Efficient extreme classification

Mouhamadou Moustapha Cisse

0.30in

University Pierre et Marie Curie

Doctoral Thesis

Efficient Extreme Classification

Author:
Moustapha Cisse

Supervisor:
Thierry Artieres
Nicolas Usunier
Patrick Gallinari

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

in the

Machine Learning and Information Access team (MLIA)
Laboratoire Informatique Paris 6 (LIP6)

June 2014

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

Abbreviations

LAH **L**ist **A**bbreviations **H**ere

For/Dedicated to/To my...

Chapter 1

Introduction

The ability to categorize information is a fundamental aspect of human intelligence. We naturally associate visual information or text data with semantic concepts that allow us to organize knowledge. Endowing the machines with the same ability is an important goal of artificial intelligence. This problem has attracted lots of research effort over decades. As a result, systems exploiting machine learning methods to automate daily classification tasks are widely adopted. Document recognition (LeCun et al., 1998) and spam filtering (Tretyakov, 2004) are examples of successful industrial applications of the classification tools developed in the machine learning community.

Despite these success stories, an important gap has existed for a long time between the scales and the complexity of the problems naturally tackled by Humans and those solved by machines. Most of the classification problems solved by computers using machine learning techniques involve at most few hundreds of classes (see small scale datasets in Table 1.1) while Humans naturally discriminate between several thousand of categories. This is intrinsically related to a basic fact of high level human intelligence: the semantic space used by humans to describe the world is extremely large. Indeed, psychologist have postulated that there are around 30 thousands visual categories (Biederman, 1987). Similarly, we use a large (potentially unbounded) number of semantic concepts to tell which are the relevant topics for a given textual document. Therefore, bridging the gap between

humans and the computers' discriminative capabilities requires large amounts of data in order to train accurate learning machines.

In the past few years, we have witnessed a spectacular increase of the amount of data uploaded daily on the web thanks to the growing number of collaborative and social websites such as Flickr[1], Wikipedia[2], Facebook[3] or Instagram[4]. For example, there are about one thousand new articles on the english Wikipedia every day and a total of 4.6 millions of articles as of March 2014[5]. Similarly, 60 millions photos are shared daily on Instagram and a total number of 20 billions of photos have been shared by the 200 millions users as of April 2014[6]. In most cases, this textual or image data comes with a system of labels that describes it. For instance, every wikipedia document is tagged with one of (roughly) the million of labels of the Wikipedia hierarchy. In order to exploit this data and make it available through user friendly applications such as search engines, it is critical to have learning machines that can efficiently deal with a large number of categories. This has remained an unchartered territory in research until recently and most of the existing classifiers are not well suited for problems of this size because of their large computational complexity. Indeed, training most machine learning based classifiers require computational resources that grow much faster than the volume of the data. Therefore, despite the exponential increase of computing power in the past years as predicted by Moore's law, less demanding classification algorithms are needed.

This thesis proposes new approaches for efficiently solving classification problems in presence of a large number of categories also termed *extreme classification*. In this introductory chapter, we explain the particularities of this problem and briefly discuss the approaches we propose. Section 1.1 explains the specificities of extreme classification and the challenges that arise in this setting. In section 1.2 we summarize our main contributions before sketching the chapters of the thesis in section 1.3.

[1] https://www.flickr.com
[2] http://www.wikipedia.org
[3] https://www.facebook.com
[4] http://instagram.com
[5] http://stats.wikimedia.org/EN/TablesWikipediaEN.htm
[6] http://blog.instagram.com/post/80721172292/200m

Name	#cat	#features	#docs	cat/doc	structure
news20	20	19,996	139217	1	Tree
rcv1	101	47236	806791	3.1	Tree
Yahoo! Directory	132,199	4,194,304	792,601	2.2	Tree
Ohsumed	14,321	72,076	233,445	12	Tree
DMOZ 2011	27,875	497,992	594,158	1.02	Tree
SWiki 2011	36,504	346,299	538,148	1.86	Graph
LWiki 2013	325,056	1,617,899	2,817,603	3.26	Graph
BioAsQ	26,563	1,617,899	10,876,004	12.55	Tree

