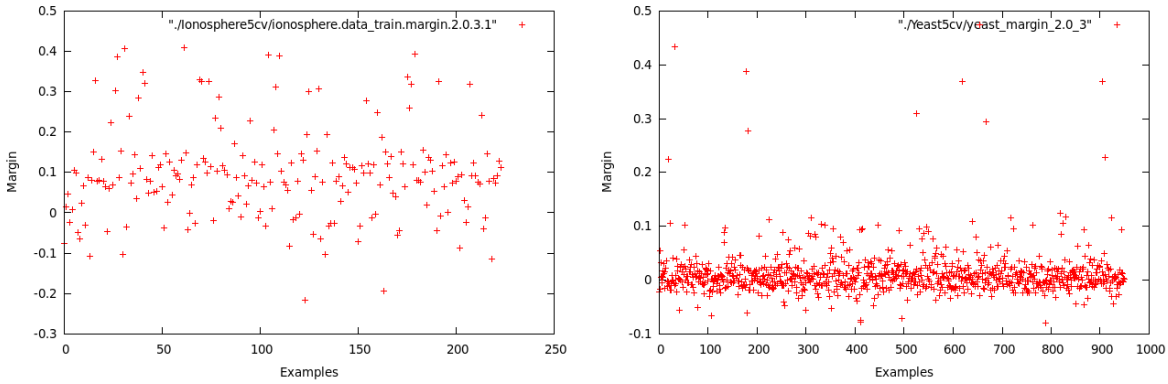

Figure 4.20: Margin for *RBS-PSD* on *Ionosphere* (left) and *Yeast* (right) datasets

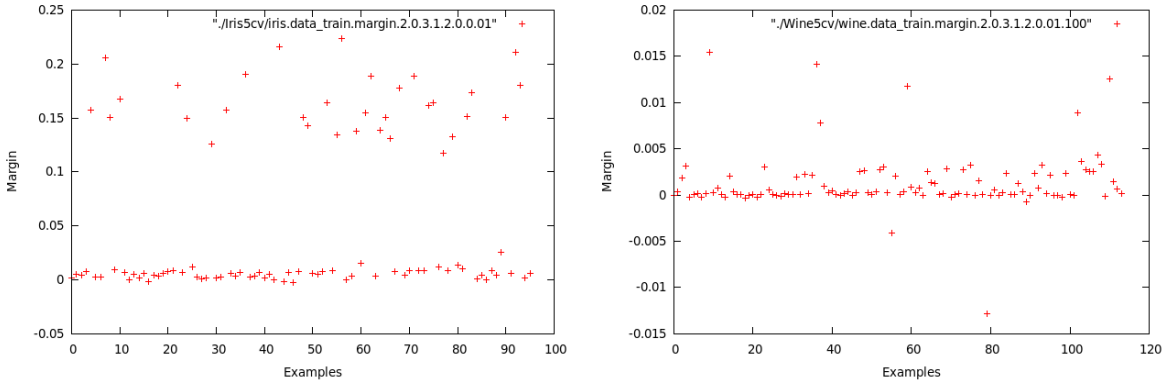

Figure 4.21: Margin for *sRBS-PSD* on *Iris* (left) and *Wine* (right) datasets

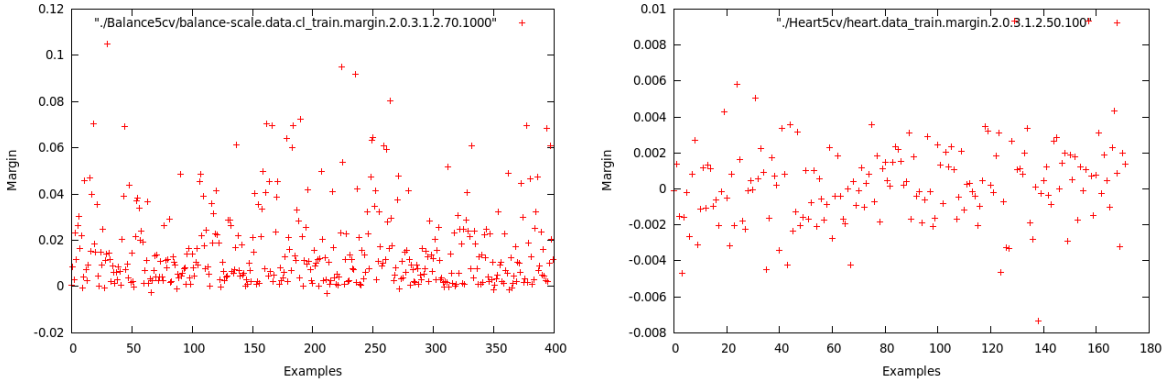

Figure 4.22: Margin for *sRBS-PSD* on *Balance* (left) and *Heart* (right) datasets

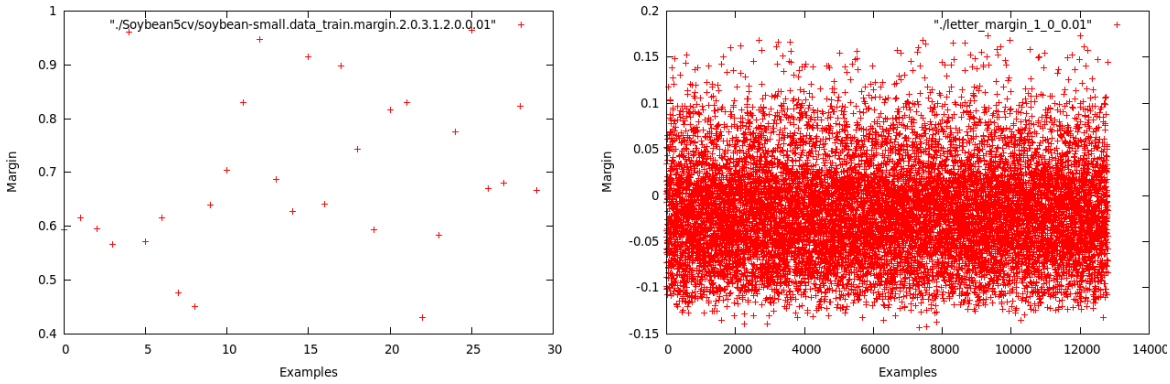

Figure 4.23: Margin for *sRBS-PSD* on *Soybean* (left) and *Letter* (right) datasets

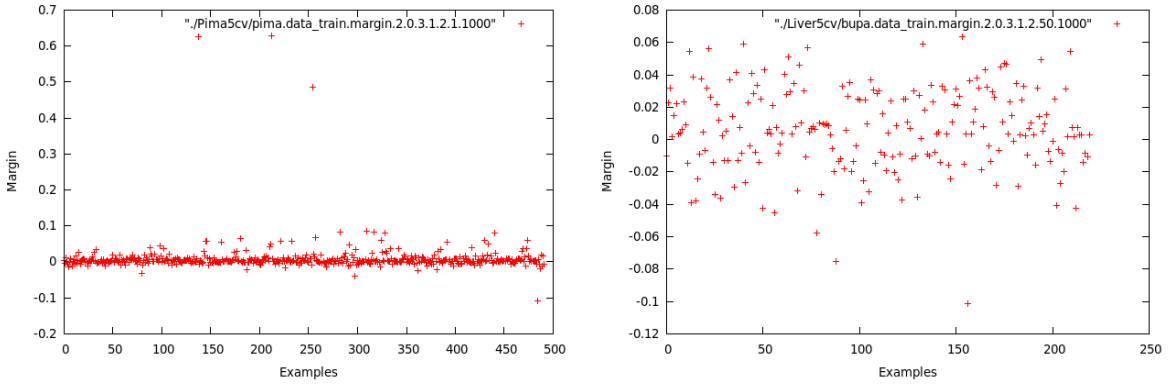

Figure 4.24: Margin for *sRBS-PSD* on *Pima* (left) and *Liver* (right) datasets

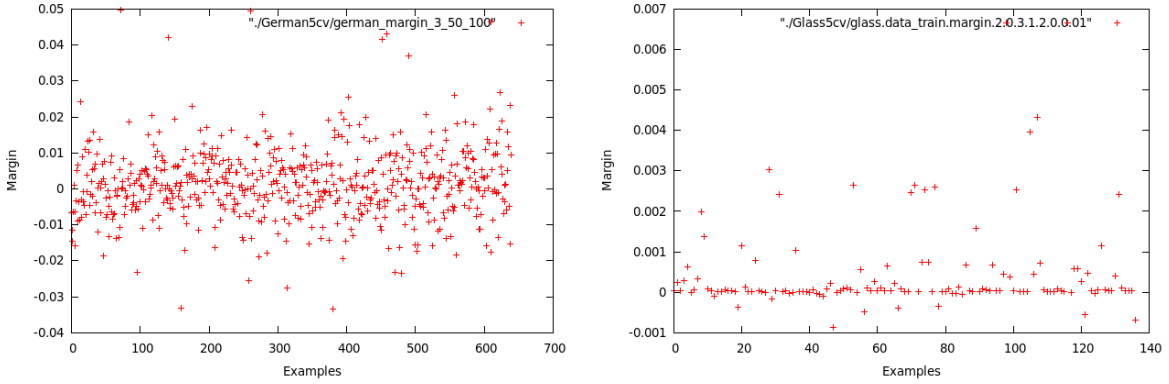

Figure 4.25: Margin for *sRBS-PSD* on *German* (left) and *Glass* (right) datasets

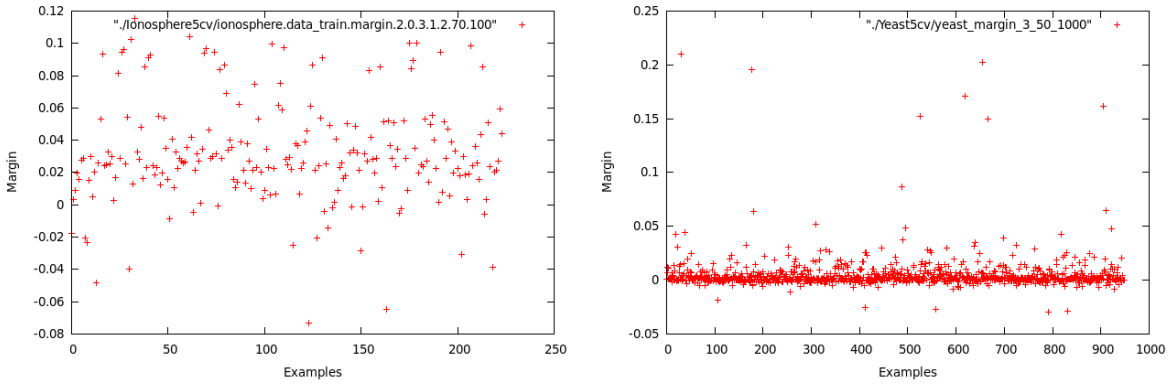

Figure 4.26: Margin for *sRBS-PSD* on *Ionosphere* (left) and *Yeast* (right) datasets

4.5 Generalized Cosine Similarity Metric Learning

The similarity measure given in equation 4.1 does not refers to a generalized cosine similarity since the normalization is completely independent of the similarity matrix. This is the motivation behind defining a generalized cosine similarity metric learning algorithm where the normalization is dependent on the similarity matrix and the similarity matrix is positive, semi-definite (PSD). In order to make a similarity matrix as positive, semi-definite, the similarity matrix is projected onto the set of positive, semi-definite matrices (PSD) inspired from the strategy given in *POLA* [99].

Since *POLA* considers the examples in the form of pairs, with each pair being either similar (e.g. belonging to same class) or dissimilar, and learns the distance metric based on the pairwise constraints (equivalence and inequivalence), the same strategy is followed in the case of generalized cosine similarity metric learning. Furthermore, similarity is learned in a *global* sense with the aim of satisfying *all* of the pairwise constraints simultaneously.

4.5.1 Problem Setting

The generalized similarity between two examples x and x' in $\mathbb{R}^p$, as given in equation 4.2 is rewritten:

$$s_A(x, x') = \frac{x^t A x'}{\sqrt{x^t A x}\sqrt{x'^t A x'}}$$

where $A \geq 0$ is a positive, semi-definite matrix and the normalization is dependent on A. One can also note that by choosing A as the identity matrix, equation 4.2 becomes the standard cosine similarity. Other positive, semi-definite matrices define different scalar products and norms, so that equation 4.2 corresponds to a cosine in a new basis of the underlying vector space. Because of this property, equation 4.2 refers to the family of Generalized Cosine Similarities [86].

The examples considered here, are in the form of tuples, (x, x', y) where each example is composed of the instance pair (x, x') and a label y which is $+1$ when x and x' are similar and is -1 in the case when they are dissimilar. When the data is separable, the margin of a sample, S, denoted by 2γ, is defined as the minimum separation between all pairs of similar $(x_1, x'_1, +1)$ and dissimilar $(x_2, x'_2, -1)$ examples:

$$s_A(x_1, x'_1) - s_A(x_2, x'_2) \geq 2\gamma$$

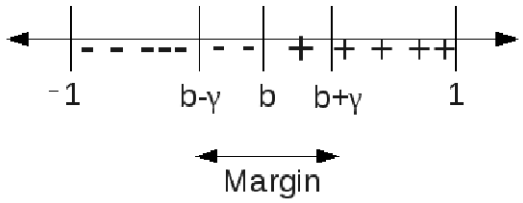

Figure 4.27: Separation between similar and dissimilar examples

By introducing a threshold $b \in \mathbb{R}$, the above inequality can be rewritten as:

$$\forall(x, x', y) : y = +1 \Rightarrow s_A(x, x') \geq b + \gamma$$
$$\forall(x, x', y) : y = -1 \Rightarrow s_A(x, x') \leq b - \gamma$$

where $\gamma > 0$ and $-1 + \gamma \leq b \leq 1 - \gamma$. Here, γ measures the extent to which one is on the wrong side of the threshold. The two inequalities can be combined to form a single linear constraint:

$$y(b - s_A(x, x')) \leq -\gamma \tag{4.3}$$

Figure 4.27 shows the similar and dissimilar example pairs separated by a margin γ. Considering tuples of the form $(x_\tau, x'_\tau, y_\tau)$, at each time step, or round τ, the loss incurred by the current matrix-threshold pair (A, b) can be computed as follows:

$$l_\tau(A, b) = \max\{0,\, y_\tau(b - s_A(x_\tau, x'_\tau)) + \gamma\}$$

which is a variant of the hinge loss. Our goal is thus to find a matrix-threshold pair (A, b) which minimizes the overall loss. When the data is separable, there exists a matrix-threshold pair such that the overall loss is 0 (as inequality 4.3 holds for matrix-threshold pairs separating the data). If $l_\tau = 0$, the following inequality holds:

$$y_\tau(b - s_A(x_\tau, x'_\tau)) + \gamma \leq 0$$

which can be rewritten as:

$$y_\tau(s_A(x_\tau, x'_\tau) - b) \geq \gamma$$

An online algorithm is presented next, in order to learn a matrix-threshold pair. In the first instance, the data is considered to be separable. The case where the data is inseparable is presented afterwards.

4.5.2 gCosLA - An online generalized Cosine similarity metric Learning Algorithm

In the case where the data is separable:

$$\exists A \succeq 0,$$

and

$$\exists b,\ -1 + \gamma \leq b \leq 1 - \gamma$$

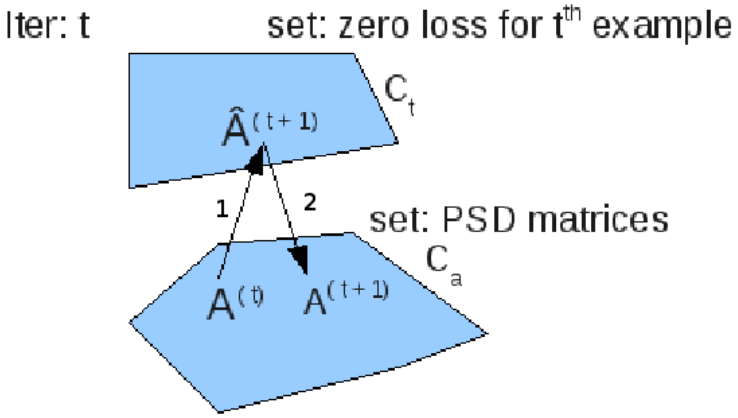

Figure 4.28: Set of projections for gCosLA

such that the matrix-threshold pair (A, b) completely separates the data, i.e. has zero loss for all time steps. Because the matrix A should separate the data and be, at the same time, positive, semi-definite, one can rely on a strategy based on first finding a matrix-threshold pair with zero loss and close to the current matrix-threshold pair so that the new matrix not only correctly classifies the new example but also the examples already considered so far. This is followed by projecting the obtained matrix on the set of positive, semi-definite matrices (an approach reminiscent of the one defined in *POLA* [99]). The first step aims at finding matrix-threshold pairs with small loss, whereas the second step ensures the fact that the obtained matrix is positive, semi-definite and hence defines a valid generalized cosine similarity.

Let $C_\tau \subset \mathbb{R}^{n^2+1}$ be the set of all matrix-threshold pairs having zero loss on the example $(x_\tau, x'_\tau, y_\tau)$:

$$C_\tau = \{(A, b) \in \mathbb{R}^{n^2+1} \ : \ l_\tau(A, b) = 0\}$$

C_a can then be defined as the set of all admissible matrix-threshold pairs:

$$C_a = \{(A, b) \in \mathbb{R}^{n^2+1} : A \succeq 0, \ -1 + \gamma \leq b \leq 1 - \gamma\}$$

The update step of our algorithm is thus based on two projections:

1. First, project the current matrix-threshold pair (A_τ, b_τ) on C_τ. The matrix-threshold pair thus obtained is denoted by $(A_{\hat{\tau}}, b_{\hat{\tau}})$,
2. Then project $(A_{\hat{\tau}}, b_{\hat{\tau}})$ onto C_a to get $(A_{\tau+1}, b_{\tau+1})$

These two projections, as shown in the figure 4.28, are now reviewed:

Projection onto C_τ

The set of matrix-threshold pairs having zero loss on $(x_\tau, x'_\tau, y_\tau)$ can be rewritten as:

$$C_\tau = \{(A, b) \in \mathbb{R}^{n^2+1} : y[\frac{x_\tau^t A x'_\tau}{\sqrt{x_\tau^t A x_\tau}\sqrt{x_\tau'^t A x'_\tau}} - b] \geq \gamma\}$$

The following two quantities are now introduced, which will help to define a simple projection:

$$R_{-1}(x_\tau, x'_\tau, A_\tau) = \left[\min(x_\tau^t A x_\tau,\ x_\tau'^t A x'_\tau)\right]^{-1}$$

$$R_{+1}(x_\tau, x'_\tau, A_\tau) = \left[\max(x_\tau^t A x_\tau,\ x_\tau'^t A x'_\tau)\right]^{-1}$$

R_{-1} is based on the minimum of the two normalization terms whereas R_{+1} depends on the maximum of the two normalization terms. Moreover, R_{-1} and R_{+1} can be written in a single inequality as follows:

$$R_{+1} x_\tau^t A x'_\tau \ \leq\ \frac{x_\tau^t A x'_\tau}{\sqrt{x_\tau^t A x_\tau}\sqrt{x_\tau'^t A x'_\tau}} \ \leq\ R_{-1} x_\tau^t A x'_\tau$$

By subtracting b from all terms and multiplying by y_τ, the above inequality becomes:

$$y_\tau(R_{+1} x_\tau^t A x'_\tau - b) \ \leq\ y_\tau\left(\frac{x_\tau^t A x'_\tau}{\sqrt{x_\tau^t A x_\tau}\sqrt{x_\tau'^t A x'_\tau}} - b\right) \ \leq\ y_\tau(R_{-1} x_\tau^t A x'_\tau - b)$$

which can be rewritten as:

$$y_\tau R_{+1} x_\tau^t A x'_\tau - y_\tau b \ \leq\ y_\tau \frac{x_\tau^t A x'_\tau}{\sqrt{x_\tau^t A x_\tau}\sqrt{x_\tau'^t A x'_\tau}} - y_\tau b \ \leq\ y_\tau R_{-1} x_\tau^t A x'_\tau - y_\tau b$$

Hence, matrix-threshold pairs (A, b) such that:

$$y_\tau R_{y_\tau} x_\tau^t A x'_\tau - y_\tau b \geq \gamma \tag{4.4}$$

will have zero loss on the example $(x_\tau, x'_\tau, y_\tau)$ where $y_\tau = \pm 1$ and represents either similar examples ($y_\tau = 1$) or dissimilar ones ($y_\tau = -1$). Using the inequality 4.4, two subsets of C_τ could be defined, on which the current matrix-threshold pair can be projected according to the value of y_τ:

$$C'^{+}_\tau = \{(A, b) \in \mathbb{R}^{n^2+1} : R_{+1} x_\tau^t A x'_\tau - b \geq \gamma\} \qquad \text{if } y_\tau = 1$$

$$C'^{-}_\tau = \{(A, b) \in \mathbb{R}^{n^2+1} : -R_{-1} x_\tau^t A x'_\tau + b \geq \gamma\} \quad \text{if } y_\tau = -1$$

which can be conveniently rewritten:

$$C'^{y}_\tau = \{(A, b) \in \mathbb{R}^{n^2+1} : y_\tau R_{y_\tau} x_\tau^t A x'_\tau - y_\tau b \ \geq\ \gamma\},\ y_\tau \in \{-1, +1\}$$

An orthogonal projection is a projection of a figure on a line, plane etc. in such a way that the line joining the corresponding elements is perpendicular to the line, plane etc. The orthogonal projection of (A_τ, b_τ) (the current matrix-threshold pair) on $C'^{y_\tau}_\tau$, i.e. the closest element from (A_τ, b_τ) in $C'^{y_\tau}_\tau$, takes the form:

$$\begin{cases} A_{\hat{\tau}} &= A_\tau + y_\tau a(x_\tau x_\tau'^t), \text{ with } a \in \mathbb{R} \\ b_{\hat{\tau}} &= b_\tau + y_\tau a \end{cases}$$

where

$$a = \frac{\gamma - y_\tau R_{y_\tau} x_\tau^t A_\tau x'_\tau + y_\tau b}{R_{y_\tau}(||x_\tau||^2 ||x'_\tau||^2)}$$

Projection onto C_a

In order to describe the projection onto C_a, it is important to note that $A_{\tau+1}$ is the projection of $A_{\hat{\tau}}$ onto the set of all positive, semi-definite matrices, and $b_{\tau+1}$ the one of $b_{\hat{\tau}}$ onto the set $b \in \mathbb{R} : -1 + \gamma \leq b \leq 1 - \gamma$.

In order to project $A_{\hat{\tau}}$ onto the set of all positive, semi-definite matrices, the following decomposition is used: $A_{\hat{\tau}} = \sum_j \lambda_j u_j u_j^T$, where λ_j and u_j are the eigenvalues and the eigenvectors of the matrix $A_{\hat{\tau}}$ respectively. The matrix $A_{\tau+1}$ is the projection of $A_{\hat{\tau}}$ onto the set of PSD matrices (see for example [44]). Knowing the eigenvalues and eigenvectors of $A_{\hat{\tau}}$, $A_{\tau+1}$ can be written in the following form:

$$A_{\tau+1} = \sum_{j,\lambda_j>0} \lambda_j u_j u_j^T$$

If the matrix $A_{\hat{\tau}}$ is already symmetric, symmetric Householder reduction is used to convert it into a tridiagonal matrix followed by QR transformation. On the contrary, the similarity matrix is converted to the Hessenberg form before converting to real Schur form. These forms make it easier to find the eigenvalues and the eigenvectors. Template Numerical Toolkit *TNT* [15] was used to find the eigenvalues and eigenvectors for the projections. Alternatively, Lanczos method (see [44]) could be used along with symmetric tridiagonal QR algorithm or bisection method to find the eigenvalues and the eigenvectors of $A_{\hat{\tau}}$.