TABLE 1.1: Examples of small scale and extreme classification datasets

1.1 Challenges in Extreme Classification

Extreme Classification is the emerging research field that tackles the task of classifying in presence of a large number of categories (which we will also call classes or labels throughout this thesis). It has gained popularity in the last few years and has been the central topic of several workshops such as the ECML-PKDD Large Scale Hierarchical Text Classification (LSHTC) workshop series 2010-2013 [7], the NIPS 2013 Extreme Classification workshop[8] and more recently the WSDM 2014 Web-Scale Classification workshop [9]. During the same period, contests such as Pascal Large Scale Hierarchical Classification challenge organized in conjunction with the LSHTC workshops, the Imagenet challenge[10] and the BioAsQ challenge [11] have been organized. The workshops allowed the researchers to discuss the main challenges arising in extreme classification problems while the contests have introduced new benchmark datasets that help improving our understanding of the specificities of the problem. Extreme classification problems are mainly of two kinds: single label classification problems in which each instance belongs to only one category and multilabel classification where every instance can be associated to many categories. In the latter case however, the number of categories per instance is generally very small compared to the number of labels involved in the

[7] http://lshtc.iit.demokritos.gr/
[8] http://research.microsoft.com/en-us/um/people/manik/events/XC13/index.html
[9] http://lshtc.iit.demokritos.gr/WSDM-WS
[10] http://image-net.org/challenges/LSVRC
[11] http://www.bioasq.org

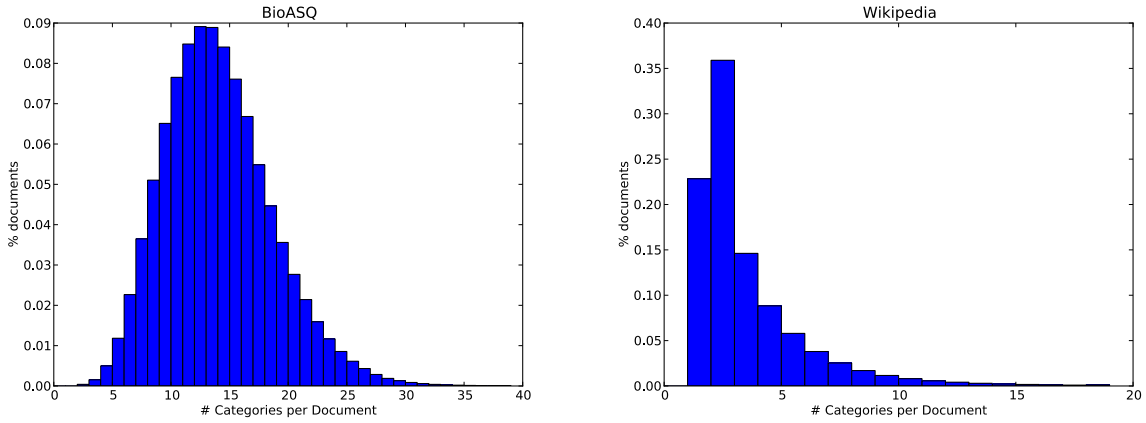


FIGURE 1.1: Distribution (in $\log_2$ scale) of the label set sizes on BioAsQ dataset (left) and on Wikipedia dataset (right).

problem as shown in Figure 1.1. The analysis of extreme classification datasets exhibits common features shared across classification problems whenever the number of categories becomes very large. Some of these features, such as the existence of a hierarchy that relates the categories, are specific to extreme classification problems. Other features such as the imbalance between the classes and the presence of very rare categories are more commonly encountered problems in traditional classification tasks. However, their combination is more specific to extreme classification problems. We discuss here the most salient specificities of extreme classification problems and the challenges they induce.

1.1.1 Class Imbalance/Data scarsity

The average category size in extreme classification problems is tightly related to the number of categories involved. The larger the number of categories, the smaller the average category size. This has been observed on several datasets as depicted on Figure 1.2 in which the average category size increases when the number of categories decreases for all datasets having a large number of categories (e.g. LSHTC datasets and Wikipedia datasets). However, the same observation does not hold for small size datasets such as CLEF and RCV datasets. It has also been observed on many extreme classification problem that the size of the categories are power law distributed. An instance of this phenomenon for the BioAsQ classification

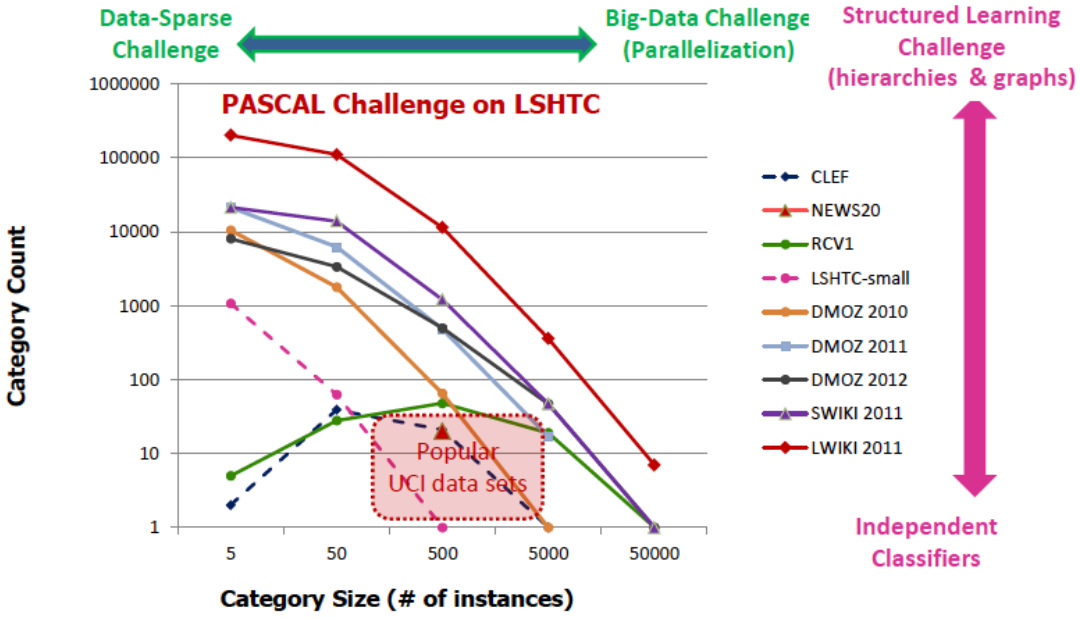


FIGURE 1.2: Landscape of research challenges in Extreme Classification (Yiming Yang Talk at WSC workshop WSDM 2014)

problem which has about 25K categories is depicted in Figure 1.3. This power law distribution of the size of categories implies a severe imbalance between the categories which can make learning very difficult. The same phenomenon appears on the Yahoo! dataset which has 246K labels. In this dataset, 76% of labels have less that 5 instances. Even though solutions to the class imbalance problem have been proposed in previous research (Japkowicz and Stephen, 2002), the scale of the problems for which these solutions were proposed is very different and the distribution of the size of categories is not power law. Therefore, it is necessary to take this phenomenon into account when building new models for extreme classification.

1.1.2 High dimensionality/Large sample size

As shown in Table 1.1, the dimensionality of the input space (the number of features) is very large in extreme classification problems. For the large Wikipedia dataset which has about 325K labels for example, there are more than a million

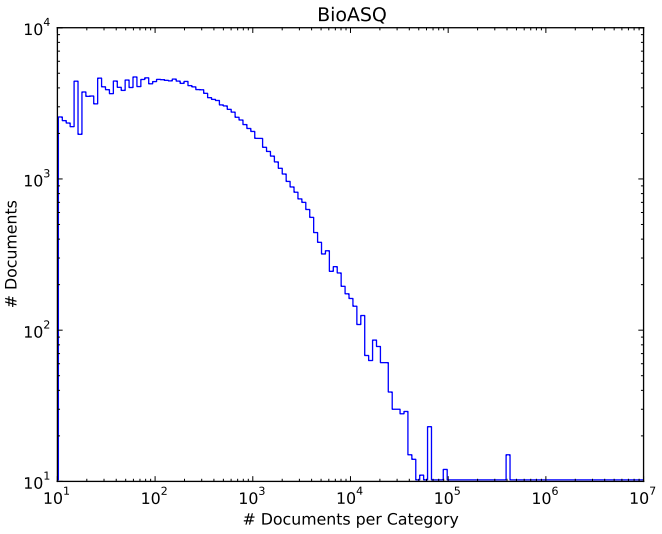


FIGURE 1.3: Distribution of size of categories for the BioAsQ dataset. The distribution exhibits a power law phenomenon.