Algorithm

Here, an online algorithm to learn generalized cosine similarities is presented. This algorithm learn similarities of the form given in the equation 4.2 based on positive, semi-definite matrices. This algorithm is denoted as **gCosLA** for generalized Cosine similarity Learning Algorithm. The update rule consists of projecting the matrix A onto the set of positive, semi-definite matrices. For each example (in the form of a pair), the loss is calculated based on the similarity s_A. The update is performed only in case the loss is greater than zero for an example under consideration.

gCosLA - Training

Input: training set of the form (x, x', y), of n vectors in $\mathbb{R}^p$, number of epochs M; b represents the threshold

Output: list of $(p \times p)$ matrices $((A_1, b_1), \cdots, (A_q, b_q))$

Initialization $t = 1, A^{(1)} = I$ (identity matrix), $b = 0$, $\gamma > 0$

Repeat M **times (epochs)**

for $i = 1, \cdots, n$

get triplet $(x_\tau, x'_\tau, \pm 1) \in R^n \times R^n$

$l_\tau(A, b) = \max\{0, y(b_\tau - s_A(x_\tau, x'_\tau)) + \gamma\}$

if $(l_\tau(A, b) > 0)$

$R_{+1}(x_\tau, x'_\tau, A) = \left[\max\left((x_\tau^t A x_\tau), (x_\tau'^t A x'_\tau)\right)\right]^{-1}$

[15] Can be obtained from http://math.nist.gov/tnt/index.html

$R_{-1}(x_\tau, x'_\tau, A) = \left[\min\left((x_\tau^t A x_\tau), (x_\tau'^t A x'_\tau)\right)\right]^{-1}$

$a = \dfrac{\gamma - y_\tau R_{y_\tau}(x_\tau^t A_\tau x'_\tau) + y_\tau b}{R_{y_\tau}(||x_\tau||^2 ||x'_\tau||^2)}$

$A_{\hat{\tau}} = A_\tau + y_\tau a (x_\tau x_\tau'^t)$

$A_{\tau+1} = \sum_{j,\lambda_j>0} \lambda_j u_j u_j^T$ (where λ_j and u_j are the eigenvalues and eigenvectors of matrix $A_{\hat{\tau}}$)

$b_{\hat{\tau}} = b_\tau + y_\tau a$

if $(b_{\hat{\tau}} > 0)$

$\quad b_{\tau+1} = \min(b_{\hat{\tau}}, 1-\gamma)$

else

$\quad b_{\tau+1} = \max(b_{\hat{\tau}}, -1+\gamma)$

To calculate the worst-time complexity of *gCosLA*, the complexity of the different steps of the algorithm is considered. The worst-time complexity for calculating the similarity between two examples is $O(p^2)$ where p represents the number of dimensions. Similarly the first projection onto the set of zero-loss matrices costs the same i.e. $O(p^2)$. However eigen-value decomposition, being a costly operation, has the worst-time complexity as $O(p^3)$. With all this, the overall worst-time complexity for *gCosLA* can be written as $O(M.n.p^3)$ where M represent the number of iterations, n is the number of train examples while p stands for the number of dimensions or attributes.

The algorithm presented earlier assumes that the data is completely separable which is rarely true in actual practice. Here the data is considered to be inseparable. In this case the loss becomes non-zero, which can be dealt with by introducing a new parameter γ_1 which is used to decrease the previously introduced margin γ (this affects only the projection onto C_τ, the projection onto C_a being left unchanged). The set C_τ thus becomes:

$$C_\tau = \{(A,b) \in \mathbb{R}^{n^2+1} : y[\frac{x_\tau^t A x'_\tau}{\sqrt{x_\tau^t A x_\tau}\sqrt{x_\tau'^t A x'_\tau}} - b] \geq \gamma - \gamma_1\}$$

Setting $\beta = \gamma - \gamma_1$ leads to:

$$C_\tau'^{y_\tau} = \{(A,b) \in \mathbb{R}^{n^2+1} : y_\tau R_{y_\tau}(x_\tau^t A x'_\tau) + y_\tau b \geq \beta\},\ y_\tau \in \{-1,+1\}$$

This finally yields the modified value for a:

$$a = \frac{\beta - y_\tau R_{y_\tau}(x_\tau^t A_\tau x'_\tau) + y_\tau b}{R_{y_\tau}(||x_\tau||^2 ||x'_\tau||^2)}$$

However, the rest of the algorithm remains the same.

4.5.3 Online to Batch Conversion

The online algorithm, *gCosLA* is used for learning a set of similarity matrices during the training phase. In order to use the similarity matrices learned during the prediction, *gCosLA* can be

easily converted to a batch algorithm using the approach previously adopted for *SiLA*. However, instead of using weighted matrices as in *SiLA*, just the averaged sum is taken over the different similarity matrices learned during training.

gCosLA - Prediction
Input: new example x in $\mathbb{R}^p$, list of $(p \times p)$ matrices $(A_1, \cdots, A_n)$; where A is defined as:

$$A = \frac{\sum_{l=1}^{n} A_l}{n}$$

Output: list of classes

Furthermore, following the approach defined by Dekel et al. [30] and used in *SiLA*, the matrices learned during the first few iterations of the algorithm can be discarded since the algorithm is supposed to make more mistakes in the beginning as compared to the end. In other words, in a sequence of n similarity matrices learned $(A_1, \cdots, A_n)$, only the last q matrices could be taken into account for classification. The value of q can be determined using cross-validation.

4.5.4 Analysis of *gCosLA*

The following theorem provides a loss bound for the algorithm *gCosLA* in the separable case. It assumes the existence of a positive, semi-definite matrix A which separates the data in a strict sense, as well as the existence of an upper bound on the scalar product between all basic instance pairs. The inseparable case is treated in exactly the same way by replacing the positive real number γ with an arbitrary real number, not necessarily positive, β.

Theorem 4. *Let $(x_1, x'_1, y_1), \cdots (x_\tau, x'_\tau, y_\tau), \cdots, (x_N, x'_N, y_N)$ be a sequence of N examples. For any positive, semi-definite matrix A, let for each τ, $1 \leq \tau \leq N$:*

$$R_{-1}(x_\tau, x'_\tau, A) = \left[min\left((x_\tau^t A x_\tau), (x_\tau'^t A x'_\tau)\right)\right]^{-1}$$

and

$$R_{+1}(x_\tau, x'_\tau, A) = \left[max\left((x_\tau^t A x_\tau), (x_\tau'^t A x'_\tau)\right)\right]^{-1}$$

Assume that there exists a positive, semi-definite matrix A^*, a threshold b^* and a positive real number γ such that:

$$(R_{+1} x_\tau^t A^* x'_\tau - b^*) \geq \gamma \ \wedge \ (b^* - R_{-1} x_\tau^t A^* x'_\tau) \geq \gamma$$

Using the notations introduced previously, let $R \in \mathbb{R}^+$ be an upper bound such that:

$$\frac{1}{||x_\tau x_\tau'^t||_2 + 1} R_{y_\tau}^2 ||x_\tau||_2^4 ||x'_\tau||_2^4 \ \leq \ R, \ y_\tau \in \{-1, +1\}$$

Then the following bound holds for any $M \geq 1$:

$$\sum_{\tau=1}^{M} (l_\tau(A, b))^2 \ \leq \ R\left(||A^* - I||_2^2 + (b^*)^2\right)$$

A proof of theorem 4 can be established along the same lines as the proof of the loss bound provided for the *POLA* algorithm in [99] and is presented in the Appendix A. The only requirement in *POLA* is that the data should lie in a sphere of radius R. This requirement is translated in the case of a generalized cosine similarity by the fact that the scalar product between data points, normalized by its maximum or minimum values, is bounded. Introducing the maximum and minimum values leads to a stricter notion of separation. It however allows one to rely on simple projections.

As the inseparable case can be treated in exactly the same way, by directly replacing the positive scalar γ by β, a scalar not necessarily positive, one can see that the condition imposed is not really restrictive, and leads to an algorithm with an explicit bound on the loss function. Furthermore, the theorem for the inseparable case (as well as its proof) is the same as the one for the separable case, β being used instead of γ.

4.6 Comparison of *SiLA* and *gCosLA* with other state of the art algorithms

SiLA and *gCosLA* are supervised online algorithms having an effective online to batch conversion mechanisms like *POLA* [99]. These three algorithms update the similarity or distance matrix only if loss > 0 and a misclassification has been made. *SiLA* works with individual examples whereas *gCosLA* and *POLA* operate on pairs of similarly and differently labeled examples. Furthermore, loss bounds on the performance have been provided for all of the three algorithms. These bounds guarantee a generalization well beyond the training examples.

SiLA as well as *gCosLA* could be considered as a *global* similarity learning algorithms since only global similarity matrices are learned for subsequent classification of test data. Moreover, the similarity matrices are not class dependent. Stahl et al. [100], on the other hand, learn local similarity measures.

Although *SiLA* is based on the voted perceptron proposed in Freund and Schapire [37] and used in Collins' algorithm [20], yet it differs substantially from these two algorithms. The aim here, is to learn similarity in *kNN* classification, whereas it was used for binary classification with a separating hyperplane in Freund and Schapire and for the discriminative training of hidden Markov models in Collins's work.

SiLA and *gCosLA* use *kNN* classification algorithm like *LMNN* [112] and *MCML* [41]. The basic aim in *SiLA* coincide with that of *LMNN*: bringing *target* neighbors closer while pushing apart the *impostors*. Both of these methods can be used for binary or multiway classification. While comparing *SiLA* with *MCML*, one can see that in the later method, the *target* neighbors are collapsed to a single point and the *impostors* are pushed infinitely apart.

SiLA does not require the similarity matrix to be positive, semi-definite (PSD) like *ITML* [28] and *OASIS* [16], and unlike *gCosLA*, *POLA* [99] and the approaches of Xing et al. [114], Globerson et al. [41] and Weinberger et al. [112]. The inclusion of PSD constraints require additional computation time. Although *gCosLA* works with bi-linear form defined by PSD matrices, yet it learns a similarity metric rather than a distance one as in other metric learning approaches.

Furthermore, no eigenvalue decomposition of the similarity matrix is required for *SiLA* just like *ITML*. An important point regarding distances is that they are related to the trace of a

matrix. On the other hand, there is no relation between similarity and the trace.

Comparing *SiLA* and *gCosLA* with Xing's algorithm reveals that Xing's algorithm is used for clustering and is batch in essence. Furthermore, it does not have a computationally effective online version and theoretical error guarantees regarding unseen examples. However, *SiLA* and *gCosLA* are used for classification purposes, are effective online algorithms and have got theoretical error guarantees. This makes sure that they make just a *limited* number of mistakes on unseen examples.

Grabowski and Szalas [46] also learn a similarity measure which is an asymmetric variant of the Jaccard coefficient, and is a special case of the similarity functions considered in the case of *SiLA*. However, their goal is more along the lines of feature selection than similarity learning.

In comparison with Hust's work [52] on Collaborative Information Retrieval, where a variant of cosine similarity is learned based on a diagonal matrix only; *SiLA* allows to learn diagonal and square matrices.

The neural network approach (*SNN*) of Melacci et al. [72], to learn similarity differs from *SiLA* owing to an always positive value of similarity. The reason is the use of sigmoidal function. *SiLA*'s similarities, on the other hand, are not necessarily positive. Another difference is that the similarity is always symmetric for *SNN* like *gCosLA*.

The aim in *SiLA* is to directly reduce the $0-1$ loss or the leave-one-out error like *NCA* - Neighborhood Components Analysis [42]. *SiLA* is a classification algorithm and requires complete supervision in the form of class labels. However, *OASIS* does not require the class labels as it learns a pairwise (dis)similarity measure. Both *SiLA* as well as *OASIS* do not require the similarity or distance matrix to be symmetric in nature. As discussed earlier, *SiLA* updates the similarity matrix only if the algorithm has made an error. On the other hand, *OASIS* is based on systematic updates.

For *gCosLA*, the initial similarity matrix is initialized with an identity matrix like *OASIS*. This means that *gCosLA* resembles the standard cosine whereas *OASIS* behaves like the Euclidean distance during the first iteration. The method of converting a similarity matrix into a PSD one resembles to the one followed by *POLA* and *MCML*. In this method, the (dis)similarity matrix A is projected onto the set of PSD matrices by taking the eigenvalue decomposition of A followed by the removal of negative eigenvalues.

Peterson et al. [84] use genetic algorithm to optimize *kNN* performance using cosine similarity, Pearson correlation and Euclidean distance. However, in this case, no metric is learned unlike *SiLA*, *gCosLA* and other metric learning algorithms.

The complexity of *gCosLA* algorithm (Mnp^3) is higher than that of *SiLA* (Mnp^2) because of the use of eigenvalue decomposition. Furthermore, the cosine similarity measure used in *SiLA* cannot be called a generalized cosine one, since the normalization is completely independent of the similarity matrix learned. Another difference between *SiLA* and *gCosLA* lies in the fact that *gCosLA* works with pairs of examples like *POLA* which can be similar or dissimilar, while *SiLA* works with individual examples.

gCosLA can be considered as belonging to the family of *passive aggressive* algorithms described in Crammer et al.[23]. It is *passive* when the current similarity matrix correctly classifies the current example, in which case the current matrix is left unchanged. On the contrary, if there is some *loss* for the current example, it *aggressively* forces the update to have zero loss for

the current example.

4.7 Conclusion

Several works have proved that cosine similarity, which is mainly used while dealing with texts, should be preferred over the Euclidean distance on several, non-textual datasets as well. This explains the importance of learning appropriate similarity measures apart from the distances for *kNN* classification.

SiLA (Similarity Learning Algorithm) is based on learning globally a similarity metric with the help of training examples. It is based on voted perceptron developed by Freund and Schapire [37] and used by Collins [20]. The aim is to move the *target* neighbors (examples belonging to the same class as that of the input example) closer while pushing apart the *impostors* (examples from other classes). It directly reduces the leave-one-out error or the $0-1$ loss by reducing the number of mistakes on unseen examples. The similarity matrices learned during the training phase can be used for prediction. The similarity used in the case of *SiLA* does not guarantees that a symmetric bi-linear form exists. Nevertheless, the similarity matrix can be projected onto the set of positive, semi-definite (PSD) matrices thus giving rise to *eSiLA*.

RELIEF is a well known feature re-weighting algorithm. It has been recently shown that *RELIEF* could in fact be seen as a distance learning algorithm in which a linear utility function with maximum margin is optimized. A version of *RELIEF* for similarity learning called *RELIEF*-Based Similarity (*RBS*) is proposed. As *RELIEF* and unlike *SiLA*, *RBS* does not try to optimize the leave-one-out error, and does not perform very well in practice. This is illustrated on many UCI collections. Therefore, a stricter version of *RBS*, called *sRBS* is developed which aims at relying on a cost function closer to the $0-1$ loss. The results for *sRBS* show that it is a much better idea of use 0-1 loss rather than its approximation. All of the *RELIEF* based algorithms were extended to work with PSD matrices.

The normalization in *SiLA* is completely independent of the learned similarity matrix which hinders in defining a truly generalized cosine similarity. The approach previously used in *SiLA* cannot be used to define a generalized cosine similarity. Since generalized cosine similarities are based on scalar products, they involve bi-linear forms defined by positive, semi-definite (PSD) matrices. However, the normalization (dependent on the similarity matrix) introduced in the cosine similarity prevents one from directly re-using the algorithms previously introduced for learning say Mahalanobis distances, also based on PSD matrices. This motivates to learn a *generalized* cosine similarity - *gCosLA*, where the similarity matrix is positive, semi-definite (PSD) and the normalization is dependent on the similarity matrix. In order to convert a matrix into its PSD equivalent, it is projected onto the set of PSD matrices inspired from the approach adopted in *POLA* (Shalev et al. [99]). Since *POLA* is based on learning the pairwise constraints i.e. equivalence and inequivalence in order to learn a *global* distance metric, *gCosLA* learns the similarity metric based on the pairwise constraints.

Chapter 5

Experiments and Results

Contents

5.1 Introduction

In order to assess the performance of a learning algorithm, it must be tested over different datasets. The datasets must be different from one another and should be able to validate an algorithm. Furthermore, the datasets should be diverse i.e. they should have different number of classes, features and examples etc. Generally a dataset is divided into three distinct parts (which means that there should not be any overlapping): training set, validation set and test set.

Training set is used exclusively for learning the different parameters of the algorithm. In order to verify whether an effective training has been performed or not, a validation set is formed from the data set which must not contain any of the training examples and is used to fine tune an algorithm. Test set is required to verify the performance of the algorithm on unseen examples. Normally 80% of the instances are used for training and validation sets whereas the rest of the examples (20%) are used for the test data. Furthermore 80% examples are retained in the training set while 20% account for the validation set.

This chapter explains the experiments conducted with different similarity learning algorithms over various datasets. Cosine similarity is compared with the Euclidean distance. This is followed by a detailed comparison between cosine, *SiLA* and *gCosLA* while using *kNN* as well as *SkNN*. All of the algorithms belonging to the *RELIEF* family are also thoroughly tested and compared with the standard *kNN* and *SkNN* rules. *SiLA* and *gCosLA* are compared with different state of the art algorithms in the field of metric learning. Similarly *kNN* is compared with its symmetric version *SkNN* while using the cosine similarity.

The next section describes the various datasets used for the experimental validation of the different algorithms.

5.2 Description of the datasets used

Many different datasets were used in order to assess the performance of the various similarity learning algorithms. All of the datasets except *Newsgroups* are part of the UCI database [36]), namely, *Ionosphere*, *Iris*, *Wine*, *Balance*, *Soybean (Small)*, *Glass Identification*, *Pima Indians Diabetes*, *BUPA Liver Disorders*, *Letter Recognition*, *(Statlog) German Credit Data*, *(Statlog) Heart*, *Yeast*, *Magic*, *Spambase*, *Magic*, *Sonar*, *Segmentation*, *Optdigits* and *Waveform*. These are standard collections which have been used by different research communities (machine learning, pattern recognition, statistics etc.). The details about the datasets are next presented as shown in Table 5.1, 5.2 and 5.3:

1. The *Iris Plant* data set contains 3 classes, each has 50 instances where each class refers a type of Iris plant. Two of the three classes are not linearly separable from each other. The number of attributes is 4. 120 examples were used for training (96 for learning and 24 for validation), and 30 for testing.

2. The *Wine Recognition* data set contains 13 attributes representing the constituents found in each of the three different types of wines. 143 examples were used for training (114 for learning and 29 for validation) while 35 for testing purposes.

	Iris	**Wine**	**Balance**	**Ionosphere**	**Glass**	**Soybean**	**Pima**	**Liver**
Learn	96	114	400	221	137	30	492	220
Valid.	24	29	100	56	35	8	123	56
Test	30	35	125	70	42	9	153	69
Class	3	3	3	2	6	4	2	2
Feat.	4	13	4	34	9	35	8	6

Table 5.1: Characteristics of datasets - I

	Letter	**German**	**Yeast**	**Heart**	**Magic**	**Spambase**	**Musk-1**	**News**
Learn	12800	640	950	172	12172	2944	304	1824
Valid.	3200	160	238	44	3044	737	77	457
Test	4000	200	296	54	3804	920	95	2280
Class	26	2	10	2	2	2	2	20
Feat.	16	20	8	13	10	57	168	200 [16]

Table 5.2: Characteristics of datasets - II

	Sonar	**Segmentation**	**Optdigits**	**Waveform**
Learn	133	134	2447	3200
Valid.	34	34	612	800
Test	41	42	764	1000
Class	2	7	10	3
Feat.	60	19	64	21

Table 5.3: Characteristics of datasets - III

3. The *Balance Scale* data set contains 3 classes along with 4 attributes. 500 examples were used for training and 125 for test. Among the training examples, 400 were chosen for learning while 100 were used for validation.

4. *Ionosphere* is a binary classification data set where the aim is to classify radar returns from the ionosphere. 281 examples were considered for training (80% or 221 for learning and the rest, 56 for validation) whereas 70 for test along with 34 features.

5. The *Glass Identification* dataset contains 6 types of glasses based on different oxide content. The motivation for this dataset is that the glass left at the scene of the crime can be used as evidence afterward. This dataset has 9 features (the first one is just the identification number and has been omitted). 172 examples were used for training (137 for learning while 35 for validation) and 42 for testing.

6. *Soybean (Small)* is a subset of the original soybean dataset. It contain 35 features. 38 examples were used for training purpose while 9 for testing. Among the training examples, 30 were chosen for learning purpose while 8 for validation. The number of classes is 4.

7. *Pima Indians Diabetes* dataset, also known as *Pima* dataset, is also a binary classification problem and consists of data from diabetes patients from Pima Indian heritage. The aim is to identify the patients who test positive for diabetes. 615 examples were used for training purpose (492 for learning and 123 for validation) while 153 for testing.

8. *BUPA Liver Disorders* dataset, sometimes referred as *Liver* dataset, is also a medical dataset where 276 examples were considered for training (220 for learning and 56 for validation) and 69 for testing. The task is to identify the presence of a liver disorder, based on 6 attributes where the first 5 refer to blood tests considered sensitive to liver disorders which can develop from excessive alcohol consumption.

9. The aim in *Letter Recognition* data set is to recognize the English language capital letters out of 26 possibilities (A-Z). The images of the letters are based on 20 fonts which makes 20000 examples in total. The attributes are composed of statistical moments and edge counts. 12800 examples were used for learning, 4800 for validation and 4000 for testing.

10. *(Statlog) German Credit* data set contains 800 examples for training (640 for learning whereas 160 for validation) while 200 account for the test set. The aim is to classify a customer has good or bad credit risk.

11. The target in *Yeast* dataset is to find the localization site of protein. It is composed of 1188 examples for training (950 for learning and 238 for validation) and 296 for testing. The number of features is 8.

12. *(Statlog) Heart* is a heart disease data set consisting of 216 training examples (172 for learning while 44 for validation) and 54 test ones. The aim is to detect the presence or absence of heart disease in patients using 13 features.

13. *Magic* dataset is a binary dataset having only two classes and 10 features. It was generated by Monte Carlo method to simulate registration of high energy gamma particles in an

atmospheric Cherenkov telescope. It is made up of 19020 examples of which 12172 make up the training set, 3044 account for the validation set and 3804 are placed in the test set.

14. *Spambase* is also a binary classification dataset having a simple aim: classify an email as spam or otherwise. It has 2944 instances as training, 737 for validation and 920 for testing purposes. The number of attributes is 57.

15. In *Musk-1* dataset, an algorithm has to predict whether new molecules will be musks or non-musks. It contains 304 training examples, 77 validation ones while 95 instances are used for testing purpose. The 166 features depend upon the exact shape or conformation of the molecule.

16. The 20-newsgroups data set is composed of posted articles from 20 newsgroups and approximately contains 20,000 documents. The 18828 version was used in which the cross-postings have been removed and includes only the "From" and "Subject" headers. the *Rainbow* package [71] was used to tokenize the data set where each document was formed of the weighted word-counts of the 20,000 most common words. This was followed by performing singular value decomposition using *SVDlibc* [17] which reduced the original 20,000 dimensions to 200. Many of the resulting documents did not contain any of the 200 selected words. The empty documents containing none of the 200 words were subsequently removed reducing the number of documents to 4561. Out of 4561 documents, 2281 documents were used for training and validation, while 2280 documents were used in the testing phase.

17. The aim in *Sonar* dataset is to separate the sonar signals bounced off a metal cylinder (mine) and those bounced off a roughly cylindrical rock. There are 111 signals which were bounced off a metal cylinder at various angles and under various conditions. Similarly 97 patterns were obtained from rocks under similar conditions. Each pattern is a set of 60 numbers (features) in the range of 0.0 to 1.0. Out of 208 signals, 133 are used for training while 34 for validation. Finally 41 signals are used for testing.

18. *(Statlog) Image Segmentation* is an image dataset consisting of randomly drawn images from a database of 7 outdoor images. The images are further hand-segmented to create a classification for every pixel. Here, only the training set containing 210 images is used for classification purposes. 134 images were used for training, 34 for validation whereas 42 were used for testing purposes. Each image is consisted of 19 features.

19. An optical recognition dataset, called *Optdigits* is also used to evaluate different algorithms. The aim in this dataset is the optical recognition of handwritten characters (0-9). Only the training set containing 3823 instances is used. Furthermore, 2447 instances are used for training while 612 are retained for validation. Similarly 764 instances compose the test set. The number of features is 64.

20. Another UCI dataset used for validating different algorithms is the *Waveform* database generator (Version 1) dataset. This dataset contains 3 classes of waves equally distributed among 5000 instances. There are 21 features in total, all of which include noise. 3200 instances were used for training, 800 for validation and 1000 for testing.

[17] Can be obtained from http://tedlab.mit.edu/ dr/svdlibc/

- Random split the sample S in V sub-samples $S^{(1)}, \ldots, S^{(V)}$
- For $v = 1, \ldots, V$, do :
  - ★ Merge $V-1$ subsamples into $S^{(-v)} = S - S^{(v)}$
  - ★ For each candidate model $m \in \mathcal{M}$, compute the discrimination rule $\theta_m^{(-v)}$ and select the best model regarding the cross-validated error-rate :

$$m_v^\star = \underset{m \in \mathcal{M}}{\operatorname{argmin}}\ CV_m^{(-v)}$$

  - ★ Evaluate the error rate t_v of $m_v^\star$ on the test sample $S^{(v)}$
- Average the V error rates $t_1, \ldots, t_V$

Figure 5.1: Double cross validation [35] algorithm

5.3 Methodology used for the experiments

This section describes how the datasets were used for different similarity learning algorithms i.e. *SiLA*, *eSiLA*, *RBS*, *sRBS*, *RBS-PSD*, *sRBS-PSD*, *gCosLA*. 20 percent of the data was used for testing purpose for each of the dataset. Of the remaining data, 80 percent was used for training whereas 20 percent for the validation sets for all of the algorithms. 5-fold double cross-validation [35] was used to learn the matrix sequence $(A_1, A_2, \cdots, A_q)$ for all of the datasets. The double cross-validation algorithm is shown in figure 5.1. In the technique of double cross-validation, the dataset is splitted into V sub-samples or folds (in this case 5). One sample is selected as a *test* sample. The remaining samples, composed of training and validation examples, are considered as the samples used for learning purposes. Based on this distribution, the algorithm is run multiple times with different parameter values (e.g. different value of k nearest neighbors) thus giving a set of accuracies over the *test* sample. This helps to determine the best model having the best parameter values for the current fold, based on the largest accuracy value.