of features. This observation is valid for the several other datasets and can be explained in a rather intuitive manner. Indeed, for textual data such as DMOZ or Wikipedia, some words of the vocabulary are very specific to some categories. Hence, the more categories we have, the larger the size of the vocabulary used. In order to make learning feasible when the number of features is very large, several feature selection and representation learning approaches have been proposed (Bengio et al., 2013, Molina et al., 2002). However, they were not initially designed for extreme classification problems. Similarly, when the number of categories is very large, the number of examples is also very large since each category is represented at least once. Learning with large amounts of data have been successfully achieved through stochastic optimizations methods (Bottou and Bousquet, 2008, Fan et al., 2008) even though these methods also must be improved in order to scale to the extreme classification setting.

1.1.3 Structure and Label Dependence exploitation

Extreme classification problems generally come with a structure representing the relationships between the labels. This structure carries some semantic about the

labels and is often organized as a hierarchy or a graph. For example, the labels of the Wikipedia dataset are structured as a graph while those of the DMOZ dataset are organized in a hierarchy. Another kind of structure information is the co-occurrence between the labels in extreme multilabel problems. Indeed, some labels are positively correlated while many others never appear together. While the classifiers can be trained independently in small scale problems, it has become an opinio communis that the structure information between the labels can be leveraged in order to improve classification performances. The larger the number of labels, the more important is the structure between the labels as depicted in Figure 1.2. Indeed, the datasets with a large number of categories (see Table 1.1) all come with a structure between the labels while those with a small number of labels rarely come with such structure. There have been successful attempts to label structure exploitation in Extreme SLC (Gopal and Yang, 2013, Weinberger and Chapelle, 2008). But only few approaches exist for MLC problems and most of them do not scale to the classification extreme setting (Dembczynski et al., 2010b, Hariharan et al., 2010)

1.1.4 Training/Inference Complexity reduction

As for most of the extreme classification problems presented in table 1.1, when the number of categories is very large, both the sample size and the dimensionality of the input space are very large. This results in both memory and computational burden when most of the classical learning machines are used. For example, if the one versus rest classifier that learns one classifier for each category (Rifkin and Klautau, 2004b) is used to solve the classification problem having 325K categories[12], more than a 1000 GB are necessary to store all the parameters. Moreover, each of the 325K classifiers must be evaluated before the relevant categories of a given test instance are recovered. The same observation holds for several other methods such as single machine large margin classifiers (Weston and Watkins, 1998) and deep neural networks (Bengio, 2009). Also, these approaches cannot be easily parallelized if one wants to exploit the structure that comes with these problems as shown in Figure 1.2. Therefore, it is critical to come up with new

[12]http://lshtc.iit.demokritos.gr

approaches having sublinear training and inference complexity in the number of categories. This has recently been the main subject of several contributions in both the single label and the multilabel setting. Most of these approaches focus on reducing either training (Dekel and Shamir, 2010) or inference complexity (Bengio et al., 2010, Deng et al., 2011). Therefore, a lot remains to be done in order to make both learning and prediction fast and memory efficient.

1.2 Contributions

As stated in the previous section, the presence of a large number of categories gives rise to several (sub-)problems that must be addressed in order to solve extreme classification. However, we decide to focus in this thesis on training and inference complexity reduction in extreme classification problems. Indeed, making learning feasible and allowing fast inference is a pre-requisite to the use of machine learning based techniques in real world applications dealing with a large number of categories. We propose new approaches for efficiently classifying in this setting while maintaining competitive classification performances. Our contribution is twofold and is mainly built on learning low dimensional binary representation of the classes.

The first method we propose deals with extreme single label classification. It uses hierarchical information to learn compact binary codes for the categories. The representation learning procedure uses an auto-encoder based architecture (Bengio et al., 2013). The method bares similarities with Error Correcting Output Codes (Dietterich and Bakiri, 1995) (ECOC). However, the induced binary problems are empirically shown to be easier than those induced by the randomly generated codes in ECOCs. Overall, this approach gives competitive performances compared to classical one versus rest method and error correcting output codes.

Our second contribution deals with extreme multilabel classification. It is based on the use of Bloom Filters (Bloom, 1970) for representing subsets of labels using low dimensional binary vectors. The first approach we propose uses standard Bloom Filters to encode and decode the subsets. Even though this method gives

competitive performances, it is not robust to individual mistakes of binary classifiers. To overcome this problem, a second approach that exploits the following key property of extreme multilabel classification problems is proposed: when the number of categories is very large, many labels never appear together. This new method is provably robust and gives competitive performances.

1.3 Outline

This section outlines the core chapters of this thesis. The first two chapters are dedicated to extreme single label classification and the last two's focus is extreme multilabel classification.

- Chapter 2 is a review of the main techniques that have been proposed in the literature for extreme single label classification. This chapter is divided in two main sections which are respectively about flat approaches and hierarchical methods for extreme single label classification.
- Chapter 3 presents our contribution to extreme single label classification. The material presented in this chapter has been published at the European Conference of Machine Learning (ECML-PKDD) in 2012 (Cissé et al., 2012).
- Chapter 4 reviews the several methods recently proposed in the blossoming research field of extreme multilabel classification.
- Chapter 5 presents the Bloom Filter based approaches we proposed for extreme multilabel classification. Part of the material presented in this chapter has been published at the Neural Information Processing Systems conference (NIPS) in 2013 (Cisse et al., 2013) and an extend version is in preparation for the Machine Learning Journal.

Chapter 2

Extreme Single Label Classification

2.1 Introduction

Single label classification is a well studied problem in machine learning for which several effective solutions resulting from decades of research have been derived and now widely applied to solve industrial problems (Bishop, 2006). However, the rapid development of the internet and the increasingly large amount of available labelled data (thanks to social and collaborative websites such as Wikipedia or Flicker) have changed the nature of the problem and made the most of the traditional approaches to single label classification obsolete since they do not scale to extreme classification. Single label classification has been initially tackled with *flat techniques*. Recently, there has been an increased interest in *hierarchical methods* because of their reduced complexity compared to flat approaches. In this chapter, we present the main contributions in the literature from these two families of approaches that can be applied to solve extreme single label classification.

2.2 Flat approaches

Flat approaches to multiclass categorization approaches do not rely on a hierarchy at inference to reduce their complexity (conversely to hierarchical approaches that will be discussed in the next section) even though they can exploit existing semantic information to increase their accuracy (Weinberger and Chapelle, 2008). This family of method can be divided into two subgroups that are machine learning reductions (Allwein et al., 2001, Dietterich and Bakiri, 1995) and single machine classifiers (Weinberger and Chapelle, 2008, Weston and Watkins, 1999). These two subgroups are rather different in the way they approach the multiclass classification problem. On one hand machine, learning reductions rest on the predictions of independently learned binary classifiers to produce the final prediction for a given test instance. A notable examples of machine learning reductions is the infamous one-versus-all classifier that will be discussed in more details next. One the other hand single machine classifiers are either embedding based methods or extensions of binary classifiers to the multiclass setting and hence generally require a joint learning of all their parameters.

Extending classical binary classifiers such as support vector machines and logistic regression to the multiclass case has been extensively studied and has resulted in several effective methods such as multiclass support vector machines (M-SVM) (Weston and Watkins, 1999), softmax regression and more recently deep neural networks (Bengio, 2009). However, despite their proven accuracy, these methods do not scale to extreme classification setting due to large their computational burden at both training and inference even though there are recent attempts to parallelize their training (Gopal and Yang). Therefore, we will only focus in this study on methods that are scalable to extreme classification and refer the interested reader to the following works and the references therein (Bengio, 2009, Bordes et al., 2007, Crammer and Singer, 2002, Gopal and Yang, Weston and Watkins, 1999). We describe next machine learning reductions and embedding based approaches to multiclass categorization and discuss how they fit in the context of extreme classification.