This is followed by considering another sample as a test one (different from the first one) taken from the V samples. Moreover, the rest of the samples are considered as learning samples. Different parameter values are tested just like the first fold so as to determine the best one. In the end, the V accuracies are averaged to find the global accuracy.

In the case of *kNN*-cos, *SkNN*-cos and *kNN*-euclidean only the best value of k was determined using the method of double cross-validation. The best value of k was chosen from the possible values of 1 and 3.

It may be further recalled that in a sequence of hypothesis, the last q elements may be more interesting than the earlier ones. Based on this fact, the validation set was used for *SiLA* to determine the value of k (nearest neighbors), optimum number of epochs and the best value of q.

However, in the case of *gCosLA*, the validation set was used to determine the aforementioned parameters learned for *SiLA* as well as the best value of the threshold β. It was observed that for each dataset, the best value of β is usually different for each class and each fold.

In order to create pairs of examples for *gCosLA*, 5 nearest neighbors were found for each of the example from the class it belongs. Additionally, the same number of nearest neighbors from

different classes was also found. Thus the total number of pairs of examples for each dataset became $10N$ where N represents the number of examples in a dataset.

For *RELIEF* and *RBS*, a single weight vector was learned whereas for *sRBS*, a sequence of matrices $(A_1, A_2, \cdots, A_q)$ is learned. Double cross-validation is used to find the best value of k for *RELIEF* and *RBS* algorithms. On the other hand, for *sRBS*, the values of k, λ and β are determined. The approaches followed in the case of methods involving PSD matrices i.e. *eSiLA*, *RELIEF-PSD*, *RBS-PSD* and *sRBS-PSD* are the same as the ones used for their counterparts without PSD matrices.

5.3.1 Prediction Rules

Two prediction rules were used for all of the experiments. The first one is the standard *kNN* rule where the classification is based on the k nearest neighbors while the second one is *SkNN* ('S' means symmetric), which is based on the difference of similarity between k nearest neighbors from the same class and k from other classes[18]. Combined with the similarity learning algorithms, these prediction rules provide four different possibilities for comparison:

1. Standard kNN rule with the cosine similarity by replacing A matrix with the Identity matrix. This rule is referred to as *kNN-cos*,
2. Standard kNN rule with the similarity learned with the similarity learning algorithms. This method is termed as *kNN-A*,
3. The symmetric prediction rule with the cosine similarity having $A = I$, which is called *SkNN-cos*,
4. The symmetric prediction rule with the similarity learned with the similarity learning algorithms. This method appears as *SkNN-A*.

Unless otherwise stated, a binary version of the algorithms was used, in which a sequence of matrices is learned for each class (*one vs others*), and the quality of a given method was assessed with its average accuracy (i.e. the accuracy averaged over the different classes).

In addition, the standard deviation was computed on all of the collections for all of the algorithms. The results were evaluated for statistical significance i.e. whether one method is significantly better than the other one or not. In case the P-value is less than or equal to 0.01, this means that the difference is *much more significant* and is denoted by $\ll$ or $\gg$. A lower level of significance occurs when the P-value lies in between 0.01 and 0.05, in which case is denoted by $<$ or $>$. In case, the P-value is greater than 0.05, the results are considered equivalent and are denoted by $=$.

5.4 Cosine similarity vs Euclidean distance in kNN classification

Even though *kNN* has been traditionally used, on the collections earlier seen, with the Euclidean distance (or with a Mahalanobis distance learned from the data, as in [28, 112]), it is shown here

[18] One can find in Nock [75] a different version of a *symmetric* kNN rule in which one considers not only the k nearest neighbors of a given example x, but also the points for which x is a nearest neighbor.

	kNN-cosine	*kNN*-Euclidean
Soybean	**1.0** ± 0.0	**1.0** ± 0.0
Iris	**0.987** ± 0.025	0.973 ± 0.029
Letter	**0.997** ± 0.002	**0.997** ± 0.002
Balance	**0.954** ± 0.021 ≫	0.879 ± 0.028
Wine	**0.865** ± 0.050 ≫	0.819 ± 0.096
Ionosphere	**0.871** ± 0.019	0.854 ± 0.035
Glass	**0.899** ± 0.085	0.890 ± 0.099
Pima	0.630 ± 0.041	**0.698** ± 0.024 ≫
Liver	**0.620** ± 0.064	**0.620** ± 0.043
German	0.594 ± 0.040	**0.615** ± 0.047
Heart	**0.670** ± 0.020	0.656 ± 0.056
Yeast	0.911 ± 0.108	**0.912** ± 0.108
Spambase	**0.858** ± 0.009	0.816 ± 0.007
Musk-1	0.844 ± 0.028	**0.848** ± 0.018

Table 5.4: Comparison between cosine similarity and Euclidean distance based on s-test

that the cosine should be preferred to the Euclidean distance on many of these collections.

The utility of the cosine similarity on text data has been recognized now for many years. However, on most non-textual collections, the majority of researchers rely on the Euclidean distance. In order to assess the validity of using the cosine similarity on non-textual collections, two standard *kNN* rules are used, one with the cosine similarity, the other one with the Euclidean distance, on the UCI collections. Table 5.4 summarizes the accuracy obtained with *kNN*-cosine and *kNN*-Euclidean along with their respective standard deviations. The first column gives the average accuracy obtained while using a binary version of the cosine-based *kNN* classifier, whereas the second one corresponds to the Euclidean distance-based *kNN* classifier. The best results are represented in bold.

As one can note, the cosine similarity yields results which are either better or the same as that for Euclidean distance for most of the data sets. Even though the results are on par with the *Glass*, *Soybean*, *Liver* and *Letter* data sets, the difference is important on *Wine* (better by 4.6%), *Balance* (better by 7.5%) and *Spambase* (better by 4.2%) collections. For *Pima*, the Euclidean distance gives better result as compared with the cosine measure (gain of 6.8%). Micro sign test (s-test), earlier used by [119], was performed to assess the statistical significance of these results. It can be observed that cosine is statistically much better (shown by '≫') than Euclidean distance on *Wine* and *Balance*. However the difference between cosine and Euclidean distance is not statistically significant on *Ionosphere* and the other data sets. Similarly Euclidean distance was much better than cosine on *Pima* data set.

Figure 5.2 depicts the comparison between cosine and Euclidean distance with *kNN* algorithm. The standard deviations can also be viewed in the figure.

These results justify the use of the cosine similarity, instead of the Euclidean distance, on some of these collections e.g. *Balance* and *Wine*.

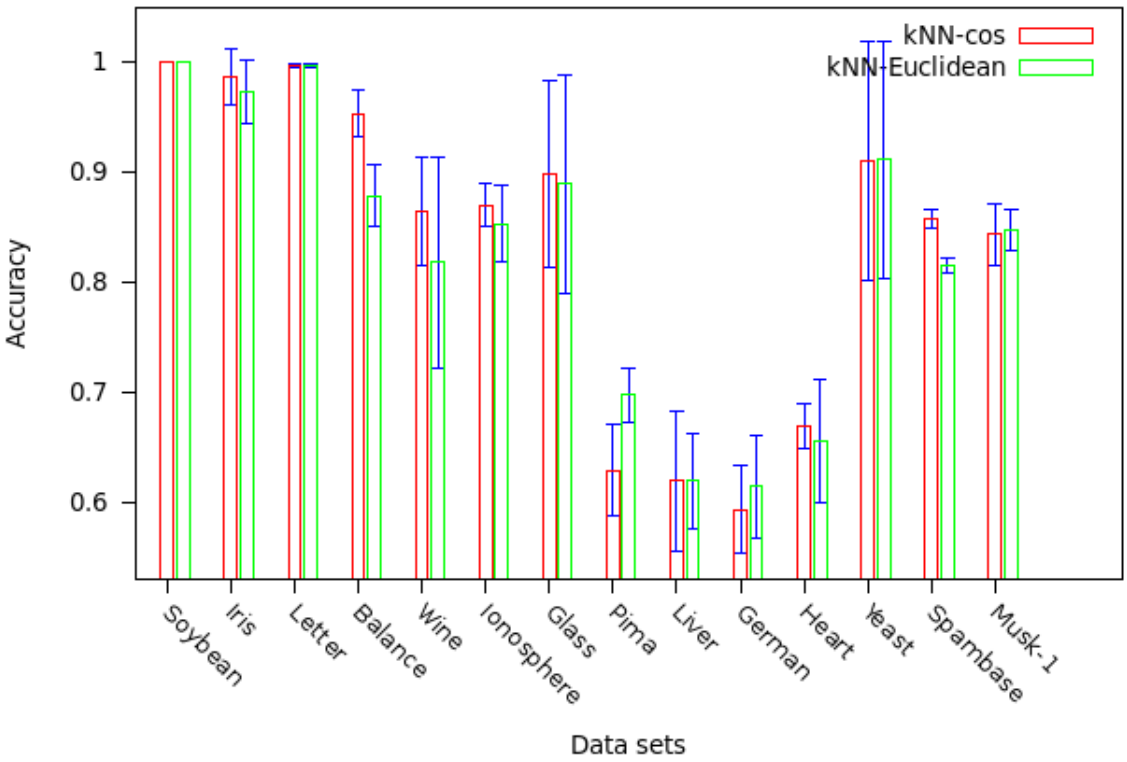

Figure 5.2: *kNN*-cos vs *kNN*-Euclidean on various datasets

5.5 Comparison between cosine, *SiLA* and *gCosLA*

In this section, cosine similarity is compared with *SiLA* and *gCosLA* on various datasets. The comparison is made both between the simple *kNN* rule as well as its symmetric version *SkNN*. Moreover, *SiLA* is also compared with *gCosLA*. This is followed by a comparison between *kNN*-A and *SkNN*-A for *gCosLA* in order to see the significance of devising a symmetric version of *kNN*. Furthermore, *kNN*-Euclidean is compared with *kNN*-A of *gCosLA* to ascertain the importance of learning a similarity metric instead of using a distance one.

5.5.1 Performance of *kNN*-cos as compared to *SiLA* and *gCosLA*

The comparison of *SiLA* and *gCosLA* algorithms with cosine while using the *kNN* prediction rule is given in Table 5.5. Figure 5.3 and 5.4 give a graphical and an easier to follow description of the comparison of *kNN*-cos with *SiLA* and *gCosLA* respectively.

It can be observed that *SiLA* performs significantly better than cosine (*kNN*-cos), as confirmed by the statistical significance test *s-test* (shown by the sign $\gg$ or $>$) for *Balance* (accuracy better by 2.5%), *Ionosphere* (better by 4.0%), *Pima* (better by 1.8%) and *German* (gain by 5.2%).

Similarly *gCosLA* performs significantly better than cosine on *Balance* (gain of 2.7% in terms of accuracy), *Wine* (gain of 5.3%), *Liver* (better by 3.8%), *German* (improvement by 14.3%) and *Heart* (gain of 6.7%). The performance of all of the methods is comparable for *Iris*, *Glass* and *Yeast*. However for *Soybean*, *kNN-cos* is significantly better than *kNN-A* for the algorithm

	kNN-cos	kNN-A (SiLA)	kNN-A (gCosLA)
Soybean	**1.0** ± 0.0	**1.0** ± 0.0	0.972 ± 0.061 (<)
Iris	**0.987** ± 0.025	0.978 ± 0.030	**0.987** ± 0.025
Letter	**0.997** ± 0.002	0.962 ± 0.003	0.995 ± 0.003
Balance	0.954 ± 0.021	0.979 ± 0.012 ≫	**0.981** ± 0.008 ≫
Wine	0.865 ± 0.050	0.884 ± 0.062	**0.918** ± 0.064 ≫
Ionosphere	0.871 ± 0.019	**0.911** ± 0.031 ≫	0.880 ± 0.039
Glass	**0.899** ± 0.085	0.892 ± 0.094	0.893 ± 0.097
Pima	0.630 ± 0.041	**0.648** ± 0.025 >	0.624 ± 0.051
Liver	0.620 ± 0.064	0.609 ± 0.040	**0.658** ± 0.070 >
German	0.594 ± 0.040	0.646 ± 0.046 ≫	**0.737** ± 0.042 ≫
Heart	0.670 ± 0.038	0.659 ± 0.020	**0.737** ± 0.062 ≫
Yeast	**0.911** ± 0.108	0.905 ± 0.114	0.909 ± 0.112

Table 5.5: Classification accuracy of cosine, *SiLA* and *gCosLA* using *kNN*

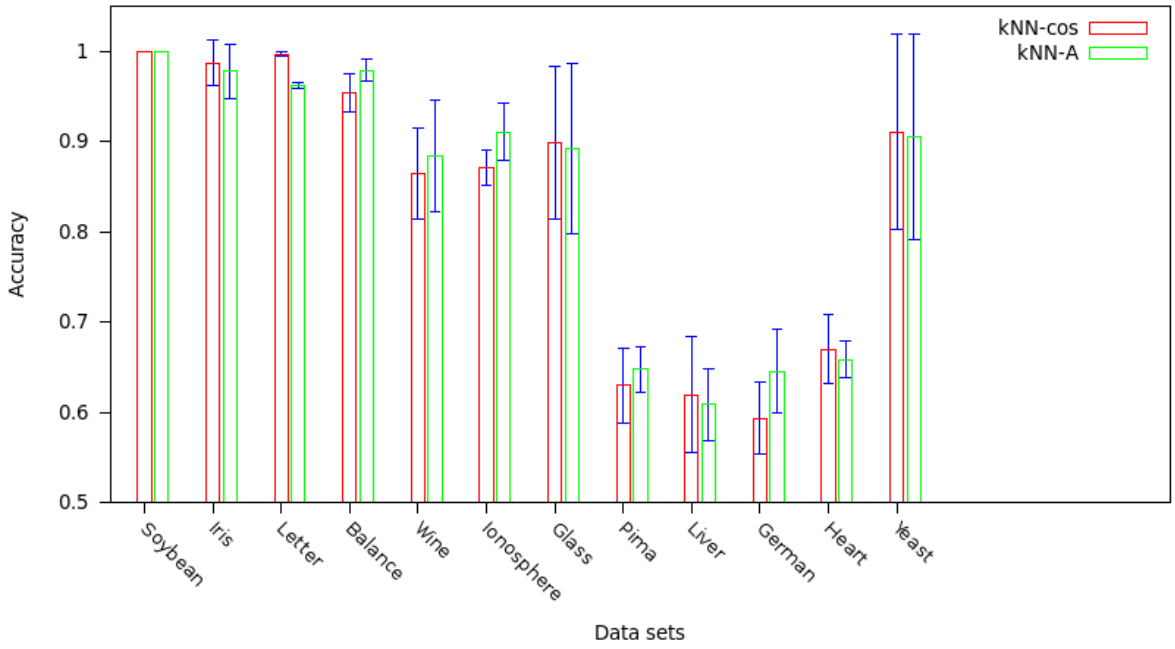

Figure 5.3: *kNN*-cos vs *kNN*-A (SiLA) on various datasets

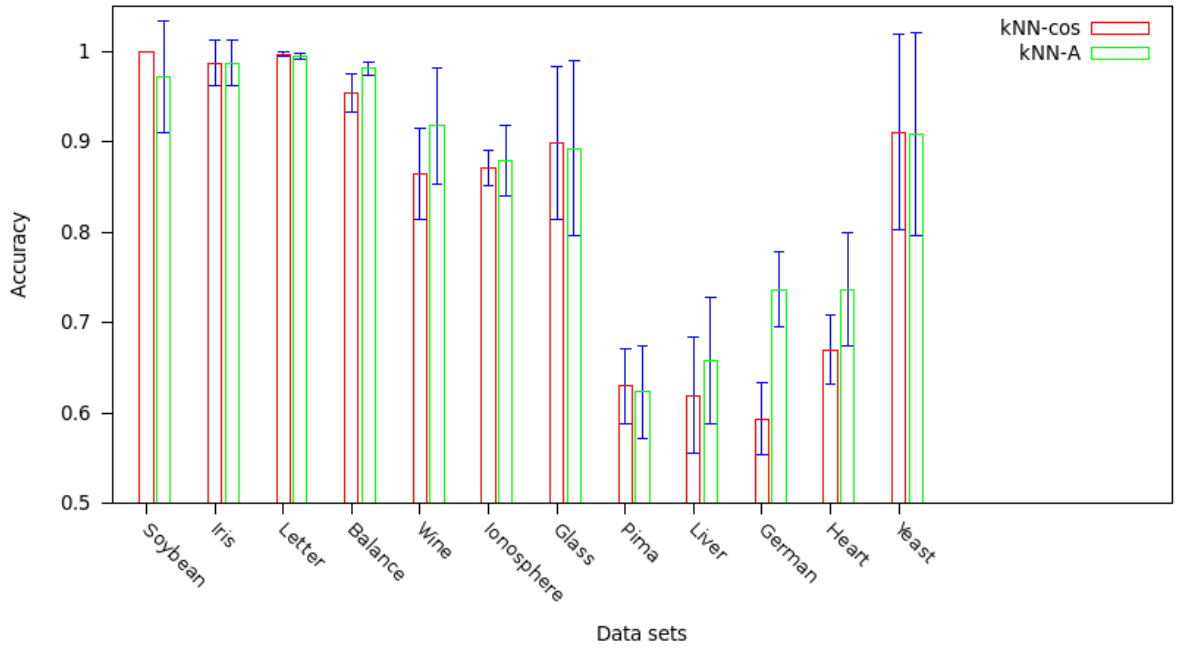

Figure 5.4: *kNN*-cos vs *kNN*-A (gCosLA) on various datasets

gCosLA. These results do not help to decide which one of *SiLA* and *gCosLA* is a significantly better similarity metric learning algorithm as compared to the standard cosine similarity. The reason is that there are many datasets on which only one of the similarity learning algorithms is significantly better than cosine i.e. *Wine*, *Ionosphere*, *Pima*, *Liver* and *Heart*.

5.5.2 Performance of *SkNN*-cos as compared to *SiLA* and *gCosLA*

The symmetric counterpart of *kNN*, i.e. *SkNN* was also used to compare cosine with *SiLA* and *gCosLA* as shown in the table 5.6. Table 5.6 also gives the statistical significance of the results for *SiLA* and *gCosLA* on the basis of *SkNN* method.

It can be observed that *SiLA* performs significantly better than cosine for *Balance* (better by 1.1%), *Wine* (gain of 2.6%), *Ionosphere* (4.6%), *Pima* (2.0%) and *German* (gain by 4.7%).

Similarly *gCosLA* performs significantly better than cosine as confirmed by the statistical significance test *s-test* (shown by the sign $\gg$ or $>$) on *Balance* (1.2%), *Wine* (gain of 5.9%) and *German* (better by 10.9%). The performance of all the methods is comparable for *Soybean*, *Iris*, *Glass* and *Liver*. It should be noted that although cosine and *SiLA* are better than *gCosLA* on *Soybean* by 2.8%, yet the improvement is not significant enough.

Moreover, *SkNN-cos* performs significantly much better than *SkNN-A* on the datasets *Heart* and *Yeast* for *SiLA*.

Figure 5.5 and 5.6 compare the performance of cosine similarity with *SiLA* and *gCosLA* respectively while using *SkNN* decision rule. The standard deviations are also depicted in these two figures.

	SkNN-cos	*SkNN*-A (SiLA)	*SkNN*-A (gCosLA)
Soybean	**0.989** ± 0.034	**0.989** ± 0.034	0.961 ± 0.075
Iris	**0.987** ± 0.025	0.980 ± 0.025	0.984 ± 0.025
Letter	**0.997** ± 0.002	0.962 ± 0.003	0.994 ± 0.003
Balance	0.969 ± 0.013	0.980 ± 0.012 ≫	**0.981** ± 0.009 ≫
Wine	0.867 ± 0.055	0.893 ± 0.062 >	**0.926** ± 0.055 ≫
Ionosphere	0.860 ± 0.024	**0.906** ± 0.035 ≫	0.883 ± 0.032
Glass	**0.898** ± 0.081	0.895 ± 0.085	0.897 ± 0.085
Pima	0.643 ± 0.030	**0.663** ± 0.028 ≫	0.643 ± 0.031
Liver	0.638 ± 0.060	0.600 ± 0.046	**0.652** ± 0.068
German	0.620 ± 0.030	0.667 ± 0.040 ≫	**0.729** ± 0.037 ≫
Heart	0.711 ± 0.036	0.674 ± 0.047 ≪	**0.717** ± 0.061
Yeast	**0.917** ± 0.103	0.910 ± 0.106 ≪	0.912 ± 0.112

Table 5.6: Classification accuracy with cosine, *SiLA* and *gCosLA* using *SkNN*

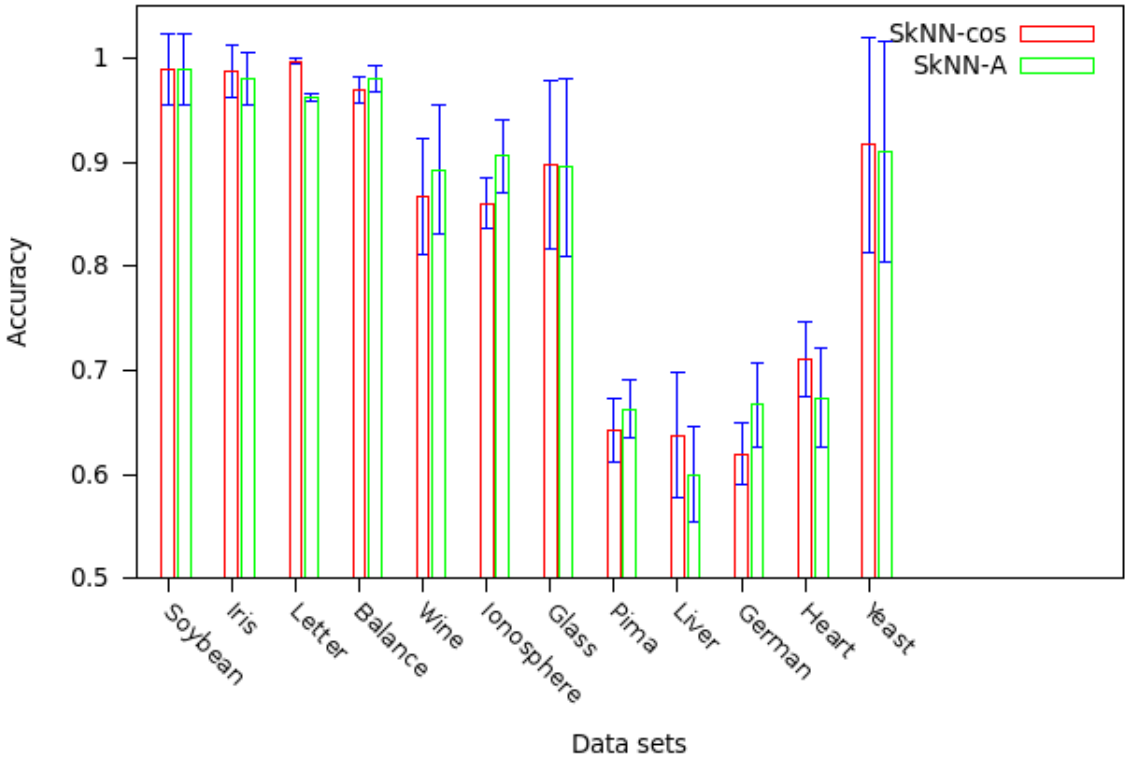

Figure 5.5: *SkNN*-cos vs *SkNN*-A (SiLA) on various datasets

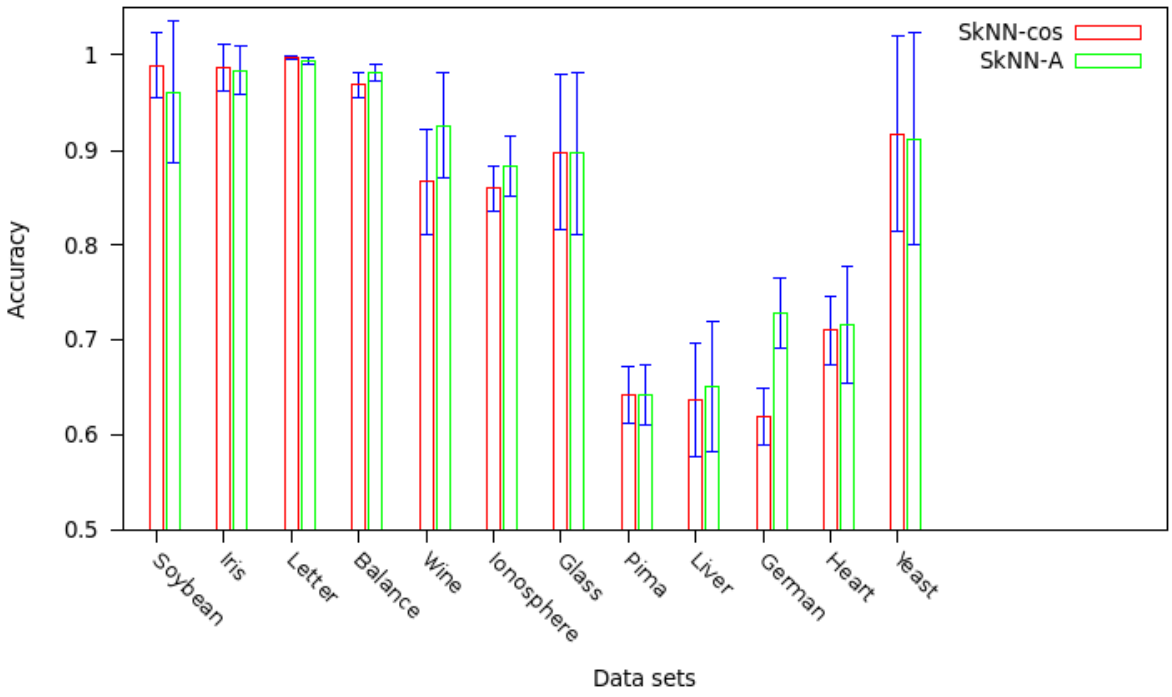

Figure 5.6: *SkNN*-cos vs *SkNN*-A (gCosLA) on various datasets

	kNN-cos	*kNN*-A (SiLA)	*SkNN*-cos	*SkNN*-A (SiLA)
News	0.929	**0.947**	0.907	0.902

Table 5.7: Comparison between cosine and *SiLA* for *News*

5.5.3 Cosine and *SiLA* on *News* dataset

The cosine similarity is also compared with *SiLA* on *News* dataset. Only 1 fold is used for this dataset due to its large size. The is the reason no standard deviation is mentioned in table 5.7. *SiLA* performs better than cosine similarity while using *kNN* rule (accuracy better by 1.8%). On the other hand, *SkNN*-cos performs slightly better than *SkNN*-A (improvement of 0.5%). *gCosLA* was not tested on this dataset since it contains a large number of attributes (200) and the complexity of *gCosLA* is cubic in terms of the number of dimensions.