2.2.1 Machine Learning Reductions

Machine learning reductions of multiclass classification to binary classification have been around for a long time (Dietterich and Bakiri, 1995, Schölkopf et al., 1995). However, these works have only been unified recently in the same framework including approaches such as Error Correcting Output Codes, One-versus-All, One-versus-One, Filter Trees and many more[1] (Allwein et al., 2001). The key idea in all these methods is to use existing binary classifiers and combine them in order to have a multiclass classifier. The difference between these methods is therefore the way the binary classifiers are combined. This guides the choice of the right method for a given task because it governs the final performance of the methods and their complexity. For example, some widely used methods such as One-versus-One classifier (which train a binary classifier for each pair of label and adopt a voting scheme at inference) cannot be used be used in extreme classification problems (despite their proven performances) because their quadratic complexity in the number of labels $O(L^2)$. Next we describe two popular and scalable approaches that are One-versus-Rest (OVR) and Error Correcting Output Codes (ECOC).

2.2.1.1 Binary Classification

The task of classification consist in learning, given a set of training examples $\mathcal{D} = \{(x_i, y_i)\}_{1 \leq 1 \leq n}$, a mapping (or hypothesis) from the d-dimensional feature space to the label space : $h : \mathcal{X}^d \rightarrow \mathcal{Y}$. The simplest non-trivial classification problem that can be considered is *binary classification* where the set of labels is reduced to $\mathcal{Y} = \{-1, +1\}$. This is arguably the most studied machine learning problem because of its obvious practical interest on its own, and also because it is a building block of more complicated machine learning systems such as multiclass classifiers. From an *empirical risk minimization* (ERM) point of view, learning a binary classifier reduces to finding the best hypothesis h from a hypothesis class $\mathcal{H}$ that minimizes the average number of disagreements between the predictions $h(x_i)$ and the actual labels y_i on the finite training set $\mathcal{D}$. This quantity, called the *empirical risk* as it is an approximation of the classifier's expected risk on

[1] http://hunch.net/ reductions-tutorial/

the distribution P from which the data was drawn, is generally associated with a regularization term to prevent overfitting (Vapnik, 1995). The final empirical risk is expressed as:

$$R_{\mathcal{D}}(h) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(h(x_i), y_i) + \lambda||h|| \tag{2.1}$$

where $\mathcal{L}$ is the loss function measuring the penalty of predicting $h(x_i)$ when the actual label is y_i for a given instance, λ is the weights the importance of the regularization term. The empirical risk minimizer is obtained when the minimum of $R_{\mathcal{D}}$ in the hypothesis class $\mathcal{H}$ is attained: $h^* = \text{argmin}_{h\in\mathcal{H}} R_{\mathcal{D}}(h)$. Depending on the difficulty of the problem at hand, various types of hypothesis classes such as linear models, kernel machines or neural networks can be used (Bishop, 2006). For instance, If the linear hypothesis class of functions is considered, each hypothesis h is parameterized by a weight vector $\mathbf{w}$ such that for a given example its corresponding prediction can be expressed as $h_{\mathbf{w}}(x) = \mathbf{w}^T x + b$ where b is a bias term. Therefore, seeking for the best hypothesis h^* in this case is equivalent to searching for the best weight vector $\mathbf{w}^*$ according to the chosen loss function $\mathcal{L}$.