5.5.4 Comparison between *SiLA* and *gCosLA*

Table 5.8 compares the statistical significance of the results for *SiLA* and *gCosLA* on the basis of *kNN-A* method. The performance of *gCosLA* is significantly better than that of *SiLA* on *Wine* (91.8% vs 88.4%), *German* (73.7% vs 64.6%), *Heart* (73.7% vs 65.9%) and *Letter* (99.5% vs 96.2%) data sets. Similarly *gCosLA* performs slightly better than *SiLA* on *Liver* (65.8% vs 60.9%).

On the other hand, the algorithm *SiLA* performs slightly better (shown by the symbol >)

	kNN-A (SiLA) / kNN-A (gCosLA)
Soybean	$>$
Iris	$=$
Letter	$=$
Balance	$=$
Wine	$\ll$
Ionosphere	$=$
Glass	$=$
Pima	$>$
Liver	$<$
German	$\ll$
Heart	$\ll$
Yeast	$=$

Table 5.8: Comparison between *SiLA* and *gCosLA* for *kNN*-A based on s-test

as compared to *gCosLA* on the datasets *Soybean* (100% vs 97.2%) and *Pima* (64.8% vs 62.4%). Nevertheless, *gCosLA* converged faster as compared with *SiLA* for all of these datasets as shown in figure 5.7 for *Wine* dataset in which case *SiLA* required more than 14,000 iterations in order to converge whereas *gCosLA* converged in less than 200 iterations for different value of k ($k = 1, 3$).

Similarly, *SiLA* and *gCosLA* are also compared based on *SkNN*-A decision rule as shown in table 5.9. The statistical significance of the results is mentioned where $=$ means that the difference is insignificant. The performance of *gCosLA* is significantly better than that of *SiLA* on *Wine* (92.6% vs 89.3%), *German* (72.9% vs 66.7%) and *Letter* (99.5% vs 96.2%) data sets with *SkNN-A*. Moreover, *gCosLA* performs slightly much better than *SiLA* for *Liver* (65.2% vs 60.0%) and *Heart* (71.7% vs 67.4%). On the other hand, *SiLA* was unable to perform significantly better than its counterpart on any of the 12 datasets. Nevertheless, *gCosLA* converged faster than *SiLA* while using *SkNN* as was earlier seen for *kNN* for all of these datasets.

5.5.5 Comparison between *kNN*-Euclidean and *kNN*-A (*gCosLA*)

Furthermore, the Euclidean distance is compared with the algorithm *gCosLA* while using *kNN* method in table 5.10. *gCosLA* outperforms the Euclidean distance significantly on many datasets (*Balance*, *Wine*, *German* and *Heart*). Moreover, *gCosLA* performs slightly better than the Euclidean distance on *Iris*, *Ionosphere* and *Liver*.

Similarly Euclidean distance proves to be significantly better than *gCosLA* for *Soybean*, while slightly better on *Pima*. Comparing table 5.4 and 5.10 it can be observed that the results after learning a similarity matrix are significantly better as compared to the ones using Euclidean distance.

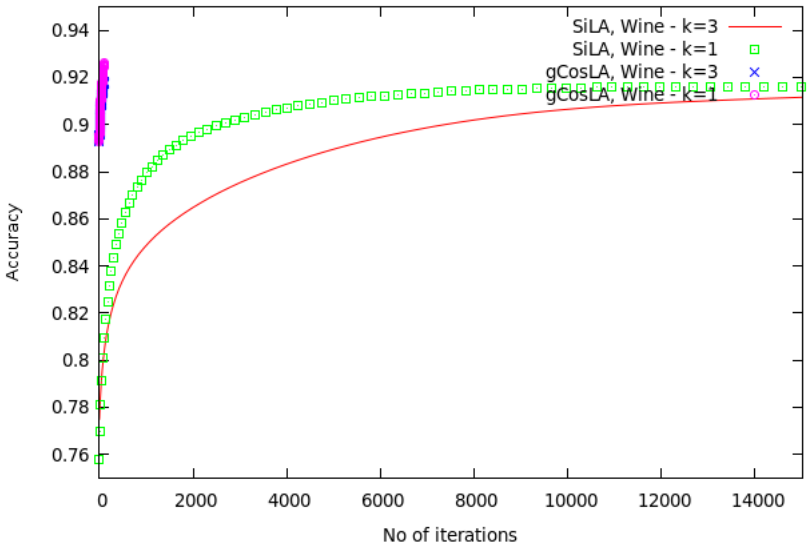

Figure 5.7: Comparison between *gCosLA* and *SiLA* in terms of rapidity for Wine

	SkNN-A (SiLA) / SkNN-A (gCosLA)
Soybean	$=$
Iris	$=$
Letter	$=$
Balance	$=$
Wine	$\ll$
Ionosphere	$=$
Glass	$=$
Pima	$=$
Liver	$<$
German	$\ll$
Heart	$<$
Yeast	$=$

Table 5.9: Comparison between *SiLA* and *gCosLA* with *SkNN*-A based on s-test

	kNN-Euclidean / kNN-A (gCosLA)
Soybean	$>$
Iris	$<$
Letter	$=$
Balance	$\ll$
Wine	$\ll$
Ionosphere	$<$
Glass	$=$
Pima	$\gg$
Liver	$<$
German	$\ll$
Heart	$\ll$
Yeast	$=$

Table 5.10: Comparison between *kNN-Euclidean* and *kNN-A* (*gCosLA*) based on s-test

5.6 *RELIEF* family of algorithms

Though basically a feature reweighting algorithm, *RELIEF* has recently been shown as belonging to the distance metric learning family by Sun and Wu [102]. In this section, the performance of *RELIEF* is compared with the cosine similarity while using *kNN* as well as *SkNN* decision rules. Furthermore, the two *RELIEF* based similarity learning algorithms i.e. *RBS* and *sRBS* are compared with the *RELIEF* algorithm using *kNN* and *SkNN*. The effect of positive, semi-definitiveness on the *RELIEF* based algorithms is also discussed.

5.6.1 Performance of cosine similarity as compared to *RELIEF*

The cosine similarity is compared with the *RELIEF* algorithm on the basis of both *kNN* as well as *SkNN* decision rules. Table 5.11 compares the *kNN*-cos with *kNN*-A for *RELIEF* algorithm. It can be observed easily that, in general, *kNN*-cos outperforms its counterpart on the basis of s-test. *kNN*-cos is significantly much better (shown by the sign $\gg$) than *kNN*-A for *RELIEF* on *Soybean*, *Iris*, *Balance*, *Wine*, *Ionosphere*, *Glass*, *Heart* and *Yeast*. Similarly *kNN*-cos is slightly better (shown by $>$) than *RELIEF* on *Pima* and *Liver*. There are only two datasets where the two algorithms perform equally well (shown by $=$ sign): *Letter* and *German* as shown in the figure 5.8.

SkNN-cos is also compared with *RELIEF* as shown in the table 5.12 and figure 5.9. Like *kNN*-cos, *SkNN*-cos performs significantly better than *SkNN*-A for *RELIEF* on all of the datasets except *Letter*.

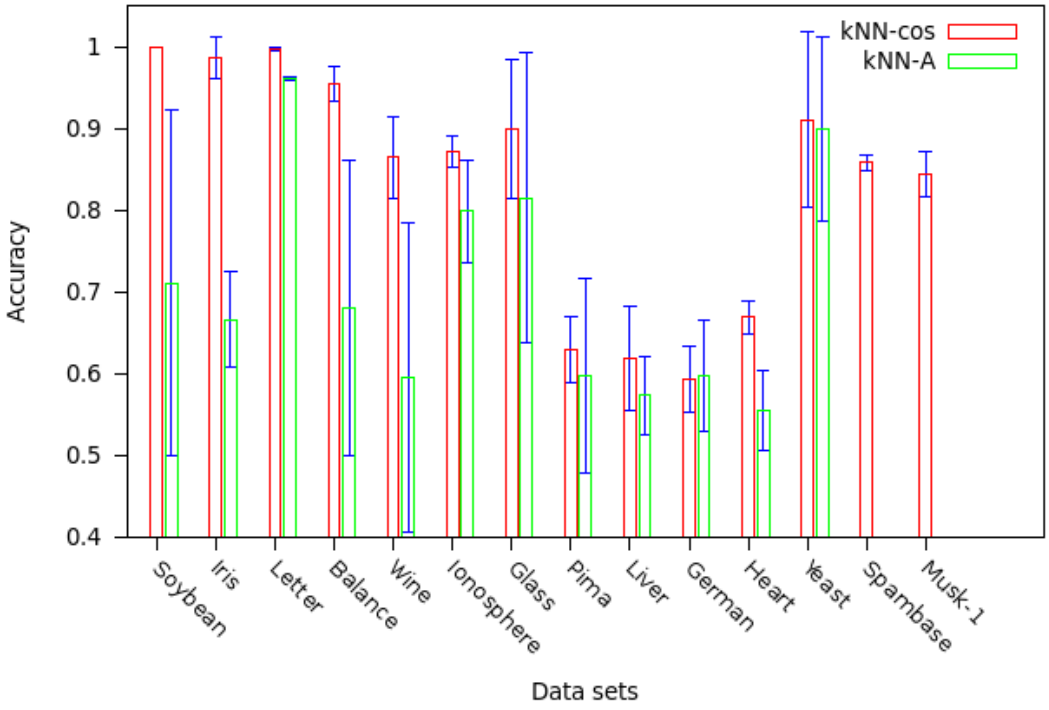

Figure 5.8: Comparison between *kNN*-cos and *RELIEF*

	kNN-cos / kNN-A (RELIEF)
Soybean	$\gg$
Iris	$\gg$
Letter	$=$
Balance	$\gg$
Wine	$\gg$
Ionosphere	$\gg$
Glass	$\gg$
Pima	$>$
Liver	$>$
German	$=$
Heart	$\gg$
Yeast	$\gg$

Table 5.11: Comparison between *kNN*-cos and *kNN*-A (*RELIEF*) based on s-test

	SkNN-cos / SkNN-A (RELIEF)
Soybean	$\gg$
Iris	$\gg$
Letter	$=$
Balance	$\gg$
Wine	$\gg$
Ionosphere	$\gg$
Glass	$\gg$
Pima	$\gg$
Liver	$\gg$
German	$\gg$
Heart	$\gg$
Yeast	$\gg$

Table 5.12: Comparison between *SkNN*-cos and *SkNN*-A (*RELIEF*) based on s-test

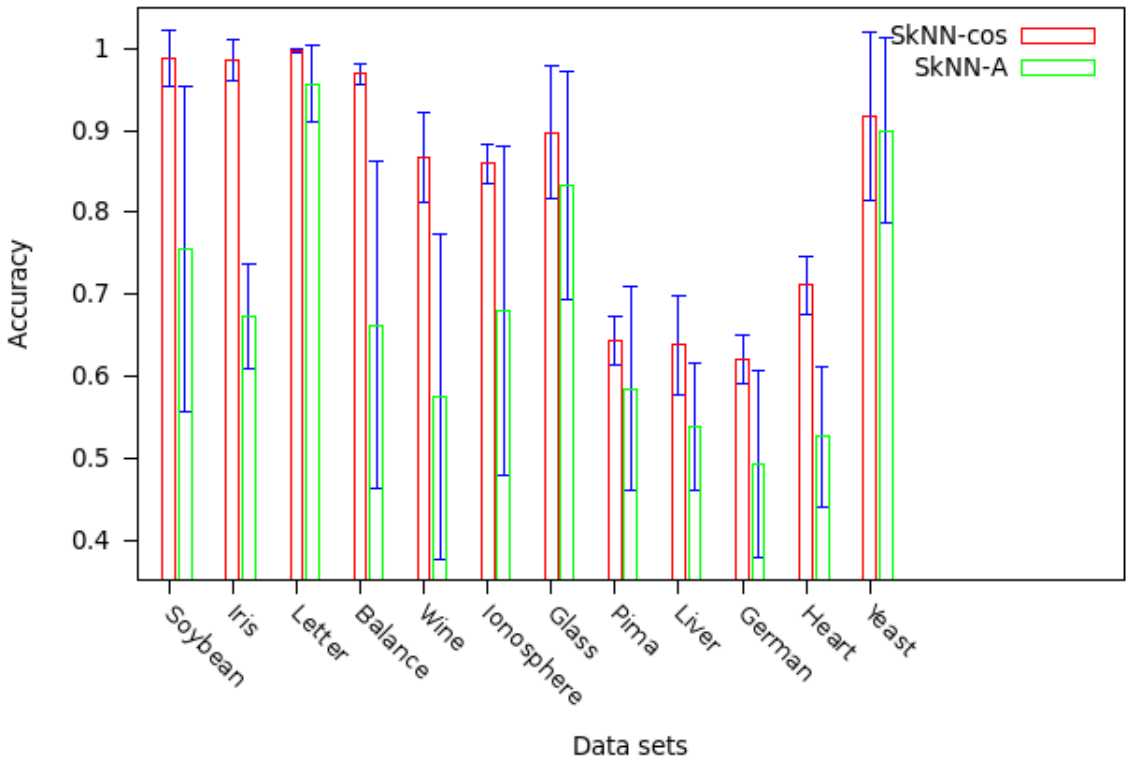

Figure 5.9: Cosine vs *RELIEF* with *SkNN* on various datasets

	kNN-A (*RELIEF*)	kNN-A (*RBS*)	kNN-A (*sRBS*)
Soybean	0.711 ± 0.211	0.750 ± 0.197 >	**1.0** ± 0.0 ≫
Iris	0.667 ± 0.059	0.667 ± 0.059	**0.987** ± 0.025 ≫
Balance	0.681 ± 0.662	0.670 ± 0.171	0.959 ± 0.016 ≫
Ionosphere	0.799 ± 0.062	0.826 ± 0.035	0.866 ± 0.015 ≫
Heart	0.556 ± 0.048	0.437 ± 0.064 ≪	**0.696** ± 0.046 ≫
Yeast	0.900 ± 0.112	0.900 ± 0.112	0.905 ± 0.113
German	0.598 ± 0.068	0.631 ± 0.020 ≫	0.609 ± 0.016
Liver	0.574 ± 0.047	0.580 ± 0.042	0.583 ± 0.015
Pima	0.598 ± 0.118	0.583 ± 0.140	0.651 ± 0.034 ≫
Glass	0.815 ± 0.177	0.821 ± 0.165	**0.886** ± 0.093 ≫
Letter	0.961 ± 0.003	0.961 ± 0.005	**0.997** ± 0.002
Wine	0.596 ± 0.188	0.630 ± 0.165	0.834 ± 0.077 ≫

Table 5.13: Comparison between different *RELIEF* based algorithms while using *kNN*-A method based on s-test

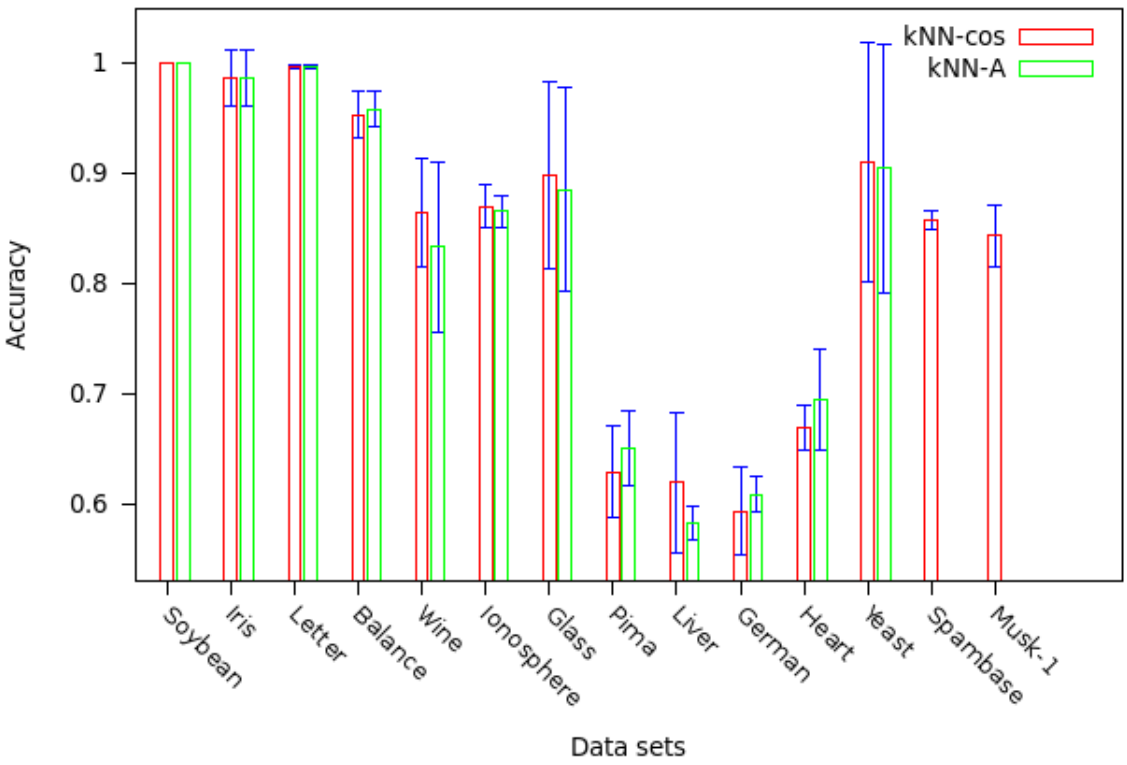

Figure 5.10: *kNN*-cos vs *kNN-sRBS* on various datasets

	SkNN-A (*RELIEF*)	SkNN-A (*RBS*)	SkNN-A (*sRBS*)
Soybean	0.756 ± 0.199	0.750 ± 0.197	0.989 ± 0.034 ≫
Iris	0.673 ± 0.064	0.667 ± 0.059	**0.987** ± 0.025 ≫
Balance	0.662 ± 0.200	0.672 ± 0.173	**0.967** ± 0.010 ≫
Ionosphere	0.681 ± 0.201	0.834 ± 0.031 ≫	**0.871** ± 0.021 ≫
Heart	0.526 ± 0.085	0.430 ± 0.057 ≪	**0.685** ± 0.069 ≫
Yeast	0.900 ± 0.113	0.900 ± 0.112	**0.908** ± 0.110
German	0.493 ± 0.115	**0.632** ± 0.021 ≫	0.598 ± 0.038 ≫
Liver	0.539 ± 0.078	0.580 ± 0.042 ≫	**0.588** ± 0.021 >
Pima	0.585 ± 0.125	0.583 ± 0.140	**0.665** ± 0.044 ≫
Glass	0.833 ± 0.140	0.816 ± 0.171 ≪	**0.884** ± 0.084 ≫
Letter	0.957 ± 0.047	0.961 ± 0.005	**0.997** ± 0.002
Wine	0.575 ± 0.198	0.634 ± 0.168 ≫	**0.840** ± 0.064 ≫

Table 5.14: Comparison between different *RELIEF* based algorithms while using *SkNN*-A based on s-test

5.6.2 Comparison between different *RELIEF* algorithms based on *kNN* decision rule

While comparing *RELIEF* with its similarity based variant (*RBS*) based on the simple *kNN* classification rule, it is evident that the later performs significantly much better only on *German* and slightly better on *Soybean* as shown in table 5.13. However *RELIEF* outperforms *RBS* for *Heart* while using *kNN*.

It can be further verified from table 5.13 that the algorithm *sRBS* performs significantly much better (≫) than the *RELIEF* algorithm for eight out of twelve datasets i.e. *Soybean*, *Iris*, *Balance*, *Ionosphere*, *Heart*, *Pima*, *Glass* and *Wine*.

5.6.3 Comparison between different *RELIEF* algorithms based on *SkNN* decision rule

While comparing *RELIEF* with its similarity based variant (*RBS*) based on the *SkNN*-A rule, it can be seen from table 5.14 that the later performs significantly much better on *Ionosphere*, *German*, *Liver* and *Wine* collections. On the other hand, *RELIEF* performs significantly much better than *RBS* on *Heart* and *Glass*.

It can further observed that *sRBS* performed significantly much better than *RELIEF* on 9 datasets out of a total of 12 i.e. *Soybean*, *Iris*, *Balance*, *Ionosphere*, *Heart*, *German*, *Pima*, *Glass* and *Wine*. On *Liver*, *sRBS* performed slightly better than the *RELIEF* algorithm. Moreover, the comparison between cosine and *sRBS* for *SkNN* is shown in figure 5.11.

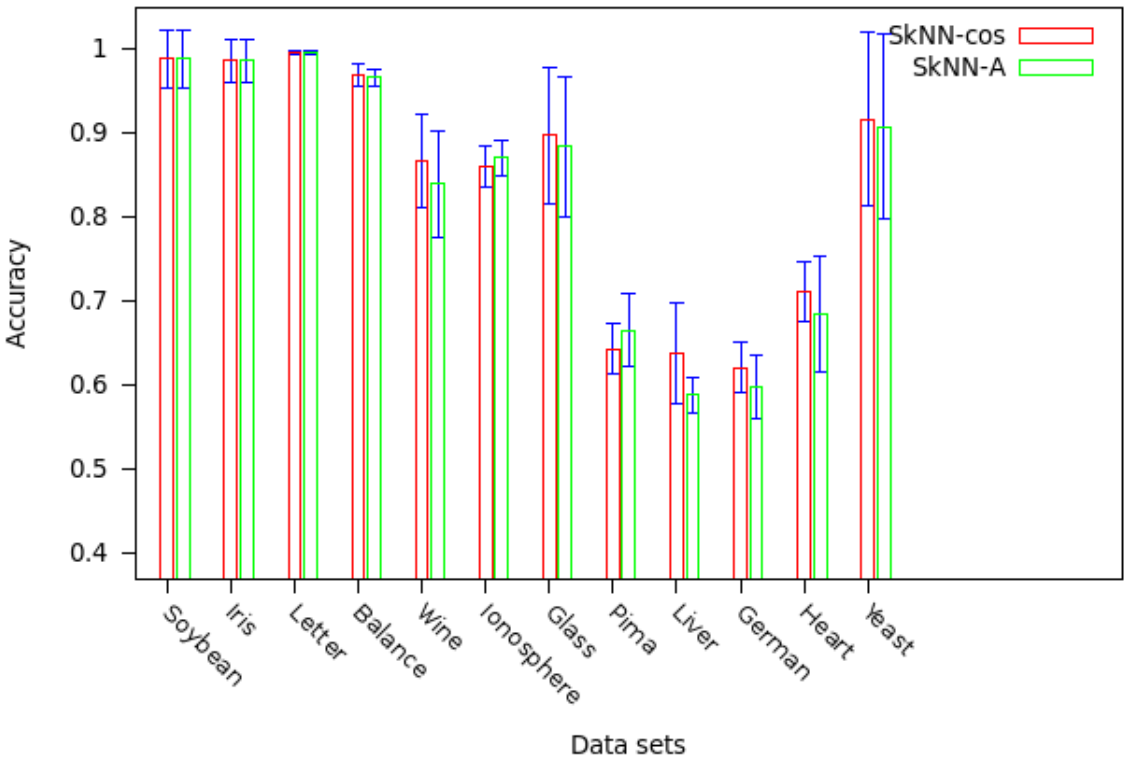

Figure 5.11: Cosine vs *sRBS* with *SkNN* rule on various datasets

5.6.4 Performance of *sRBS* as compared to *RBS*

Furthemore, the two *RELIEF* based similarity learning algorithms i.e. *RBS* and *sRBS* are compared using both *kNN* as well as *SkNN* as shown in table 5.15. On most of the datasets, the algorithm *sRBS* outperforms *RBS* for both *kNN* and *SkNN*. *sRBS* performs significantly much better (as shown by $\ll$) than its counterpart on the following datasets: *Soybean*, *Iris*, *Balance*, *Ionosphere*, *Heart*, *Pima*, *Glass* and *Wine* for the two classification rules (*kNN* and *SkNN*). On the other hand, *RBS* was able to perform slighty better than its stricter version *sRBS* on *German* while using the *kNN* rule. Similarly *RBS* performs significantly much better than *sRBS* on only one dataset i.e. *German* while using the *SkNN* classification rule. The performance of *RBS* and *sRBS* is equivalent for *Yeast*, *Liver* and *Letter*.