The natural classification loss function we would like to optimize for is the *zero-one loss* : $\mathcal{L}_{0/1}(h(x_i), y_i) = I(h(x_i) \neq y_i)$. It counts the number of disagreements between the prediction and the actual label. However this loss is not optimization friendly because it is not convex. Therefore, convex surrogates of $\mathcal{L}_{0/1}$ are used in practice hence resulting in various classifiers. Two notable examples of convex surrogates for the *zero-one loss* are the *Hinge loss* and the *Logistic loss* whose respective expressions are given below for an instance (x, y) and a hypothesis h and figure 2.1 shows how they upper bound the zero-one loss function (Bishop, 2006) :

$$\mathcal{L}_{hinge}(y, h(x)) = \max(0, 1 - y \cdot h(x)) \tag{2.2}$$

$$\mathcal{L}_{logistic}(y, h(x)) = \log(1 + \exp(y \cdot h(x))) \tag{2.3}$$

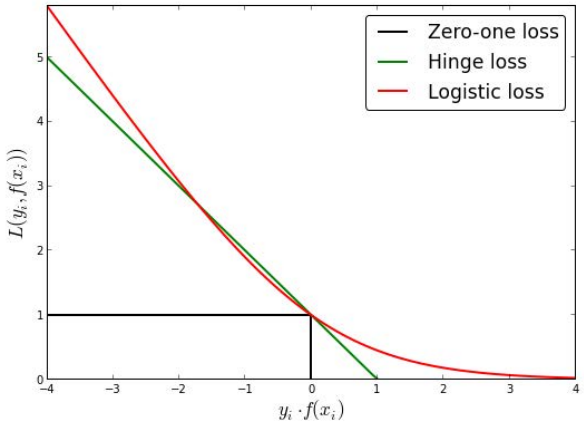


FIGURE 2.1: Surrogates loss functions for the zero-one loss

When the linear hypothesis class is considered, the Hinge loss and the Logistic loss respectively give rise to the *Support Vector Machine* (SVM) and the *Logistic Regression* (LR) classifiers. Several efficient solvers exist for these two problems[2][3] mainly relying on stochastic gradient and dual coordinate descent algorithms (Bottou and Bousquet, 2008, Yu et al., 2011). Both of SVMs and LR models have demonstrated state of the art performances on several large scale binary classification benchmarks [4]. They have become methods of choice for solving binary problems but also as for building machine learning reduction based multiclass classifiers. This has been successfully done with One-Versus-All (OVA) and Error Correcting Output Code (ECOC) classifiers as will be discussed next.

2.2.1.2 One-versus-Rest Classifier

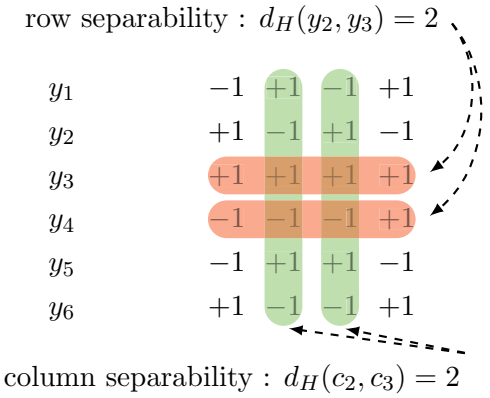
One-versus-Rest (OVR) also called One-versus-All (OVA) is the simplest approach among multiclass machine learning reductions to binary. It consist in training training a binary classifier for each category to distinguish its examples from those of the other classes. At inference, all the classifiers are evaluated and the test example is associated to the category whose classifier is the most confident (has

[2]http://www.csie.ntu.edu.tw/ cjlin/liblinear/
[3]http://leon.bottou.org/projects/sgd
[4]http://largescale.ml.tu-berlin.de/about/

highest prediction). This a winner-takes-all principle. Early work using this strategy dates back to (Schölkopf et al., 1995). However a more cited paper about this method is (Rifkin and Klautau, 2004b). In this work, the authors empirically show that when the binary classifiers are correctly calibrated, this approach outperforms other machine learning reductions (such as One-versus-One classifier (which trains a binary classifier for each pair of category) and Error Correcting Output Codes) and multiclass support vector machines (M-SVM) (Crammer and Singer, 2002, Weston and Watkins, 1999). Moreover, OVA is readily parallelizable since the binary classifiers can be trained and evaluated independently. This makes it a good candidate for extreme classification despite the class imbalance problem generally faced when training the binary classifiers.