5.6.5 Effect of positive, semi-definitiveness on *RELIEF* based algorithms

In this subsection, the effect of learning PSD matrices is investigated for the *RELIEF* based algorithms.

RELIEF based approaches and positive, semi-definite matrices with *kNN* classification rule

In table 5.16, *RELIEF-PSD* is compared with *RELIEF*-Based Similarity learning algorithm *RBS-PSD* and its stricter version (*sRBS-PSD*) while using the *kNN* classification rule. It can be seen

	kNN-A (RBS) / kNN-A (sRBS)	SkNN-A (RBS) / SkNN-A (sRBS)
Soybean	$\ll$	$\ll$
Iris	$\ll$	$\ll$
Balance	$\ll$	$\ll$
Ionosphere	$\ll$	$\ll$
Heart	$\ll$	$\ll$
Yeast	$=$	$=$
German	$>$	$\gg$
Liver	$=$	$=$
Pima	$\ll$	$\ll$
Glass	$\ll$	$\ll$
Letter	$=$	$=$
Wine	$\ll$	$\ll$

Table 5.15: Comparison between *RBS* and *sRBS* based on s-test

	kNN-A (*RELIEF-PSD*)	kNN-A (*RBS-PSD*)	kNN-A (*sRBS-PSD*)
Soybean	0.739 ± 0.192	0.733 ± 0.220	**1.0** ± 0.0 $\gg$
Iris	0.664 ± 0.058	0.667 ± 0.059	**0.987** ± 0.025 $\gg$
Balance	0.665 ± 0.193	0.670 ± 0.171	**0.959** ± 0.016 $\gg$
Ionosphere	0.839 ± 0.055	0.826 ± 0.035	**0.880** ± 0.015 $>$
Heart	0.556 ± 0.048	0.437 ± 0.036 $\ll$	**0.693** ± 0.047 $\gg$
Yeast	0.893 ± 0.132	0.900 ± 0.112 $\gg$	**0.911** ± 0.109 $\gg$
German	**0.637** ± 0.017	0.624 ± 0.015 $<$	0.609 ± 0.016 $<$
Liver	0.574 ± 0.034	0.580 ± 0.042	**0.606** ± 0.034
Pima	0.593 ± 0.077	**0.661** ± 0.024 $\gg$	0.651 ± 0.034 $\gg$
Glass	0.819 ± 0.164	0.835 ± 0.138 $>$	**0.886** ± 0.093 $\gg$
Letter	0.961 ± 0.005	0.961 ± 0.005	**0.997** ± 0.002
Wine	0.608 ± 0.185	0.630 ± 0.165	**0.834** ± 0.077 $\gg$
Magic	0.516 ± 0.085	0.360 ± 0.007	**0.767** ± 0.009
Spambase	**0.618** ± 0.031	0.611 ± 0.020 $\ll$	0.855 ± 0.009 $\gg$
Musk-1	0.698 ± 0.055	**0.851** ± 0.033 $\gg$	0.838 ± 0.024 $\gg$

Table 5.16: Comparison between different *RELIEF* based algorithms using *kNN*-A and PSD matrices

	kNN-A (*RELIEF*) / kNN-A (*RELIEF-PSD)*
Soybean	=
Iris	=
Balance	=
Ionosphere	<
Heart	=
Yeast	≫
German	≪
Liver	=
Pima	=
Glass	=
Letter	=
Wine	=

Table 5.17: Comparison between *RELIEF* and *RELIEF-PSD* based on s-test using *kNN*

that *sRBS-PSD* performs much better than the other two algorithms on majority of the data sets. *sRBS-PSD* is statistically much better (as shown by the symbol ≫) than *RELIEF-PSD* for the following 10 datasets: *Soybean*, *Iris*, *Balance*, *Heart*, *Yeast*, *Pima*, *Glass*, *Wine*, *Spambase* and *Musk-1*. Similarly for *Ionosphere*, *sRBS-PSD* is slightly better than the *RELIEF-PSD* algorithm. On the other hand, *RELIEF-PSD* performs slightly better (<) than *sRBS-PSD* for *German* dataset.

Moreover, while comparing *RBS-PSD* with *RELIEF-PSD*, it can be observed that the former performs significantly better than the later for *Yeast*, *Pima* and *Musk-1*, and slightly better for *Glass* dataset. On the other hand, *RELIEF-PSD* was able to perform significantly better than *RBS-PSD* for *Heart* and *Spambase*, while slightly better for *German*.

While comparing *RELIEF* (with no PSD matrices) with *RELIEF-PSD* algorithm (table 5.17), it can be observed that *RELIEF-PSD* performs significantly better than *RELIEF* on *German* and slightly better on *Ionosphere*. On the other hand, *RELIEF* was able to outclass its counterpart for *Yeast*. However, for rest of the datasets the performance of these two algorithms was comparable.

RELIEF based approaches and positive, semi-definite matrices with *SkNN* classification rule

Table 5.18 compares different *RELIEF* based algorithms based on *SkNN* decision rule while using PSD matrices. It can be observed that *sRBS-PSD* performs much better than the other two algorithms on majority of the data sets as seen earlier while using the *kNN* rule . *sRBS-PSD* is statistically much better (as shown by the symbol ≫) than *RELIEF-PSD* for the following 10 datasets (out of 15): *Soybean*, *Iris*, *Balance*, *Heart*, *Yeast*, *Liver*, *Glass*, *Wine*, *Spambase* and *Musk-1*. *RELIEF-PSD* performs slightly better (<) than *sRBS-PSD* for only one dataset i.e. *German*.

Similarly, *RBS-PSD* outperforms *RELIEF-PSD* for 6 datasets (*Iris*, *Yeast*, *Liver*, *Glass*, *Spambase* and *Musk-1*) while the reverse is true for the following 3 datasets: *Balance*, *Iono-*

	SkNN-A (*RELIEF-PSD*)	SkNN-A (*RBS-PSD*)	SkNN-A (*sRBS-PSD*)
Soybean	0.783 ± 0.163	0.733 ± 0.220	**0.983** ± 0.041 ≫
Iris	0.571 ± 0.164	0.667 ± 0.059 ≫	**0.987** ± 0.025 ≫
Balance	0.708 ± 0.175	0.672 ± 0.173 ≪	**0.967** ± 0.010 ≫
Ionosphere	0.886 ± 0.028	0.834 ± 0.031 ≪	**0.889** ± 0.011
Heart	0.533 ± 0.067	0.437 ± 0.036 ≪	**0.685** ± 0.069 ≫
Yeast	0.897 ± 0.122	0.900 ± 0.112 ≫	**0.914** ± 0.106 ≫
German	**0.625** ± 0.035	0.624 ± 0.015	0.598 ± 0.038 <
Liver	0.528 ± 0.085	0.580 ± 0.042 ≫	**0.609** ± 0.035 ≫
Pima	0.659 ± 0.027	0.658 ± 0.030	**0.665** ± 0.044
Glass	0.768 ± 0.235	0.835 ± 0.138 ≫	**0.884** ± 0.084 ≫
Letter	0.961 ± 0.008	0.961 ± 0.004	**0.997** ± 0.002
Wine	0.606 ± 0.177	0.634 ± 0.168	**0.840** ± 0.064 ≫
Magic	0.539 ± 0.109	0.360 ± 0.007	**0.777** ± 0.009
Spambase	0.583 ± 0.075	0.611 ± 0.020 ≫	**0.857** ± 0.010 ≫
Musk-1	0.712 ± 0.037	**0.857** ± 0.029 ≫	0.842 ± 0.010 ≫

Table 5.18: Comparison between different *RELIEF* based algorithms using *SkNN*-A and PSD matrices

sphere and *Heart*.

Table 5.19 compares the effect of using PSD matrices with the *RELIEF* algorithm while using the *SkNN* decision rule. It can be observed that *RELIEF-PSD* performs significantly better than *RELIEF* on *Balance*, *Ionosphere*, *German* and *Pima*. On the other hand, *RELIEF* was able to outclass its counterpart for *Iris*, *Yeast* and *Glass*. The performance of these two algorithms was comparable for the remaining collections.

Performance of *sRBS-PSD* as compared to *RBS-PSD*

Table 5.20 compares statistically the results obtained while using *RBS-PSD* and *sRBS-PSD* algorithms. The later outperforms the former for the following 7 datasets (out of 13 considered for comparison): *Soybean*, *Iris*, *Balance*, *Ionosphere*, *Heart*, *Glass* and *Wine* with both *kNN* as well as *SkNN*. *RBS-PSD* performs slightly better than its counterpart for *German* while using the *SkNN* rule. However, for the rest of the datasets, the two algorithms' performance is comparable.

5.7 How *SiLA* and *gCosLA* perform as compared to the state of the art approaches

In this section, *SiLA* and *gCosLA* are compared with different state of the art methods in metric learning. A detailed comparison between *SiLA* and *gCosLA* and several state of the art ones is

	SkNN-A (*RELIEF*) / SkNN-A (*RELIEF-PSD)*
Soybean	$=$
Iris	$\gg$
Balance	$\ll$
Ionosphere	$\ll$
Heart	$=$
Yeast	$\gg$
German	$\ll$
Liver	$=$
Pima	$\ll$
Glass	$\gg$
Letter	$=$
Wine	$=$

Table 5.19: Comparison between *RELIEF* and *RELIEF-PSD* based on s-test using *SkNN*

	kNN-A (RBS-PSD) / (sRBS-PSD)	SkNN-A (RBS-PSD) / (sRBS-PSD)
Soybean	$\ll$	$\ll$
Iris	$\ll$	$\ll$
Balance	$\ll$	$\ll$
Ionosphere	$\ll$	$\ll$
Heart	$\ll$	$\ll$
Yeast	$=$	$=$
German	$=$	$>$
Liver	$=$	$=$
Pima	$=$	$=$
Glass	$\ll$	$\ll$
Letter	$=$	$=$
Wine	$\ll$	$\ll$
Musk-1	$=$	$=$

Table 5.20: Comparison between *RBS-PSD* and *sRBS-PSD* based on s-test

	gCosLA	SiLA	SNN	MCML	LMNN	ITML	Multiclass SVM
Balance	**0.976**	0.952	0.879	0.925	0.916	0.920	0.922
Wine	0.857	0.806	0.951	0.837	**0.974**	**0.974**	0.801
Iris	**0.967**	**0.967**	0.934	**0.967**	0.953	0.961	0.956

Table 5.21: Different similarity and metric learning algorithms on UCI datasets

give in Table 5.21. The first one ([72]) learns similarity whereas the next three ([28, 41, 112]) are interested in learning distances with *kNN* algorithm. The algorithms are: Similarity Learning with Neural Network *SNN*, Information Theoretic Metric Learning *ITML*, Maximally Collapsing Metric Learning *MCML*, Large Margin Nearest Neighbor *LMNN* and a multiclass version of SVMs [25]. To compare the methods based on *SiLA* and *gCosLA* with different approaches, a multiclass version for both of these algorithms was used followed by the calculation of the global accuracy. Furthermore, only the standard kNN approach (*kNN-A*) and not the symmetric one (*SkNN-A*) was used in order to have a fair comparison.

The methods are compared on three UCI datasets (*Iris*, *Balance* and *Wine*) common to all of the previous approaches.

Comparing *gCosLA* with *SiLA*, it can be observed that for *Balance* and *Wine*, *gCosLA* not only outperformed *SiLA* but it converged very rapidly (in terms of number of iterations and time) also. The performance for *gCosLA* is on a par with that of *SiLA* on *Iris* but nevertheless, *gCosLA* is faster as was seen for the binary version of these two algorithms.

While comparing *SiLA* and *gCosLA* with *SNN*, it can be noted that the algorithms *SiLA* and *gCosLA* outperformed *SNN* for *Balance* and *Iris*. However for *Wine*, *SNN* has got a much better performance as compared with *SiLA* and *gCosLA*. The primary reason is that *SNN* was able to down-weigh an influential attribute for *Wine* whereas *SiLA* and *gCosLA* were unable to do so, since they do not perform feature selection while *SNN* does so.

SiLA and *gCosLA* performed much better than *MCML* for *Balance* whereas the three algorithms got the same accuracy for *Iris*. *SiLA* and *gCosLA* also outperformed *LMNN* and *ITML* on two out of three data sets, namely *Balance* and *Iris*. However, *LMNN* and *ITML* performed better on *Wine* because they were able to down-weigh an influential attribute just like *SNN*.

In comparison with *Multiclass SVM*, *gCosLA* performed much better for all of the three collections whereas *SiLA* was better for *Balance* and *Iris*.

gCosLA and *SiLA* are further compared with many other state of the art approaches like Xing's algorithm [114], KRCA (Kernal Relevant Component Analysis) [104], IGML (Linear Information Geometric Approach for Metric Learning), KIGML (Kernel Information Geometric Approach for Metric Learning), Euclidean distance and Mahalanobis distance. The results for distance learning methods are copied from Wang and Jin [107] whereas we report the results for cosine similarity and our similarity metric learning approaches i.e. *SiLA* and *gCosLA*. Moreover, Wang and Jin have have found that the best value of k is 4. However, in our case, we find the value of k using double cross-validation. Table 5.22 and 5.23 give the results where the best ones are written in bold. Similarly the ranking for different algorithms is given in table 5.24 where 1 represents the best algorithm whereas 10 stands for the worst.

It can be observed that for collections like *Iris*, *Soybean*, *Ionosphere*, *Sonar* and *Glass*, for which the cosine measure performs better than the Euclidean distance, the cosine based methods i.e. Cosine, *SiLA* and *gCosLA* outclass the distance based ones. This means that for these collections, it is better to use similarity based methods rather than learning distance metrics. The standard cosine measure has got the highest accuracy for *Iris* and *Soybean* while it is ranked second for *Sonar* as well as *Glass* datasets. Similarly *SiLA* got the first position for *Soybean* and *Ionosphere* whereas it got the third rank for *Sonar*, *Glass* and *Optdigits*. *gCosLA* got the third rank for *Iris* and *Sonar*.

	Eucl	Mahal	Xing	LMNN	ITML
Iris	5.0 ± 2.9	10.8 ± 3.3	3.5 ± 1.9	4.5 ± 2.1	4.3 ± 2.7
Soy	6.0 ± 5.1	2.8 ± 3.2	1.1 ± 2.2	2.2 ± 2.1	0.7 ± 1.0
Iono	17.8 ± 1.6	18.4 ± 2.0	**10.3 ± 1.3**	15.0 ± 1.9	11.1 ± 2.6
Sonar	28.9 ± 4.2	28.9 ± 3.8	28.9 ± 4.2	20.3 ± 4.4	28.3 ± 6.3
Glass	35.5 ± 3.5	34.9 ± 3.2	41.7 ± 4.9	34.9 ± 3.2	36.2 ± 3.4
Opt	2.1 ± 0.3	5.9 ± 0.5	12.3 ± 0.9	1.6 ± 0.3	2.1 ± 0.3
Wine	29.6 ± 3.6	7.5 ± 2.2	10.8 ± 4.6	**4.1 ± 1.8**	7.7 ± 3.0
Seg	23.6 ± 3.1	16.9 ± 3.6	23.2 ± 3.4	14.7 ± 1.9	16.6 ± 5.0
Wave	19.5 ± 0.6	36.1 ± 0.8	**17.0 ± 0.8**	19.1 ± 0.7	19.7 ± 0.7
Pima	28.0 ± 1.8	27.8 ± 2.0	27.9 ± 1.7	**27.1 ± 1.7**	27.8 ± 1.7

Table 5.22: Comparison of *SiLA* and *gCosLA* with many state of the art approaches - I

	KRCA	IGML	KIGML	Cosine	gCosLA	SiLA
Iris	4.1 ± 1.6	2.7 ± 1.7	3.9 ± 2.8	**2.0 ± 3.0**	3.3 ± 3.3	3.3 ± 3.3
Soy	0.1 ± 0.8	1.8 ± 2.1	0.4 ± 1.3	**1.0 ± 0.0**	8.9 ± 9.3	**1.0 ± 0.0**
Iono	17.2 ± 1.6	16.6 ± 1.8	14.2 ± 1.6	12.9 ± 2.0	13.4 ± 2.6	**8.9 ± 3.3**
Sonar	26.5 ± 4.6	28.1 ± 4.5	**14.6 ± 4.0**	18.5 ± 5.1	20.0 ± 4.01	20.0 ± 9.0
Glass	36.9 ± 2.7	35.8 ± 2.3	**33.3 ± 3.1**	33.8 ± 8.5	36.2 ± 4.9	34.8 ± 8.4
Opt	2.1 ± 0.3	3.2 ± 0.3	**1.4 ± 0.2**	2.1 ± 0.3	2.5 ± 0.6	2.0 ± 0.3
Wine	4.6 ± 1.5	5.0 ± 1.6	6.1 ± 1.9	21.1 ± 5.2	14.3 ± 7.0	19.4 ± 8.9
Seg	15.0 ± 2.7	12.9 ± 3.4	**12.4 ± 3.5**	30.5 ± 9.9	24.3 ± 6.16	26.2 ± 12.5
Wave	20.1 ± 0.7	30.6 ± 0.7	21.1 ± 0.6	20.2 ± 1.2	20.2 ± 1.2	20.7 ± 1.1
Pima	27.8 ± 1.6	27.6 ± 1.9	27.8 ± 2.0	37.0 ± 4.4	38.2 ± 5.1	35.3 ± 1.8

Table 5.23: Comparison of *SiLA* and *gCosLA* with many state of the art approaches - II

Furthermore, it is better to use algorithms based on distance metrics for collections on which the Euclidean distance performs much better than the standard cosine i.e. *Segmentation*, *Waveform* and *Pima*. This suggests that the decision to use either the similarity or distance metric learning could be based on the relative performance of the cosine similarity and the Euclidean distance.

	1	2	3	4	5	6	7	8	9	10	11
Iri	C	IG	g, S	X	KI	KR	IT	L	E	M	
Soy	C, S	KR	KI	IT	X	IG	L	M	E	g	
Ion	S	X	IT	C	g	KI	L	IG	KR	E	M
Son	KI	C	g, S	L	KR	IG	IT	M, E, X			
Gla	KI	C	S	L, M	E	IG	IT, g	KR	X		
Opt	KI	L	S	C, E, IT, KR	g	IG	M	X			
Win	L	KR	IG	KI	M	IT	X	g	S	C	E
Seg	KI	IG	L	KR	IT	M	X	E	g	S	C
Wav	X	L	E	IT	KR	C, g	S	KI	IG	M	
Pim	L	IG	KR, IT, KI, M	X	E	S	C	g			

Table 5.24: Ranking of different algorithms on UCI datasets

Although for *Wine* dataset, similarity learning algorithms perform better than the Euclidean distance, yet they are not ranked in the top algorithms because of the presence of an influential attribute. Algorithms like *LMNN* and *KRCA* were able to downweigh this influential attribute as opposed to the similarity learning ones.

On this set of collections, it can be observed that the similarity learning approaches (*SiLA* and *gCosLA*) have difficulties to outclass the standard cosine measure, unlike what we observed on other collections (table 5.5). We know of no way of assessing in advance whether similarity metric learning should be preferred over the standard cosine on a particular collection, and this should be investigated in the future.

5.8 Comparison between *kNN*-cos and *SkNN*-cos

Table 5.25 compares the performance of *kNN* and *SkNN* on various datasets with the cosine measure. *s-test* was used to find the statistical signifance of the results. *SkNN* performed significantly much better ($\gg$) than *kNN* on *Balance*, *German*, *Heart* and *Yeast* datasets while slightly better ($>$) on *Pima* dataset. On the other hand, *kNN* was able to perform significantly better ($\gg$) than its symmetric variant only on one of the datasets i.e. *Ionosphere*.

Although the accuracy for *SkNN* on *Liver* was 63.8% against 62.0% while using the standard *kNN*, the results were not significant enough. These results show that it is much better, in general, to use the symmetric version of *kNN* rather than the original *kNN* classification rule.

Figure 5.12 describes the performance of *kNN*-cos and *SkNN*-cos on different datasets. The precision as well as standard deviation is shown in the figure.

5.9 Conclusion

Most of the works involving metric learning have restricted themselves to learning distance metrics. However we showed that cosine similarity should be preferred over the Euclidean distance on non-textual data collections apart from the usual textual ones. A statistical test, s-test was

	kNN-cos	*SkNN*-cos
Soybean	1.0 ± 0.0	0.989 ± 0.034
Iris	0.987 ± 0.025	0.987 ± 0.025
Letter	0.997 ± 0.002	0.997 ± 0.002
Balance	0.954 ± 0.021	0.969 ± 0.013 ≫
Wine	0.865 ± 0.050	0.867 ± 0.055
Ionosphere	0.871 ± 0.019 >	0.860 ± 0.024
Glass	0.899 ± 0.085	0.898 ± 0.081
Pima	0.630 ± 0.041	0.643 ± 0.030 >
Liver	0.620 ± 0.064	0.638 ± 0.060
German	0.594 ± 0.040	0.620 ± 0.030 ≫
Heart	0.670 ± 0.020	0.711 ± 0.036 ≫
Yeast	0.911 ± 0.108	0.917 ± 0.103 ≫
Spambase	0.858 ± 0.009	
Musk-1	0.844 ± 0.028	

Table 5.25: Comparison between *kNN*-cos and *SkNN*-cos on s-test

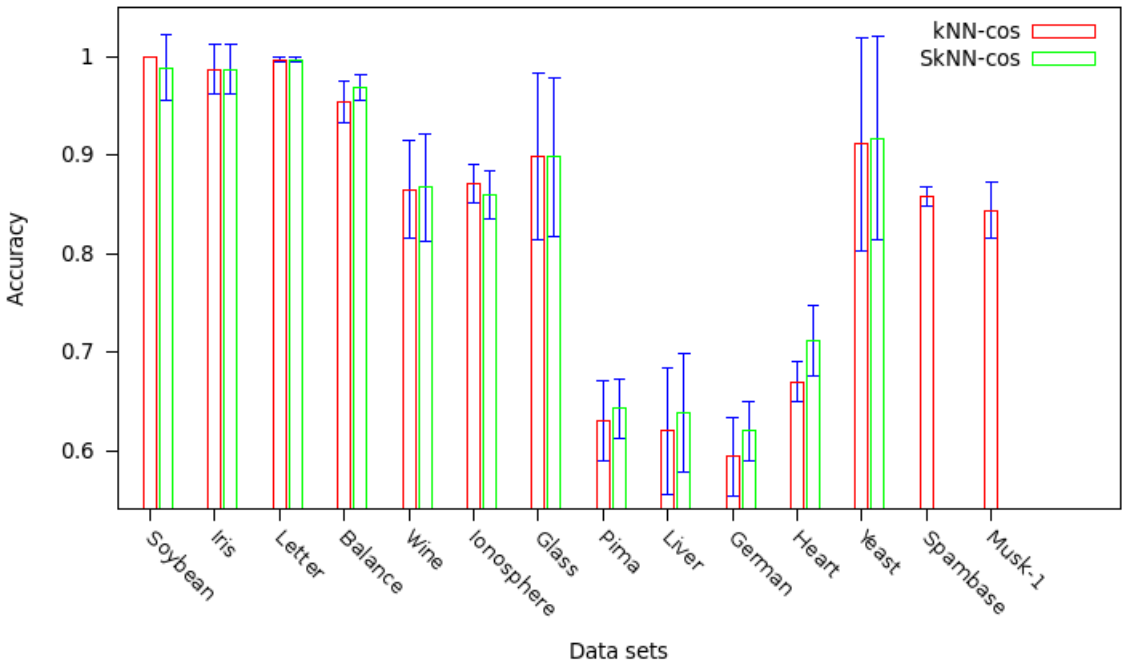

Figure 5.12: *kNN*-cos vs *SkNN*-cos on various datasets

performed to assess whether the results are significanly different or not. Furthermore, double cross-validation technique was employed in order to determine the different parameters of various algorithms. The cosine similarity outperformed the Euclidean distance on some of the collections like *Iris*, *Balance*, *Wine*, *Ionosphere* and *Spambase* using binary classification. The uncontrained similarity learning algorithm, *SiLA* as well as the generlized Cosine similarity Learning Algorithm *gCosLA* were compared with the standard cosine using both the *kNN* as well as *SkNN* rules. On many of the data sets, the algorithms learning a similarity metric performed significantly better than the standard cosine similarity. Moreover, *gCosLA* performed better than *SiLA* on many of the data sets.

While comparing the *RELIEF* family of algorithms, we found that the stricter version of *RELIEF*-Based Similarity algorithm (*sRBS* performed significantly much better than its counterparts on most of the datasets using the two classification rules: *kNN-A* as well as *SkNN-A*. This proved that it is far better to use the O-1 loss function rather than its approximation as was done in the case of *RELIEF* and *RBS*. Moreover, the performance of *RELIEF* algorithm improved with the use of positive, semi-definite matrices.

gCosLA and *SiLA* were also compared with many state of the art approaches in metric learning like Xing's algorithm [114], Large Margin Nearest Neighbor classfication (*LMNN*) [112], Information Theoretic Metric Learning (*ITML*) [28], Maximally Collapsing Metric Learning algorithm (*MCML*) [41], Similarity Learning with Neural Networks (*SNN*), Kernel Relevant Component Analysis (*KRCA*) [104], Linear Information Geometric approach for Metric Learning (*IGML*) [107], Kernel Information Geometric approach for Metric Learning (*KIGML*) [107]. It was observed that for collections like *Iris*, *Soybean*, *Ionosphere*, *Glass* and *Sonar*, on which cosine performs better than the Euclidean distance, similarity metric learning algorithms outperform the distance metric learning ones. On the other hand, it is better to use distance metric learning algorithms on collections like *Segmentation*, *Waveform* and *Pima* for which the Euclidean distance proves to be a better option than the standard cosine. Although the similarity based methods perform better than the Euclidean distance for *Wine*, yet they do not rank among the top algorithms because of the presence of an influential attribute. Algorithms like *LMNN* and *KRCA* were able to reduce the influence of this attribute as opposed to the similarity learning ones.

Chapter 6

Conclusion and Perspectives

Contents

Machine learning is the study of computer algorithms that improve their performance automatically by experience. As different data types exhibit different properties, it can be useful to learn the geometry underlying the data to be processed. Indeed, many recent works, e.g. Weinberger et al. [112], Jain et al. [53] etc., have shown that learning a metric, based on the geometry of the space containing the data, is often a better idea than assuming the presence of a simple geometric structure. However, most of the works in the field of metric learning work only with distance metric learning and do not consider similarities e.g. Goldberger et al. [42], Xing et al. [114], Davis et al. [28], Globerson et al. [41]. Traditionally the cosine measure has been shown to perform well for the textual datasets [95]. However some recent works like Qamar et al. [87], Peterson et al. [84] have shown that cosine similarity should also be preferred over the distances on non-textual data collections.

6.1 Main contributions

We have focussed here on learning (complete) similarities from data to be used in *kNN* classification, considering different scenarios, some relying on few labelled data, others making use of data sets fully annotated. In situations where only a small amount of annotation is available, one can not learn complex structures, and we limited ourselves to learning a few meta-parameters controlling cosine-based similarities. This work was appropriately deployed in the context of the INFILE tracks, during the evaluation campaigns CLEF 2008 and CLEF 2009. In situations with more annotation, we have considered two possible generalizations of existing, well-established similarity measures. These two generalizations mainly differ in the constraints they rely on. The first one imposes almost no constraint on the transformation to be used; in particular, the normalizations considered do not depend on the metric learned, which makes the learning process easier. The second however imposes strong constraints on the metric learned, in particular that it should correspond to a true cosine measure in an embedded space. As such, it should rely on semi-definite matrices, with a normalization which does depend on the metric learned. If the first generalization was based on the perceptron algorithm family, the second one requires a different approach. In both cases, we have provided theoretical proofs of the correct behavior of our algorithms.

Learning a metric implies to model dependencies between features, and weigh them correctly. This objective is shared by feature re-weighting procedures, and several recent works have emphasized the links between such procedures (as *RELIEF*) and supervised learning of metrics. We have studied here this link in detail, and have shown that the objective function approximated in the *RELIEF* family was not optimal. We have then throughly evaluated our algorithms, trying to assess when they provided a significant improvement in the results. We have furthermore compared their performance with alternative approaches. It is always difficult to compare two approaches which are very different in nature. We believe that the comparison we have performed indicates that similarity learning methods, and the algorithms we have proposed for this, are valuable machine learning tools which can complement existing distance metric learning ones. We now provide a summary of the main contributions of our work.

1. A thorough study of metric learning algorithms including the distance metric learning algorithms as well as the similarity metric learning ones is performed.

2. An information filtering technique is developed which can be used to learn cosine based category specific thresholds, provided some sort of supervision is present. Online and Batch algorithms were developed for the information filtering process. Both methods were able to get the best F-score during INFILE track of CLEF campaign in the years 2008 and 2009.

3. Cosine similarity was shown to perform better than the Euclidean distance on many datasets.

4. An unconstrained similarity metric learning algorithm called *SiLA* was developed to learn the similarity metrics for *kNN* classification. The normalization in *SiLA* is totally independent of the similarity matrix which helps to learn different types of similarity functions based on diagonal, symmetric or asymmetric matrices. The convergence and the generalization properties were established and the proofs have been provided. A statistical test, s-test, was used to statistically analyze all of the results.

5. The links between *RELIEF* and *SiLA* were studied. This was followed by the development of a *RELIEF* Based Similarity (*RBS*) learning algorithm. However it turned out that *RBS* did not perform well in practice. The main reason is that *RBS* tries to optimize a cost function approximating the $0-1$ loss on the footsteps of *RELIEF*. We showed that this approximation is loose, and proposed a stricter version of *RBS*, called *sRBS*, based on a cost function closer to the $0-1$ loss. *sRBS* performed significantly better than the other *RELIEF* based algorithms indicating in particular that the $0-1$ loss is a more appropriate cost function that the one implicitly used by *RELIEF*.

6. Lastly, an algorithm based on the generalized cosine similarity was developed. The algorithm is named *gCosLA* for Generalized Cosine similarity metric Learning Algorithm. The normalization in the case of *gCosLA* was dependent on the similarity matrix and the similarity matrix belonged to the class of positive, semi-definite matrices. The results showed that *gCosLA* was significantly better than *SiLA* on many of the collections considered.

7. *SiLA* and *gCosLA* were compared with many state of the art metric learning algorithms and were found to be performing very well in situations where similarities are useful. As such, they constitute new machine learning tools which can adequately complement existing distance metric learning algorithms.

Having reviewed the main contributions of our thesis, we now turn to the limitations of our work, and the perspectives it opens.

6.2 Limitations and Perspectives

As with any machine learning algorithm, the similarity learning algorithms have their own limitations. The process of threshold learning does not perform like the metric learning one as it does not take into account the geometry of the space containing the data. Although *SiLA* was used with a large dataset of *Newsgroup*, yet it remains to be shown how it can work with massive datasets. The complexity of *SiLA* is quadratic in the number of dimensions. Though a very promising algorithm, *gCosLA* is a bit slow owing to its cubic complexity in terms of the number

of dimensions. This is the reason why *gCosLA* took a lot of time with the *Newsgroup* dataset. As *gCosLA* learns positive, semi-definite (PSD) matrices using eigenvalue decomposition, its complexity can be reduced using eigenvalue approximation methods e.g. Lanczos algorithm and its specialized variants, but this has to be investigated more throughly. There is yet another way in which the complexity of *gCosLA* could be reduced, using the fact that any PSD matrix M could be decomposed into U^tU where U is a matrix of lower rank. In this case, the constraints on semi-definiteness need not be enforced, which leads to a faster algorithm (such a trick is employed for example in [47] in the context of distance learning). However, even though faster, the problem of learning U is not necessarily easier, because of local optima. It is thus not clear whether this strategy would be beneficial to *gCosLA*, and further investigation is necessary here.

Related to speed issues, but with additional implications, is the lack of control of the *aggressiveness* of the update rules underlying the algorithms we have presented (in particular *SiLA*). One of the strengths of the Passive-Aggressive family is precisely such a control, which could be added in our case as well. This being said, tuning meta-parameters is not always an easy task, and may lead to additional computation. One can nevertheless hope that a valid solution would be attained faster, and thus requiring less updates and leading to an overall faster learning process. Because of the potential practical and theoretical implications they can have, we believe it would be worth to investigate in a near future the use of aggressiveness parameters in our algorithms.

Another limitation of our work lies in the fact that only global similarity metrics were learned (by resorting to binary classification and the standard one-vs-the-rest rule, several matrices are in fact learned to solve a multi-class categorization problem; however, all the matrices are global in the sense that they are not adapted to specific regions of the space). Another possibility is to learn different local similarity metrics in different parts of the input space as is the case for Multi-Metric LMNN algorithm [112]. One possibility with the approach we have followed would be to consider neighborhood regions around each point and all the examples they contain, and then learn matrices for each such regions. The classification of a data point would then involve only the regions which yield the neighborhood of the point. If this approach seems simple and promising, it would certainly involve more computation than the current ones. They thus call for simpler and faster versions of the algorithms we have presented.

Lastly, another perspective we would like to explore is the use of *SiLA* algorithm in a different context, namely the one of Information Retrieval (IR), as this domain heavily relies on the cosine similarity measure, which could be learned from existing relevance judgements. In IR, the similarity is calculated between a query q and a document d. A possible application of *SiLA* in this case could go along the following lines: the query q could replace $x^{(i)}$, repeated $N1$ (number of retrieved documents judged relevant by the user) times; the target neighbor y could then be chosen arbitrarily, or according to the standard cosine similarity measure, from the set of relevant documents, whereas z would represent the closest non-relevant documents. As mentioned above, the matrix A could be learned using existing relevance judgements, or potentially user feedback. We plan on investigating these different possibilities in the near future.

Appendix A

Proofs for Theorems for *SiLA* and *gCosLA*

This Appendix gives the proof of theorems 1 and 2 for *SiLA* and theorem 1 for *gCosLA*. Theorem 1 for *SiLA* is based on Block [13] and Novikoff [77] and was used in Freund and Schapire [37]. Similarly, the proof for theorem 2 of *SiLA* parallel the one provided in Collins [20] adapted from Freund and Schapire [37].

A.1 Theorem 1 - *SiLA* (separable case)

For any training sequence $\mathcal{S} = ((x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)}))$ *separable with margin* γ*, for one iteration (epoch) of the (on-line) update rule of SiLA*

$$\textit{Number of mistakes} \leq R^2/\gamma^2$$

where R *is a constant such that:*

$$\forall i, \forall (z_1, \cdots, z_k) \in \bar{c}^i, \| \sum_{y \in T(i)} \phi(x^{(i)}, y) - \sum_{n=1}^{k} \phi(x^{(i)}, z_n) \| \leq R$$

Proof: Let α^k be the weight vector before the k'th mistake is made. It follows that $\alpha^1 = 0$ (since initial weights are zero). Suppose that the k'th mistake is made at the i'th example. Let $B(i)$ represent the k nearest neighbors from the class $\bar{c}^{(i)}$:

$$B(i) = \text{kNN}(A^{(t)}, x^{(i)}, \bar{c}^{(i)})$$

The update for the *SiLA* algorithm can be written in the vector notation in the following manner:

$$\alpha^{k+1} = \alpha^k + \sum_{y \in T(i)} \phi(x^{(i)}, y) - \sum_{z \in B(i)} \phi(x^{(i)}, z)$$

This is followed by taking the inner product of both sides with the vector $\mathbf{U}$:

$$\begin{aligned} \mathbf{U}.\alpha^{k+1} &= \mathbf{U}.\alpha^k + \mathbf{U}.\textstyle\sum_{y \in T(i)} \phi(x^{(i)}, y) - \mathbf{U}.\sum_{z \in B(i)} \phi(x^{(i)}, z) \\ &\geq \mathbf{U}.\alpha^k + \gamma \end{aligned}$$

where the inequality follows from definition 1 of *SiLA*. As $\alpha^1 = 0$, and hence $\mathbf{U}.\alpha^1 = 0$, it follows by induction on k that $\forall k\ \mathbf{U}.\alpha^{k+1} \geq k\gamma$. Since $\mathbf{U}.\alpha^{k+1} \leq \|\mathbf{U}\|\|\alpha^{k+1}\|$, it follows that:

$$\|\alpha^{k+1}\| \geq k\gamma \tag{A.1}$$

which gives the lower bound for $\|\alpha^{k+1}\|$.
The upper bound for $\|\alpha^{k+1}\|^2$ can now be derived in the following manner:

$$\begin{aligned}\|\alpha^{k+1}\|^2 &= \|\alpha^k\|^2 + \|\textstyle\sum_{y\in T(i)} \phi(x^{(i)}, y) - \sum_{z\in B(i)} \phi(x^{(i)}, z)\|^2 \\ &\quad +2\alpha^k.(\textstyle\sum_{y\in T(i)} \phi(x^{(i)}, y) - \sum_{z\in B(i)} \phi(x^{(i)}, z)) \\ &\leq \|\alpha^k\|^2 + R^2\end{aligned}$$

where the inequality follows as $\|\sum_{y\in T(i)} \phi(x^{(i)}, y) - \sum_{z\in B(i)} \phi(x^{(i)}, z)\| \leq R^2$ by assumption, and $\alpha^k.(\sum_{y\in T(i)} \phi(x^{(i)}, y) - \sum_{z\in B(i)} \phi(x^{(i)}, z)) \leq 0$ since z is the highest scoring candidate for x_i under the parameters α^k (as it is the closest example from all of the classes other than $c^{(i)}$). It follows by induction that:

$$\|\alpha^{k+1}\|^2 \leq kR^2 \tag{A.2}$$

which represents the upper bound for $\|\alpha^{k+1}\|^2$.
The inequalities for the lower bound A.1 and the upper bound A.2 can be combined to complete the proof:

$$\forall k\ \ k^2\gamma^2 \leq \|\alpha^{k+1}\|^2 \leq kR^2 \implies k \leq \frac{R^2}{\gamma^2} \quad \square$$

A.2 Theorem 2 - *SiLA* (non separable case)

For any training sequence $\mathcal{S} = ((x^{(1)}, c^{(1)}), \cdots, (x^{(n)}, c^{(n)}))$ *separable with margin* γ*, for one iteration (epoch) of the (on-line) update rule of SiLA*

$$\textit{Number of mistakes} \leq min_{\alpha,\gamma} \frac{(R+D_{\alpha,\gamma})^2}{\gamma^2}$$

where R *is a constant such that:*

$$\forall i, \forall (z_1, \cdots, z_k) \in \bar{c}^i,\ \|\sum_{y\in T(i)} \phi(x^{(i)}, y) - \sum_{n=1}^{k} \phi(x^{(i)}, z_n)\| \leq R,$$

and the min is taken over α *and* γ *such that* $\|\alpha\| = 1, \gamma > 0$*.*

Proof: In order to prove Theorem 2, the representation $\phi(x, y) \in \mathbb{R}^d$ is modified to $\phi(x, y) \in \mathbb{R}^{d+n}$ in the following manner:

For $i = 1, \cdots, d$ define $\phi_i(x, y) = \phi_i(x, y)$. For $j = 1, \cdots, n$ define $\phi_{d+j}(x, y) = \Delta$ if $(x, y) = (x_j, y_j)$, 0 otherwise, where Δ is a parameter and is greater than 0. Similarly, consider a $\mathbf{U}$, γ pair, and corresponding values for ϵ_i as defined above. Consequently a modified parameter vector $\overline{\mathbf{U}} \in \mathbb{R}^{d+n}$ can be defined along with $\overline{\mathbf{U}}_i = \mathbf{U}_i$ for $i = 1, \cdots, d$ and $\overline{\mathbf{U}}_{d+j} = \frac{\epsilon_j}{\Delta}$ for $j = 1, \cdots, n$. Under these conditions, it can be verified that:

1.
$$\forall i,\ \forall (z_1,\cdots,z_k) \in \overline{c}^i\ \ \overline{\mathbf{U}}.\sum_{y\in T(i)} \phi(x^{(i)},y) - \overline{\mathbf{U}}.\sum_{z\in B(i)} \phi(x^{(i)},z) \geq \gamma$$

2.
$$\forall i,\ \forall (z_1,\cdots,z_k) \in \overline{c}^i\ \ \| \sum_{y\in T(i)} \phi(x^{(i)},y) - \sum_{z\in B(i)} \phi(x^{(i)},z)\|^2 \leq R^2+\Delta^2$$

3.
$$\|\overline{U}\|^2 = \|U\|^2 + \sum_i \frac{\epsilon_i^2}{\Delta^2} = 1 + \frac{D_{U,\gamma}^2}{\Delta^2}$$

It can be observed that the vector $\frac{\overline{U}}{\|\overline{U}\|}$ is able to separate the data with the margin $\frac{\gamma}{\sqrt{1+\frac{D_{U,\gamma}^2}{\Delta^2}}}$. From Theorem 1, it can be concluded that the first pass of the algorithm *SiLA* with representation $\overline{\phi}$ makes $k_{max}(\Delta) = \frac{1}{\gamma^2}(R^2+\Delta^2)(1+\frac{D_{U,\gamma}^2}{\Delta^2})$ mistakes in the worst case. However, it can be further noticed that the first pass of the original algorihtm *SiLA* with representation ϕ is similar to the first pass of *SiLA* along with the new representation $\overline{\phi}$, since the parameter weights for the additional features $\overline{\phi}_{d+j}$ for $j = 1,\cdots,n$ each affect a single example of training data, and do not affect the classification phase of the test data. Thus the original algorithm *SiLA* also makes $k_{max}(\Delta)$ mistakes in the worst case scenario during the first pass over the training set of examples. Finally, $k_{max}(\Delta)$ can be minimized with respect to Δ, thus giving $\Delta = \sqrt{RD_{U,\gamma}}$ and hence $k_{max}(\sqrt{RD_{U,\gamma}}) = \frac{(R^2+D_{U,\gamma}^2)}{\gamma^2}$, implying the bound in the theorem. □

A.3 Theorem 4 - *gCosLA*

Let $(x_1,x_1',y_1),\cdots(x_\tau,x_\tau',y_\tau),\cdots,(x_N,x_N',y_N)$ *be a sequence of* N *examples. For any positive, semi-definite matrix* A*, let for each* τ*,* $1 \leq \tau \leq N$*:*

$$R_{-1}(x_\tau,x_\tau',A) = [min((x_\tau^t A x_\tau),(x_\tau'^t A x_\tau'))]^{-1}$$

and

$$R_{+1}(x_\tau,x_\tau',A) = [max((x_\tau^t A x_\tau),(x_\tau'^t A x_\tau'))]^{-1}$$

Assume that there exists a positive, semi-definite matrix A^**, a threshold* b^* *and a positive real number* γ *such that:*

$$(R_{+1}x_\tau^t A^* x_\tau' - b^*) \geq \gamma \wedge (b^* - R_{-1}x_\tau^t A^* x_\tau') \geq \gamma$$

Using the notations introduced previously, let $R \in \mathbb{R}^+$ *be an upper bound such that:*

$$\frac{1}{\|x_\tau x_\tau'^t\|_2 + 1} R_{y_\tau}^2 \|x_\tau\|_2^4 \|x_\tau'\|_2^4 \leq R,\ y_\tau \in \{-1,+1\}$$

Then the following bound holds for any $M \geq 1$*:*

$$\sum_{\tau=1}^{M} (l_\tau(A,b))^2 \leq R\left(\|A^*-I\|_2^2 + (b^*)^2\right)$$

Proof:
Let $\Delta_\tau = \|(A_\tau, b_\tau) - (A^*, b^*)\|_2^2 - \|(A_{\tau+1}, b_{\tau+1}) - (A^*, b^*)\|_2^2$. Then:

$$\sum_{\tau=1}^{T} \Delta_\tau = \|(A_1, b_1) - (A^*, b^*)\|_2^2 - \|(A_{\tau+1}, b_{\tau+1}) - (A^*, b^*)\|_2^2 \quad (A.3)$$

$$\leq \|(A_1, b_1) - (A^*, b^*)\|_2^2 \quad (A.4)$$

and

$$\Delta_\tau = \left(\|(A_\tau, b_\tau) - (A^*, b^*)\|_2^2 - \|(\hat{A}_\tau, \hat{b}_\tau) - (A^*, b^*)\|_2^2\right) \quad (A.5)$$

$$+\left(\|(\hat{A}_\tau, \hat{b}_\tau) - (A^*, b^*)\|_2^2 - \|(A_{\tau+1}, b_{\tau+1}) - (A^*, b^*)\|_2^2)\right) \quad (A.6)$$

By assumption, $(A^*, b^*) \in C_\tau'^{y_\tau}$ and $(\hat{A}_\tau, \hat{b}_\tau) \in C_\tau'^{y_\tau}$. $(\hat{A}_\tau, \hat{b}_\tau)$ is the projection of (A_τ, b_τ) on $C_\tau'^{y_\tau}$. So, using equation A.6, Δ_τ can be written as:

$$\Delta_\tau \geq \|(A_\tau, b_\tau) - (\hat{A}_\tau, \hat{b}_\tau)\|_2^2 + \left(\|(\hat{A}_\tau, \hat{b}_\tau) - (A^*, b^*)\|_2^2 - \|(A_{\tau+1}, b_{\tau+1}) - (A^*, b^*)\|_2^2\right)$$

Furthermore, and again by assumption, $(A^*, b^*) \in C_a$ and $(A_{\tau+1}, b_{\tau+1}) \in C_a$. So, one again using equation A.6, Δ_τ can be expressed as:

$$\Delta_\tau \geq \|(A_\tau, b_\tau) - (\hat{A}_\tau, \hat{b}_\tau)\|_2^2 + \|(\hat{A}_\tau, \hat{b}_\tau) - (A_{\tau+1}, b_{\tau+1})\|_2^2 \geq \|(A_\tau, b_\tau) - (\hat{A}_\tau, \hat{b}_\tau)\|_2^2$$

By definition:

$$l_\tau(A, b) = \max\{0, y_\tau(b - \frac{x_\tau^t A x_\tau'}{\sqrt{x_\tau^t A x_\tau}\sqrt{x_\tau'^t A x_\tau'}}) + \gamma\},$$

and

$$\hat{A}_\tau = A_\tau + y_\tau a(x_\tau, x_\tau'^t)\ ,\ a = \frac{\gamma - y_\tau R_{y_\tau}(x_\tau^t A_\tau x_\tau') + y_\tau b}{R_{y_\tau}(\|x_\tau\|^2 \|x_\tau'\|^2)}$$

In case $y_\tau = +1$, $\hat{A}_\tau = A_\tau + ya(x_\tau, x_\tau'^t)$, $\hat{b}_\tau = b_\tau + a$ Thus, a can be rewritten as: $a = \frac{\gamma - R_{+1}(x_\tau^t A_\tau x_\tau') + b}{R_{+1}(\|x_\tau\|^2 \|x_\tau'\|^2)}$ and:

$$l_\tau(A_\tau, b_\tau) = \gamma + b_\tau - \frac{x_\tau^t A_\tau x_\tau'}{\sqrt{x_\tau^t A_\tau x_\tau}\sqrt{x_\tau'^t A_\tau x_\tau'}}\ ,\ \|(A_\tau, b_\tau) - (\hat{A}_\tau, \hat{b}_\tau)\|_2^2 = a^2(\|x_\tau x_\tau'^t\|_2^2 + 1)$$

But it is already known that:

$$R_{+1} x_\tau^t A_\tau x_\tau' \leq \frac{x_\tau^t A_\tau x_\tau'}{\sqrt{x_\tau^t A_\tau x_\tau}\sqrt{x_\tau'^t A_\tau x_\tau'}}$$

So:

$$\gamma - R_{+1} x_\tau^t A_\tau x_\tau' + b_\tau \geq l_\tau(A_\tau, b_\tau)$$

Hence:

$$\|(A_\tau, b_\tau) - (\hat{A}_\tau, \hat{b}_\tau)\|_2^2 \geq \frac{(l_\tau(A_\tau, b_\tau))^2}{(R_{+1}\|x_\tau\|_2^2\|x'_\tau\|_2^2)^2}(\|x_\tau x_\tau'^t\|_2^2 + 1)$$

and

$$\|(A_\tau, b_\tau) - (\hat{A}_\tau, \hat{b}_\tau)\|_2^2 \geq \frac{(l_\tau(A_\tau, b_\tau))^2}{(R_{+1}^2\|x_\tau\|_2^4\|x'_\tau\|_2^4)^2}(\|x_\tau x_\tau'^t\|_2^2 + 1)$$

As $\frac{R_{+1}^2\|x_\tau\|_2^4\|x'_\tau\|_2^4}{\|x_\tau x_\tau'^t\|+1} \leq R$, combining the above results leads to the desired bound for $y_\tau = +1$. The case $y_\tau = -1$ is treated in a similar way. □

Appendix B

French Translation

B.1 Introduction

Les algorithmes d'apprentissage automatique améliorent automatiquement leur performance P mesurée à travers une expérience E sur une tâche T. Par exemple, on peut considérer le problème de la conception d'un système qui apprend à jouer aux dames. Dans ce cas, la tâche T est de jouer aux dames, la mesure de performance P est le pourcentage de jeux gagnés dans un tournoi mondial et E est l'occasion de jouer contre soi-même.

L'apprentissage automatique a récemment émergé comme l'un des domaines clés de l'intelligence artificielle. L'une des principales raisons de sa popularité réside dans le désir passionné de l'homme à explorer et à reproduire le processus de l'apprentissage humain. L'apprentissage automatique peut être considéré comme une double tâche; consistant d'une part à apprendre les propriétés invariantes et communes d'un ensemble d'échantillons qui caractérisent une classe, et d'autre part de décider qu'un nouvel échantillon est un membre possible de la classe en vérifiant s'il a des propriétés communes à celleapprises de l'ensemble d'échantillons.

Les algorithmes d'apprentissage automatique peuvent être classés dans trois catégories différentes : l'apprentissage supervisé où l'apprentissage est basé sur un ensemble de données étiquetées, l'apprentissage non-supervisé, qui ne nécessite aucun type d'intervention humaine (il est généralement utilisé lorsque les classes ne sont pas connues à l'avance), et l'apprentissage semi-supervisé qui se situe entre les approches supervisées et non-supervisées.

L'apprentissage automatique a été utilisé dans divers milieux différents tels que la classification (par exemple la reconnaissance des chiffres manuscrits [63], classification des documents [55], reconnaissance des visages [105] etc.), le clustering (k-means clustering [11], la classification spectrale [115]), le bio-informatique, la finance, les systèmes de filtrage de l'information qui apprennent automatiquement les intérêts des utilisateurs, la détection des fumées dangereuses sur des installations industrielles [39] etc. Il est basée sur l'apprentissage à partir des données, et donc étroitement liée au domaine de la fouilles de données. Ce domaine se base sur l'extraction des modèles utiles à partir des données brutes.

Chaque algorithme d'apprentissage automatique travaille avec un ensemble d'exemples. Dnas cet ensemble, quelques exemples sont utilisés pour apprendre les caractéristiques sous-jacentes des données à partir d'un ensemble de traits. Ce sous-ensemble est appelé ensemble d'apprentissage. Afin de valider un algorithme, il est exécuté sur des nouveaux exemples constituant un ensemble

de test. Un ensemble de validation peut éventuellement être utilisé pour optimiser les différents paramètres de l'algorithme.

B.1.1 Motivation

Considérons deux objets à comparer, par exemple deux documents ou des images. Afin de faire cette comparaison, une similarité ou une distance peut être calculée entre ces deux objets. La plupart du temps, des mesures par défaut sont utilisées, c'est-à-dire la distance euclidienne dans le cas des images et la similarité cosinus pour la classification de texte. Ces mesures par défaut considèrent que la métrique entre les différents objets est paramétrée par une matrice d'identité. En d'autres termes, des mesures comme la distance euclidienne et la similarité cosinus considèrent une géométrie très simple de l'espace dans lequel les données se trouvent. De nombreux travaux ont démontré qu'il est beaucoup mieux d'apprendre une métrique à partir des données plutôt que de supposer une métrique simple comme la distance euclidienne ou la similarité cosinus.

La récente popularité d'Internet a conduit à une énorme augmentation de la quantité d'informations, et à un élargissement des domaines de recherche consacrés à l'organisation automatique de ces informations. Depuis 2000, un forum d'évaluation nommé Cross Language Evaluation Forum (CLEF) est organisé chaque année. Le but est d'évaluer les systèmes de recherche d'information utilisant les langues européennes dans les contextes monolingues ainsi qu'inter-langues. Une campagne pour le filtrage des informations (INFILE) a été menée comme une piste pilote de CLEF en 2008 et 2009. L'objectif d'INFILE était de filtrer un flux continu de documents de différents thèmes prédéfinis. Dans le cas du filtrage de l'information, les seuils basés sur le cosinus pourraient être appris sur la base des flux entrant de documents, à condition q'une sorte de supervision existe. C'est le domaine de l'*apprentissage de métriques* [53, 54]. La figure B.1 indique les cinq premières images classées par *OASIS* [16] (un algorithme d'apprentissage des distances sur les images [19]) sur quatre exemples de requêtes-images dans un ensemble de données de Google. Les requêtes texte pertinentes pour chaque image sont notées sous l'image. La ligne la plus haute montre une requête-image, retrouvé à l'origine comme réponse à la requête textuelle *illusion*. Nous remarquons que tous les cinq images hautement classées par *OASIS* sont sémantiquement liées, représentant d'autres types d'illusions visuelles. Les trois autres exemples montrent que *OASIS* a pu capturer la sémantique des photos d'animaux (chiens et chats), des montagnes et des différents produits alimentaires.

L'objectif principal de l'apprentissage de métriques est d'apprendre une métrique adaptée au problème consideré. Les algorithmes de classification et le regroupement de données dépendent fortement de la présence d'une bonne mesure. En dehors de ces domaines, l'apprentissage de métriques est un élément très important dans les problèmes comme la reconnaissance des visages, la reconnaissance d'objets visuels, la reconnaissance automatique de la parole [107], la similarité de la musique, l'estimation de la pose, la similarité et la recherche d'images [59] etc. Beaucoup d'algorithmes d'apprentissage de métriques se divisent en deux types différents: apprentissage de distance et apprentissage de similarité.

La plupart des travaux relatifs à l'apprentissage se concentrent uniquement sur l'apprentissage de distance et essayent d'apprendre la métrique sous-jacente à la distance de Mahalanobis. Toute-

[19] Dans ce travail, aucune distinction n'est faite entre la distance et la similarité.

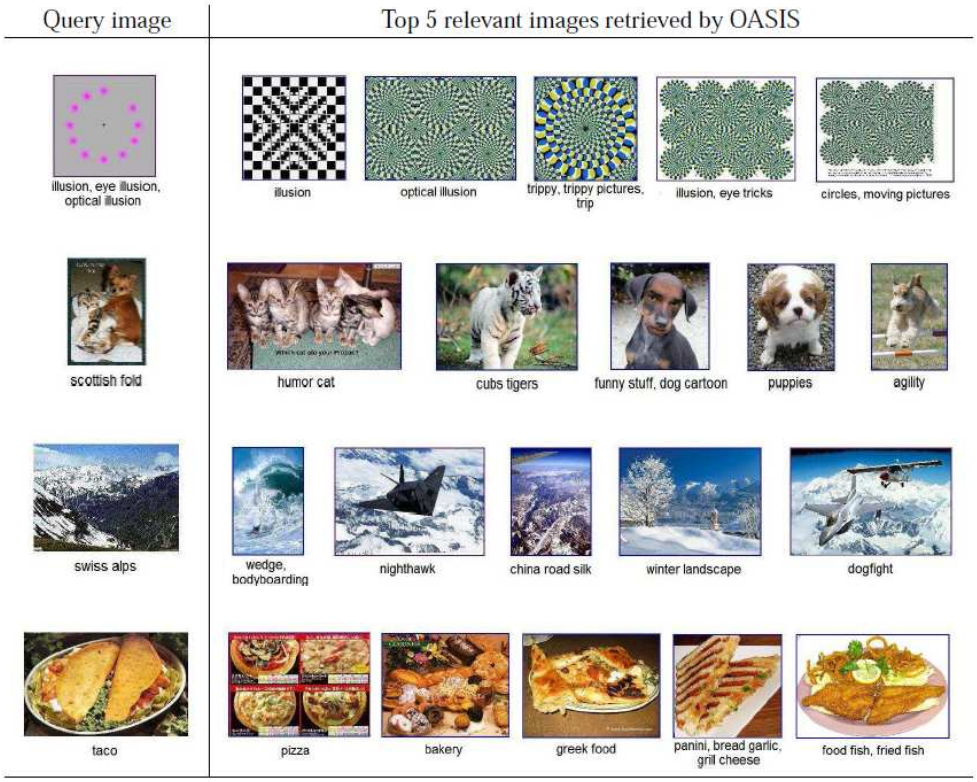

Figure B.1: OASIS: Un algorithme d'apprentissage de la métrique de la distance pour trouver les images similaires [16]

fois, dans de nombreuses situations pratiques, il est préférable d'utiliser des similarités et non des distances. C'est typiquement le cas quand on travaille sur des textes, pour lesquels la mesure du cosinus a été jugée plus appropriée que la distance euclidienne ou celle de Mahalanobis. En outre, plusieurs expériences montrent que l'utilisation de la similarité cosinus doit être préférée à la distance euclidienne sur plusieurs collections non textuelles (voir par exemple [18, 72, 84, 87]). Le fait de pouvoir apprendre de manière efficace des mesures de similarité appropriées, par opposition aux distances, par exemple dans le cadre de la classification à k plus proches voisins *kPPV*, à une grande importance pour différentes collections. Si plusieurs travaux ont partiellement résolu ce problème (comme par exemple [1, 46, 52]) pour différentes applications, nous ne connaissons aucun travail antérieur qui a pleinement traité le cas de l'apprentissage des métriques de similarité pour la classification *kPPV*. C'est la motivation principale de ce travail. Dans une première étape, un algorithme d'apprentissage d'une métrique de similarité sans contrainte est développé. Dans ce cas, la normalisation est complètement indépendante de la matrice de similarité. Les preuves montrent que l'erreur de généralisation est limitée, et donc que l'algorithme a des bonnes propriétés de généralisation. Ensuite, nous avons développé un algorithme basé sur le cosinus généralisé ayant une normalisation dépendant de la matrice de similarité. En outre, l'apprentissage de similarité sans contrainte est comparée à la famille d'algorithmes *RELIEF*. Bien que *RELIEF* soit fondamentalement un algorithme de re-pondération, il a été prouvé récemment par Sun et Wu [102] qu'il s'agit d'un algorithme d'apprentissage de métrique de distance qui permet d'optimiser une approximation de la perte 0-1. Nous montrons ici que cette approximation est trop permissive, et nous proposons un autre approximation stricte et mieux adaptée à la classification.

B.1.2 Plan de la thèse

- Nous décrivons dans le chapitre 2 les notions de base liées à l'apprentissage automatique et nous passons en revue diverses techniques de l'état de l'art pour l'apprentissage des métriques. Les deux principaux types d'apprentissage automatique (supervisé ou non supervisé) sont examinés en détail. De plus, nous introduisons les bases de l'apprentissage en ligne et par lots. Certains des principaux algorithmes d'apprentissage de distance, par exemple la classification par les plus proches voisins avec une vaste marge [112], comme les approches fondées sur la théorie de l'information [28] et POLA [99], sont discutés et comparés. *RELIEF*, un algorithme de pondération des attributs, est également présenté avec son interprétation mathématique. Les paramètres d'évaluation et les techniques de comparaison des classifieurs sont finalement discutés.

- Dans le chapitre 3, nous montrons comment on peut apprendre efficacement des seuils basés sur le cosinus lorsqu'on a très peu ou pas du tout de supervision. Cette technique est établie pour une tâche de filtrage, où un ensemble de documents est filtré en fonction des profils utilisateurs. Les algorithmes en ligne ainsi que par lots sont discutés et une comparaison poussée est menée. Les algorithmes sont développés dans le cadre de la campagne INFILE de la compétition CLEF.

- Le chapitre 4 commence par la description d'une méthode d'apprentissage de similarité, appelée *SILA*, où la normalisation est indépendante de la similarité apprise. *SILA* est comparé

à l'algorithme *RELIEF* pour lequel Sun et Wu [102] ont montré qu'il apprend essentiellement une mesure de distance, tout en optimisant une fonction de coût se rapprochant de la perte 0-1. Nous montrons que l'approximation utilisée par *RELIEF* est lâche, et nous proposons une version plus stricte en utilisant une fonction de coût plus proche de la perte 0-1. Cette version plus stricte conduit à une nouvelle et meilleure version de *RELIEF*.

En outre, un algorithme d'apprentissage de similarité du type cosinus généralisé (*gCosLA*) est élaboré, dans ce cas, la normalisation dépend de la matrice de similarité.

- Les différentes algorithmes d'apprentissage de similarité développés au cours de cette thèse sont évalués au chapitre 5. Afin d'évaluer si les résultats sont significativement différents ou non, un s-test est utilisé. Nous montrons que la similarité est une alternative meilleure que la distance sur différents jeux de données. De plus, les algorithmes d'apprentissage de similarité sans contraintes, ainsi que ceux de similarité généralisée sont comparés avec des autres algorithmes de classification. Les algorithmes d'apprentissage de similarité sont plus performants que leurs homologues sur certaines bases de données *UCI*.

- Le chapitre 6 conclut cette thèse avec les limitations des approches proposées et les perspectives d'avenir.

- Enfin, les preuves de convergence, et de bon comportement pour *SILA* et *gCosLA* sont fournies dans l'annex A.

B.2 Conclusion

L'apprentissage automatique concerne l'étude des algorithmes capables d'améliorer automatiquement leurs performances par l'expérience. Les différentes bases de données ayant des propriétés différentes, il peut être utile d'apprendre la géométrie sous-jacente des données à traiter. En effet, récemment, de nombreux travaux tels que Weinberger et al. [112], Jain et al. [53], ont montré que l'apprentissage d'une métrique, basée sur la géométrie de l'espace contenant les données, est souvent une meilleure idée que de supposer la présence d'une structure géométrique simple. Cependant, la plupart des travaux dans le domaine de l'apprentissage de métriques ne considèrent que l'apprentissage de distances et ne s'intéressent pas aux similarités, entre autres Goldberger et al. [42], Xing et al. [114], Davis et al. [28], Globerson et al. [41]. Traditionnellement, la mesure de similarité du cosinus a montré de bons résultats pour les jeux de données textuelles [95]. De plus, certains travaux récents comme Qamar et al. [87], Peterson et al. [84] ont montré que la similarité du cosinus devrait également être préférée aux mesures de distance sur les jeux de données non textuels.

B.2.1 Les principales contributions

Nous nous sommes concentrés ici sur l'apprentissage de similarités (complètes) à partir de données en vue d'une tâche de classification par k plus proches voisins (kNN). Nous considérons différents scénarios, certains s'appuyant sur peu de données étiquetées, d'autres utilisant des ensembles de données entièrement annotés. Dans les situations où seule une petite quantité d'annotations est disponible, on ne peut pas apprendre des structures complexes, et nous nous

sommes limités à l'apprentissage de quelques méta-paramètres de contrôle à base de similarités basées sur le cosinus. Ces travaux ont été judicieusement utilisé dans le contexte des pistes INFILE, pendant les campagnes d'évaluation CLEF 2008 et CLEF 2009. Dans les situations avec plus d'annotations, nous avons examiné deux généralisations possibles des mesures de similarités existantes. Ces deux généralisations se distinguent principalement par les contraintes sur lesquelles elles reposent. La première n'impose presque aucune contrainte sur la transformation à utiliser, en particulier, les normalisations considérées ne dépendent pas de la métrique apprise, ce qui rend le processus d'apprentissage plus simple. La seconde, quant à elle, impose de fortes contraintes sur la métrique apprise, en particulier, elle doit correspondre à une mesure de cosinus dans un espace intégré. Ainsi, elle doit s'appuyer sur des matrices semi-définies positives, avec une normalisation dépendante de la métrique apprise. Si la première généralisation a été basée sur la famille du perceptron, la seconde nécessite une approche différente. Dans les deux cas, nous avons fourni des preuves théoriques du comportement correct de nos algorithmes.

L'apprentissage d'une métrique implique de modéliser les dépendances entre les caractéristiques, et de les pondérer convenablement. Cet objectif est réalisé par des procédures de repondération des caractéristiques, et plusieurs travaux récents ont souligné les liens entre ces procédures (comme *RELIEF*) et l'apprentissage supervisé de métriques. Nous avons étudié ce lien ici en détail, et nous avons montré que la fonction objectif approchée par la famille de procédures *RELIEF* n'est pas optimale. Nous avons ensuite soigneusement évalué nos algorithmes, essayant d'évaluer les cas où ils apportent une amélioration significative dans les résultats. De plus, nous avons comparé leurs performances avec celles d'autres approches. Il est toujours difficile de comparer deux approches qui sont de nature très différente. Nous croyons que la comparaison que nous avons effectuée indique que les méthodes d'apprentissage de similarités, ainsi que les algorithmes que nous avons proposés à cet effet, sont de précieux outils d'apprentissage automatique, pouvant compléter les outils d'apprentissage de distances. Nous allons maintenant présenter un résumé des principales contributions de notre travail.

1. Une étude approfondie des algorithmes d'apprentissage de métriques, y compris des algorithmes d'apprentissage de distances et de similarité est effectuée.

2. Une méthode de filtrage de l'information capable d'apprendre des seuils spécifiques pour les catégories basés sur la mesure du cosinus, tant qu'une forme de supervision est présente, a été développée. Des algorithmes offline et online ont été mis au point pour le processus de filtrage d'information. Les deux méthodes ont été en mesure d'obtenir le meilleur F-score de la campagne CLEF INFILE des années 2008 et 2009.

3. Nous avons montré que la similarité du cosinus donnait de meilleurs résultats que la distance euclidienne sur de nombreux jeux de données.

4. Un algorithme d'apprentissage de similarité non-contraintes appelé *SILA* a été développé pour apprendre les mesures de similarité pour la tâche de classification par k plus proches voisins. La normalisation, dans *SILA*, est totalement indépendante de la matrice de similarité, ce qui permet d'apprendre différents types de fonctions de similarité basées sur des matrices diagonales, symétriques ou asymétriques. Des preuves de convergence et de

généralisation des algorithmes développés ont de plus été fournies. Un test statistique, le *s-test*, a été utilisé pour analyser statistiquement l'ensemble des résultats.

5. Les liens entre *RELIEF* et *SILA* ont été étudiés. Nous avons ensuite développé un algorithme d'apprentissage (*RBS*) basé sur *RELIEF*. Cependant il s'est avéré que *RBS* n'a pas donné de résultats satisfaisants. La raison principale est que *RBS* essaie d'optimiser une fonction de coût se rapprochant de la perte 0-1 à la manière de *RELIEF*. Nous avons montré que cette approximation est imprécise, et nous avons proposé une version plus stricte de *RBS*, appelée *sRBS*, basée sur une fonction de coût plus proche de la perte 0-1. *sRBS* a obtenu des résultats significativement meilleurs que les autres algorithmes basés sur *RELIEF*, confirmant en particulier que la perte 0-1 est une fonction de coût plus appropriée que celle utilisée implicitement par *RELIEF*.

6. Finalement, un algorithme basé sur la similarité de cosinus généralisée a été développé. La normalisation dans le cas de *gCosLA* était dépendante de la matrice de similarité et celle-ci appartenait à la classe des matrices semi-définies positives. Les résultats ont montré que *gCosLA* était significativement meilleure que *SILA* sur de nombreuses collections de données considérées.

7. *SILA* et *gCosLA* ont été comparés à de nombreux algorithmes d'apprentissage de métriques de l'état de l'art et ont montré de très bons résultats dans les situations où les similarités sont utiles. Comme tels, ils constituent de nouveaux outils d'apprentissage automatique, pouvant judicieusement compléter les algorithmes d'apprentissage de métriques existants.

Après avoir examiné les principales contributions de notre thèse, nous nous tournons vers les limites de notre travail, et les perspectives qu'il ouvre.

B.2.2 Limites et perspectives

Comme avec n'importe quel algorithme d'apprentissage automatique, les algorithmes d'apprentissage de similarité ont leurs propres limites. Le processus d'apprentissage de seuils ne fonctionne pas comme celui d'apprentissage de métriques car il ne tient pas compte de la géométrie de l'espace contenant les données. Bien que *SILA* ait été utilisé avec une grande base de données de *Newsgroup*, il reste à montrer comment il pourrait être adapter à des jeux de données de grande dimension. La complexité de *SILA* est quadratique dans le nombre de dimensions. Bien que très prometteur, *gCosLA* est un peu lent en raison de sa complexité cubique en terme du nombre de dimensions. Comme *gCosLA* apprend une matrice semi-définie positive (PSD) en utilisant la décomposition de la matrice en valeurs propres, sa complexité peut être réduite en utilisant des méthodes d'approximation des valeurs propres. Par exemple, l'algorithme de Lanczos et ses variantes spécialisées devraient être étudiés. Il y a également une autre manière envisageable pour réduire la complexité de *gCosLA*, en utilisant le fait que toute matrice M semi-définie positive peut être décomposée en U^tU, où U est une matrice de rang inférieur. Dans ce cas, la contrainte que la matrice soit semi-définie n'a plus besoin d'être vérifiée, conduisant à un algorithme plus rapide (cette astuce est déjà utilisée dans [47] dans le contexte de l'apprentissage de distances). Cependant, bien que plus rapide, le problème de l'apprentissage de U n'est pas

nécessairement plus simple, en raison des optimum locaux. Il n'est donc pas clair si cette stratégie serait bénéfique pour *gCosLA*, et une étude plus approfondie est nécessaire ici.

Relatif à la question de la vitesse d'exécution, mais avec des implications supplémentaires, est le manque de contrôle de l'*agressivité* des règles sous-jacentes de mise à jour des algorithmes que nous avons présenté (en particulier *SILA*). Un des points forts de la famille passif-agressif réside justement dans un tel contrôle qui pourrait être ajouté dans notre cas aussi. Ceci étant dit, le réglage des méta-paramètres n'est pas toujours une tâche facile, et peut conduire à des calculs supplémentaires. On peut cependant espérer qu'une solution valable serait atteinte plus rapidement, et donc nécessitant moins de mises à jour et conduisant à un processus d'apprentissage globalement plus rapide. En raison des conséquences possibles théoriques et pratiques qu'ils pourraient avoir, nous pensons qu'il serait intéressant d'étudier dans un proche avenir l'utilisation de paramètres d'agressivité dans nos algorithmes.

Une autre limitation de notre travail réside dans le fait qu'une seule matrice de similarité est apprise (en recourant à la classification binaire et à la règle standard d'un contre-le-reste, plusieurs matrices sont en pratique apprises pour résoudre un problème de catégorisation multi-classes ; mais toutes les matrices sont globales dans le sens où elles ne sont pas adaptées à des régions spécifiques de l'espace). Une autre possibilité est d'apprendre différentes mesures de similarités locales dans différentes parties de l'espace d'entrée comme c'est le cas pour l'algorithme de Multi-Metric *LMNN* [112]. Une possibilité avec l'approche que nous avons suivie serait de considérer les régions de voisinage autour de chaque point et tous les exemples qu'elles contiennent, puis d'apprendre les matrices pour chacune de ces régions. La classification d'un point impliquerait alors uniquement les régions contenant le point. Si cette approche semble simple et prometteuse, elle impliquerait plus de calcul. Il faudrait donc développer des versions plus rapides des algorithmes que nous avons présentés.

Enfin, une autre perspective que nous aimerions explorer est l'utilisation de l'algorithme *SILA* dans un contexte différent, à savoir celui de la recherche d'information (RI), car ce domaine s'appuie fortement sur la mesure du cosinus. Dans la recherche d'information, la similarité est calculée entre une requête q et un document d. Une application possible de *SILA* dans ce cas, pourrait aller dans le sens suivant : la requête q pourrait remplacer $x^{(i)}$, répétée $N1$ (nombre de documents récupérés et jugés pertinents par l'utilisateur) fois ; le voisin objectif y pourrait alors être choisi arbitrairement, ou selon la mesure similarité cosinus standard, parmi l'ensemble des documents pertinents, alors que z représenterait le plus proche document non pertinent. Comme mentionné ci-dessus, la matrice A pourraient être apprise à l'aide des jugements de pertinence existants, ou par les commentaires des utilisateurs. Nous avons l'intention d'étudier ces différentes possibilités dans un avenir proche.

Bibliography

[1] J.-P. Bao, J.-Y. Shen, X.-D. Liu, and H.-Y. Liu. Quick asymmetric text similarity measures. *Proceedings of International Conference on Machine Learning and Cybernetics*, 2003.

[2] L. Baoli, L. Qin, and Y. Shiwen. An adaptive k-nearest neighbor text categorization strategy. *ACM Transactions on Asian Language Information Processing (TALIP)*, 3(4):215–226, 2004.

[3] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, 6:937–965, 2005.

[4] S. D. Bay. Nearest neighbor classification from multiple feature subsets. *Intelligent Data Analysis*, 3:191–209, 1999.

[5] A. E. Bernal, K. Hospevian, T. Karadeniz, and J. L. Lassez. Similarity based classification. In M. R. Berthold, H. J. Lenz, E. Bradley, R. Kruse, and C. Borgelt, editors, *Proceedings of 5th International Symposium on Intelligent Data Analysis (IDA)*, volume 2811/2003 of *Lecture Notes in Computer Science*, pages 187–197. Springer, 2003.

[6] R. Besançon, S. Chaudiron, D. Mostefa, O. Hamon, I. Timimi, and K. Choukri. Overview of CLEF 2008 INFILE Pilot Track. In *Working Notes of the Cross Language Evaluation Forum (CLEF)*, Aarhus, Denmark, 17–19 Sept 2008.

[7] R. Besançon, S. Chaudiron, D. Mostefa, I. Timimi, and K. Choukri. The INFILE Project: a Crosslingual Filtering Systems Evaluation Campaign. In ELRA, editor, *Proceedings of LREC'08*, Morocco, May 2008.

[8] R. Besançon, S. Chaudiron, D. Mostefa, I. Timimi, K. Choukri, and M. Laïb. Information filtering evaluation: Overview of clef 2009 infile track. In C. Peters, G. Di Nunzio, M. Kurimo, D. Mostefa, A. Penas, and G. Roda, editors, *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, volume 6241 of *Lecture Notes in Computer Science*, pages 342–353. Springer Berlin / Heidelberg, 2011.

[9] R. Besançon, S. Claudiron, D. Mostefa, I. Timimi, K. Choukri, and M. Laïb. Overview of CLEF 2009 INFILE track. In *Working Notes of the Cross Language Evaluation Forum (CLEF)*, Corfu, Greece, 30 Sept–02 Oct 2009.

[10] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *International Conference on Machine Learning (ICML)*, pages 81–88, 2004.

[11] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1st edition, January 1996.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing edition, October 2007.

[13] H. D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34(1):123–135, Jan 1962.

[14] V. Bodinier, A. M. Qamar, and E. Gaussier. Working notes for the InFile campaign : Online document filtering using 1 nearest neighbor. In *Working Notes of the Cross Language Evaluation Forum (CLEF)*, Aarhus, Denmark, 17–19 Sept 2008.

[15] V. Bodinier, A. M. Qamar, and E. Gaussier. Online document filtering using adaptive k-nn. In C. Peters, T. Deselaers, N. Ferro, J. Gonzalo, G. Jones, M. Kurimo, T. Mandl, A. Peñas, and V. Petras, editors, *Evaluating Systems for Multilingual and Multimodal Information Access*, volume 5706 of *Lecture Notes in Computer Science*, pages 947–950. Springer Berlin / Heidelberg, 2009.

[16] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11:1109–1135, 2010.

[17] L.-B. Chen, Y.-N. Wang, and B.-G. Hu. Kernel-based similarity learning. In *Proceedings of International Conference on Machine Learning and Cybernetics*, volume 4, pages 2152–2156 vol.4, Nov. 2002.

[18] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti. Similarity-based classification: Concepts and algorithms. *Journal of Machine Learning Research*, 10:747–776, March 2009.

[19] V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley-Interscience, 1998.

[20] M. Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 1–8, Morristown, NJ, USA, 2002.

[21] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[22] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman and Hall, 1994.

[23] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[24] K. Crammer, M. Dredze, and F. Pereira. Exact convex confidence-weighted learning. In *Proceedings of Advances in Neural Information Processing Systems (NIPS) 21*, pages 345–352, 2009.

[25] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2002.

[26] N. Cristianini. Tutorial: Support vector and kernel machines. In *International Conference on Machine Learning (ICML)*, Williams College, Williamstown, MA, USA, 2001.

[27] J. V. Davis and I. S. Dhillon. Structured metric learning for high dimensional problems. In *KDD '08: Proceedings of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 195–203, New York, NY, USA, 2008. ACM.

[28] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

[29] O. Dekel. From online to batch learning with cutoff-averaging. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Proceedings of Advances in Neural Information Processing Systems (NIPS), 21*, pages 377–384. MIT Press, 2009.

[30] O. Dekel and Y. Singer. Data-driven online to batch conversions. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2005.

[31] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition (Stochastic Modelling and Applied Probability)*. Springer, corrected edition, April 1996.

[32] M. Diligenti, M. Maggini, and L. Rigutini. Learning similarities for text documents using neural networks. In *Proceedings of International Association for Pattern Recognition (IAPR) – TC3 International Workshop on Artificial Neural Networks in Pattern Recognition (ANNPR)*, 2003.

[33] M. Dredze and K. Crammer. Confidence-weighted linear classification. In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, pages 264–271. ACM, 2008.

[34] E. Fix and J. L. Hodges. Discriminatory analysis: Nonparametric discrimination: Consistency properties, Tech. Rep. 4, Usaf School of Aviation Medicine, Randolf Field, Texas, 1951.

[35] R. François and F. Langrognet. Double cross validation for model based classification. In *Proceedings of The R User Conference (useR'2006)*, Vienna, Austria, June 2006.

[36] A. Frank and A. Asuncion. UCI machine learning repository, University of California, Irvine, School of Information and Computer Sciences, 2010.

[37] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[38] A. Frome, F. Sha, Y. Singer, and J. Malik. Learning globally-consistent local distance functions for shape-based image retrieval and classification. In *Proceedings of Eleventh IEEE International Conference on Computer Vision (ICCV)*, 2007.

[39] D. Gacquer. *Sur l'utilisation active de la diversité dans la construction d'ensembles de classifieurs. Application à la détection de fumees nocives sur site industriel.* Phd thesis, Université de Valenciennes et du Hainaut-Cambresis, 2008.

[40] S. I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.

[41] A. Globerson and S. Roweis. Metric learning by collapsing classes. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2005.

[42] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighborhood component analysis. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems (NIPS), 17*, pages 513–520. MIT Press, 2005.

[43] R. L. Goldstone and A. Kersten. *Comprehensive handbook of Psycology*, volume 4. Wiley, New Jersey, 2003.

[44] G. Golub and C. V. Loan. *Matrix Computations (2nd ed.)*. John Hopkins University Press, 1989.

[45] R. C. Gonzalez and R. E. Woods. *Digital Image Processing.* Addison-Wesley, 1992.

[46] M. Grabowski and A. Szałas. A technique for learning similarities on complex structures with applications to extracting ontologies. In J. Szczepaniak, P. S. Kacprzyk and A. Niewiadomski, editors, *Proceedings of the 3rd Atlantic Web Intelligence Conference, AWIC*, LNCS. Springer Verlag, 2005.

[47] M. Guillaumin, J. Verbeek, and C. Schmid. Multiple instance metric learning from automatically labeled bags of faces. In *European Conference on Computer Vision*, sep 2010.

[48] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 18(6):607–616, 1996.

[49] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning.* Springer, July 2003.

[50] D. P. Helmbold, J. Kivinen, and M. K. Warmuth. Relative loss bounds for single neurons. *IEEE Transactions on Neural Networks*, 10:1291–1304, 1996.

[51] D. P. Helmbold and M. K. Warmuth. On weak learning. *Journal of Computer and System Sciences*, 50(3), 1995.

[52] A. Hust. Learning similarities for collaborative information retrieval. In *Proceedings of KI 2004 workshop "Machine Learning and Interaction for Text-Based Information Retrieval", TIR-04*, pages 43–54, September 2004.

[53] P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman. Online metric learning and fast similarity search. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, volume 22, pages 761–768, 2008.

[54] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *Proceedings of IEEE Computer Vision and Pattern Recognition conference (CVPR)*, 2008.

[55] T. Joachims. Transductive inference for text classification using support vector machines. In I. Bratko and S. Dzeroski, editors, *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 200–209, Bled, Slovenia, 1999. Morgan Kaufmann Publishers, San Francisco, US.

[56] K. S. Jones and P. Willett, editors. *Readings in information retrieval*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[57] K. Kira and L. A. Rendell. A practical approach to feature selection. In *ML92: Proceedings of the ninth international workshop on Machine learning*, pages 249–256, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[58] I. Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In F. Bergadano and L. De Raedt, editors, *Proceedings of European Conference on Machine Learning (ECML)*, volume 784 of *Lecture Notes in Computer Science (LNCS)*, pages 171–182. Springer Berlin / Heidelberg, 1994.

[59] B. Kulis. Tutorial: Metric learning. In *International Conference on Machine Learning ICML*, Haifa, Israel, June 2010.

[60] B. Kulis, M. Sustik, and I. Dhillon. Learning low-rank kernel matrices. In *ICML '06: Proceedings of the 23rd International Conference on Machine learning*, pages 505–512, New York, NY, USA, 2006. ACM.

[61] B. Kulis, M. A. Sustik, and I. S. Dhillon. Low-rank kernel learning with bregman matrix divergences. *Journal of Machine Learning Research*, 10:341–376, 2009.

[62] O.-W. Kwon and J.-H. Lee. Text categorization based on k-nearest neighbor approach for web site classification. *Information Processing and Management*, 39(1):25–44, 2003.

[63] Y. LeCun, , L. Jackel, L. Bottou, A. Brunot, U. Müller, E. Säckinger, C. Cortes, J. Denker, H. Drucker, I. Guyon, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In *International Conference on Artificial Neural Networks*, pages 53–60, Paris, France, 1995.

[64] Y. Li and P. M. Long. The Relaxed Online Maximum Margin Algorithm. In *Machine Learning*, pages 361–387, 2002.

[65] N. Littlestone. From on-line to batch learning. In *COLT '89: Proceedings of the second annual workshop on Computational learning theory*, pages 269–284, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[66] N. Liu, B. Zhang, J. Yan, Q. Yang, S. Yan, Z. Chen, F. Bai, and W.-Y. Ma. Learning similarity measures in non-orthogonal space. In *Proceedings of the 2004 ACM International Conference on Information and Knowledge Management (CIKM), Washington, DC, USA, November 8-13*, pages 334–341. ACM, 2004.

[67] J. Ma, A. Kulesza, M. Dredze, K. Crammer, L. K. Saul, and F. Pereira. Exploiting feature covariance in high-dimensional online learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010*, volume 9 of JMLR: W&CP, 2010.

[68] M. Maggini, S. Melacci, and L. Sarti. Learning similarity measures from pairwise constraints with neural networks. In V. Kurková, R. Neruda, and J. Koutník, editors, *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN), Prague, Czech Republic*, volume 5164 of *Lecture Notes in Computer Science*, pages 81–90. Springer-Verlag, 2008.

[69] P. C. Mahalanobis. On the generalised distance in statistics. In *Proceedings National Institute of Science, India*, volume 2, pages 49–55, April 1936.

[70] T. Mandl. Learning similarity functions in information retrieval. In *6th European Congress on Intelligent Techniques and Soft Computing (EUFIT)*, pages 771–775, 1998.

[71] A. K. McCallum. BOW: A toolkit for statistical language modeling, text retrieval, classification and clustering, 1996.

[72] S. Melacci, L. Sarti, M. Maggini, and M. Bianchini. A neural network approach to similarity learning. In L. Prevost, S. Marinai, and F. Schwenker, editors, *Proceedings of Third IAPR Workshop on Artificial Neural Networks in Pattern Recognition, ANNPR*, volume 5064 of *Lecture Notes in Computer Science*, pages 133–136, Paris, France, 2008. Springer.

[73] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[74] R. Nock and M. Sebban. Prototype selection using boosted nearest-neighbors. In *Instance Selection and Construction for Data Mining*, pages 301–318. Kluwer Academic Publishers, 2001.

[75] R. Nock, M. Sebban, and D. Bernard. A simple locally adaptive nearest neighbor rule with application to pollution forecasting. *International Journal for Pattern Reconition and Artificial Intelligence*, 17(8):1369–138, 2003.

[76] R. Nock, M. Sebban, and P. Jabby. A symmetric nearest neighbor learning rule. In *EWCBR '00: Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning*, pages 222–233, London, UK, 2000. Springer-Verlag.

[77] A. B. Novikoff. On convergence proofs for perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1963.

[78] S. K. Pal and P. Mitra. *Pattern Recognition Algorithms for Data Mining: Scalability, Knowledge Discovery, and Soft Granular Computing*. Chapman & Hall, Ltd., London, UK, 2004.

[79] R. Paredes and E. Vidal. Learning weighted metrics to minimize nearest-neighbor classification error. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(7):1100–1110, June 2006.

[80] M. Partridge and R. A. Calvo. Fast Dimensionality Reduction and Simple PCA. *Intelligent Data Analysis*, 2(1-4):203–214, 1998.

[81] J.-F. Pessiot. *Apprentissage Automatique pour l'Extraction de Caractéristiques*. Phd thesis, Université Paris VI - Pierre et Marie Curie, 2008.

[82] K. B. Petersen and M. S. Pedersen. The Matrix Cookbook, Oct 2008. Version 20081110.

[83] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[84] M. R. Peterson, T. E. Doom, and M. L. Raymer. GA-facilitated knn classifier optimization with varying similarity measures. In *Proceedings of IEEE Congress on Evolutionary Computation*, volume 3, pages 2514–2521, Sept. 2005.

[85] A. M. Qamar and E. Gaussier. Apprentissage de différentes classes de similarité dans les k-ppv. In *XVIièmes Rencontres de la Société Française de Classification, Grenoble, France*, 2009.

[86] A. M. Qamar and E. Gaussier. Online and batch learning of generalized cosine similarities. In *Proceedings of International Conference on Data Mining (ICDM)*, pages 926–931, Miami, Florida, USA, 2009.

[87] A. M. Qamar, E. Gaussier, J.-P. Chevallet, and J. H. Lim. Similarity learning for nearest neighbor classification. In *Proceedings of International Conference on Data Mining (ICDM)*, pages 983–988, Pisa, Italy, 2008.

[88] A. M. Qamar, E. Gaussier, and N. Denos. Batch document filtering using nearest neighbor algorithm. In *Working Notes of the Cross Language Evaluation Forum, Corfu, Greece*, 30 Sept–02 Oct 2009.

[89] A. M. Qamar, E. Gaussier, and N. Denos. Batch document filtering using nearest neighbor algorithm. In C. Peters, G. Di Nunzio, M. Kurimo, D. Mostefa, A. Penas, and G. Roda, editors, *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, volume 6241 of *Lecture Notes in Computer Science*, pages 354–361. Springer Berlin / Heidelberg, 2011.

[90] A. M. Qamar and Eric Gaussier. Similarity Learning in Nearest Neighbor and RELIEF Algorithm. In *Proceedings of the Ninth IEEE International Conference on Machine Learning and Applications (ICMLA), 12-14 Dec, 2010*, Washington DC, USA, to appear.

[91] A. M. Qamar and Eric Gaussier. Similarity Learning in Nearest Neighbor; Positive Semi-Definitiveness and Relief Algorithm. In *Proceedings of the Second IEEE International Conference on Soft Computing and Pattern Recognition (SoCPaR), 7-10 Dec, 2010*, Cergy Pointoise / Paris, France, to appear.

[92] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, December 2005.

[93] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[94] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326, 2000.

[95] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[96] B. Schölkopf, R. Herbrich, and A. Smola. A generalized representer theorem. In D. Helmbold and B. Williamson, editors, *Computational Learning Theory*, volume 2111 of *Lecture Notes in Computer Science*, pages 416–426. Springer Berlin / Heidelberg, 2001.

[97] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2004.

[98] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[99] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng. Online and batch learning of pseudo-metrics. In *ICML '04: Proceedings of the twenty-first International Conference on Machine learning*, New York, NY, USA, 2004. ACM.

[100] A. Stahl and T. Gabel. Using evolution programs to learn local similarity measures. In K. D. Ashley and D. G. Bridge, editors, *Proceedings of 5th International Conference on Case-Based Reasoning Research and Development, ICCBR, Trondheim, Norway*, volume 2689 of *Lecture Notes in Computer Science*, pages 537–551. Springer, 2003.

[101] P. Sterlin. Overfitting Prevention with Cross-Validation. IAD Masters Report, Université Paris VI - Pierre et Marie Curie, 2007.

[102] Y. Sun and D. Wu. A RELIEF based feature extraction algorithm. In *Proceedings of the SIAM International Conference on Data Mining, SDM*, pages 188–195, Atlanta, Georgia, USA, 2008. SIAM.

[103] N. T. V. Truong. *Apprentissage de fonctions d'ordonnancement avec peu de données étiquetees: Application au filtrage d'information*. Phd thesis, Université Paris VI - Pierre et Marie Curie, 2009.

[104] I. W. Tsang, P. Cheung, and J. T. Kwok. Kernel relevant component analysis for distance metric learning. In *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 954–959, 2005.

[105] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, January 1991.

[106] F. Wang, J. Sun, T. Li, and N. Anerousis. Two Heads Better Than One: Metric+Active Learning and its Applications for IT Service Classification. In *Proceedings of 9th IEEE International Conference on Data Mining (ICDM)*, pages 1022–1027, Miami, Florida, USA, 2009. IEEE Computer Society.

[107] S. Wang and R. Jin. An information geometry approach for distance metric learning. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS) 2009*, volume 5 of JMLR: W&CP, 2009.

[108] K. Weinberger. *Metric Learning with Convex Optimization.* Phd thesis, University of Pennsylvania, US, August 2007.

[109] K. Weinberger, F. Sha, and L. Saul. Convex optimizations for distance metric learning and pattern classification [applications corner]. *IEEE Signal Processing Magazine*, 27(3):146–158, May 2010.

[110] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 1473–1480, 2006.

[111] K. Q. Weinberger and L. K. Saul. Fast solvers and efficient implementations for distance metric learning. In *Proceedings of International Conference on Machine Learning (ICML)*, 2008.

[112] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, Feb 2009.

[113] S. Xiang, F. Nie, and C. Zhang. Learning a mahalanobis distance metric for data clustering and classification. *Pattern Recognition*, 41(12):3600–3612, December 2008.

[114] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 505–512, 2002.

[115] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Proceedings of Advances in Neural Information Processing Systems (NIPS) 17*, pages 1537–1544. MIT Press, Cambridge, MA, 2005.

[116] L. Yang. An Overview of Distance Metric Learning. Technical report, Michigan State University. October 2007.

[117] L. Yang and R. Jin. Distance metric learning: A comprehensive survey. Technical report, Michigan State University, 2006.

[118] L. Yang, R. Jin, R. Sukthankar, and Y. Liu. An efficient algorithm for local distance metric learning. In *AAAI'06: Proceedings of the 21st national conference on Artificial Intelligence*, pages 543–548. AAAI Press, 2006.

[119] Y. Yang and X. Liu. A re-examination of text categorization methods. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49, New York, NY, USA, 1999. ACM Press.

[120] A. Zamolotskikh, S. J. Delany, and P. Cunningham. A methodology for comparing classifiers that allow the control of bias. In *SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 582–587, New York, NY, USA, 2006. ACM Press.

Résumé

Les performances des algorithmes d'apprentissage automatique dépendent de la métrique utilisée pour comparer deux objets, et beaucoup de travaux ont montré qu'il était préférable d'apprendre une métrique à partir des données plutôt que se reposer sur une métrique simple fondée sur la matrice identité. Ces résultats ont fourni la base au domaine maintenant qualifié d'apprentissage de métrique. Toutefois, dans ce domaine, la très grande majorité des développements concerne l'apprentissage de distances. Toutefois, dans certaines situations, il est préférable d'utiliser des similarités (par exemple le cosinus) que des distances. Il est donc important, dans ces situations, d'apprendre correctement les métriques à la base des mesures de similarité. Il n'existe pas à notre connaissance de travaux complets sur le sujet, et c'est une des motivations de cette thèse.

Dans le cas des systèmes de filtrage d'information où le but est d'affecter un flot de documents à un ou plusieurs thèmes prédéfinis et où peu d'information de supervision est disponible, des seuils peuvent être appris pour améliorer les mesures de similarité standard telles que le cosinus. L'apprentissage de tels seuils représente le premier pas vers un apprentissage complet des mesures de similarité. Nous avons utilisé cette stratégie au cours des campagnes CLEF INFILE 2008 et 2009, en proposant des versions en ligne et batch de nos algorithmes. Cependant, dans le cas où l'on dispose de suffisamment d'information de supervision, comme en catégorisation, il est préférable d'apprendre des métriques complètes, et pas seulement des seuils. Nous avons développé plusieurs algorithmes qui visent à ce but dans le cadre de la catégorisation à base de k plus proches voisins.

Nous avons tout d'abord développé un algorithme, *SiLA*, qui permet d'apprendre des similarités non contraintes (c'est-à-dire que la mesure peut être symétrique ou non). *SiLA* est une extension du perceptron par vote et permet d'apprendre des similarités qui généralisent le cosinus, ou les coefficients de Dice ou de Jaccard. Nous avons ensuite comparé *SiLA* avec *RELIEF*, un algorithme standard de re-pondération d'attributs, dont le but n'est pas sans lien avec l'apprentissage de métrique. En effet, il a récemment été suggéré par Sun et Wu que *RELIEF* pouvait être considéré comme un algorithme d'apprentissage de métrique avec pour fonction objectif une approximation de la fonction de perte 0-1. Nous montrons ici que cette approximation est relativement mauvaise et peut être avantageusement remplacée par une autre, qui conduit à un algorithme dont les performances sont meilleurs. Nous nous sommes enfin intéressés à une extension directe du cosinus, extension définie comme la forme normalisée d'un produit scalaire dans un espace projeté. Ce travail a donné lieu à l'algorithme *gCosLA*.

Nous avons testé tous nos algorithmes sur plusieurs bases de données. Un test statistique, le s-test, est utilisé pour déterminer si les différences entre résultats sont significatives ou non. *gCosLA* est l'algorithme qui a fourni les meilleurs résultats. De plus, *SiLA* et *gCosLA* se comparent avantageusement à plusieurs algorithmes standard, ce qui illustre leur bien fondé.

Mots-clés: Apprentissage de similarité, cosinus généralisé, k plus proches voisins, filtrage d'information, apprentissage automatique, fouille de données

Abstract

Almost all machine learning problems depend heavily on the metric used. Many works have proved that it is a far better approach to learn the metric structure from the data rather than assuming a simple geometry based on the identity matrix. This has paved the way for a new research theme called metric learning. Most of the works in this domain have based their approaches on distance learning only. However some other works have shown that similarity should be preferred over distance metrics while dealing with textual datasets as well as with non-textual ones. Being able to efficiently learn appropriate similarity measures, as opposed to distances, is thus of high importance for various collections. If several works have partially addressed this problem for different applications, no previous work is known which has fully addressed it in the context of learning similarity metrics for *kNN* classification. This is exactly the focus of the current study.

In the case of information filtering systems where the aim is to filter an incoming stream of documents into a set of predefined topics with little supervision, cosine based category specific thresholds can be learned. Learning such thresholds can be seen as a first step towards learning a complete similarity measure. This strategy was used to develop Online and Batch algorithms for information filtering during the INFILE (Information Filtering) track of the CLEF (Cross Language Evaluation Forum) campaign during the years 2008 and 2009. However, provided enough supervised information is available, as is the case in classification settings, it is usually beneficial to learn a complete metric as opposed to learning thresholds. To this end, we developed numerous algorithms for learning complete similarity metrics for *kNN* classification.

An unconstrained similarity learning algorithm called *SiLA* is developed in which case the normalization is independent of the similarity matrix. *SiLA* encompasses, among others, the standard cosine measure, as well as the Dice and Jaccard coefficients. *SiLA* is an extension of the voted perceptron algorithm and allows to learn different types of similarity functions (based on diagonal, symmetric or asymmetric matrices). We then compare *SiLA* with *RELIEF*, a well known feature re-weighting algorithm. It has recently been suggested by Sun and Wu that *RELIEF* can be seen as a distance metric learning algorithm optimizing a cost function which is an approximation of the $0-1$ loss. We show here that this approximation is loose, and propose a stricter version closer to the the $0-1$ loss, leading to a new, and better, *RELIEF*-based algorithm for classification. We then focus on a direct extension of the cosine similarity measure, defined as a normalized scalar product in a projected space. The associated algorithm is called generalized Cosine simiLarity Algorithm (*gCosLA*).

All of the algorithms are tested on many different datasets. A statistical test, the s-test, is employed to assess whether the results are significantly different. *gCosLA* performed statistically much better than *SiLA* on many of the datasets. Furthermore, *SiLA* and *gCosLA* were compared with many state of the art algorithms, illustrating their well-foundedness.

Keywords: Similarity metric learning, generalized cosine similarity, kNN classification, information filtering, metric learning, machine learning, data mining