2.2.1.3 Error Correcting Output Codes

Solving multiclass categorization problems with error correcting output codes (ECOC) has been introduced in (Dietterich and Bakiri, 1995). In this early work, the presented method consist in associating each of the L classes with a bit vector also called binary codeword of fixed size d_e. The set of codewords are the rows of a coding matrix $M \in \{-1, +1\}^{L \times d_e}$ whose columns correspond to binary classification problems. For each column l, its corresponding binary induced problem reduces to discriminate between the examples of the classes whose corresponding codeword M_j are such that $M_{jl} = +1$ from those for which $M_{jl} = -1$. Therefore, given a set of training examples $\{(x_i, y_i)\}_{1 \leq n}$ and a coding matrix M, the set of positive and negative example examples of the binary problem induced by column l respectively write $\mathcal{D}^l_+ = \{(x_i, y_i) : M_{y_i l} = +1\}$ and $\mathcal{D}^l_- = \{(x_i, y_i) : M_{y_i l} = -1\}$. Binary classifiers also called dichotomizers $(h_i)_{1 \leq i \leq d_e}$ (such as logistic regression or support vector machines previously described) are learned to predict the bit positions of a codeword. A given test instance x is associated to the class whose binary codeword is the closest (according to some distance measure d) to the predicted codeword for x. If we denote this decoding procedure D, we have: $D(x) = \text{argmin}_j \, d(M_j, \mathbf{h}(x))$ where $\mathbf{h}(x) = (h_1(x), \ldots, h_{de}(x))$ and M_j is the codeword of class j. The decoding procedure generally uses the hamming distance d_H



average row separability : $1.83, (std = 1.12)$

	y_1	y_2	y_3	y_4	y_5	y_6
y_1	0	4	2	1	2	2
y_2	4	0	2	3	2	2
y_3	2	2	0	2	2	2
y_4	1	3	2	0	2	1
y_5	2	2	2	2	0	4
y_6	2	2	2	1	4	0

FIGURE 2.2: Row and column separability properties ensure good discrimination and error correcting capabilities of the the ECOC model

measure since it is simple and fast to compute thanks to existing procedures (Pappalardo et al., 2009). Subsequent studies such as (Allwein et al., 2001) proposed to use ternary codes instead of binary codes to reduce the number of classes involved in the binary induced problems.

Choosing a good coding matrix is a key factor in the success of an ECOC based multiclass categorizer. Mainly, the two following properties (also depicted in figure 2.2) should be ensured:

Row separability : the codewords must be well separated to guarantee error correction capabilities of the method. Indeed, if the smallest hamming distance between two codewords is δ, the ECOC procedure will be able to correct up to $\delta/2$ mistakes of the dichotomizers at inference (Allwein et al., 2001)

Column separability: to avoid correlated errors between dichotomizers, it is necessary that the problems are sufficiently different. This is guaranteed when the columns of the matrix are well separated.

In general, these two properties are taken into account when generating the coding matrix by sampling several matrices such that $P(M_{ij} = +1) = 1/2$ and $P(M_{ij} = -1) = 1/2$ and considering the one having the best error correcting capabilities. Another approach is to choose the coding matrix to be a Hadamard

matrix (Langford and Beygelzimer, 2005) which guarantees that the distance between any two codewords is $L/2$.

Another important element in the final performance of a ECOC based classifier is the accuracy of the dichotomizers. Training accurate dichotomizers is a challenging problem when the coding matrix is randomly generated. In this case there is no other way of ensuring accurate dichotomizers than using powerful non linear classifiers such as kernel machines or neural networks as suggested by (Dietterich and Bakiri, 1995) because of the difficulty of the binary induced problems. To overcome this limitation of randomly generated codewords, several authors have proposed to learn the coding matrix (Allwein et al., 2001, Gao and Koller, 2011b, Schapire, 1997). An important contribution in this line of work with applications to extreme classification is the work by (Zhao and Xing, 2013). This approach learns a coding matrix M with "easy" binary induced problems while ensuring well separated codewords to guarantee error correcting capabilities of the method by solving the following problem.

$$\text{max: } F_b(M) - \lambda_r F_r(M) - \lambda_c \sum_{i=1}^{d_e} ||m_i||_2^2 \tag{2.4}$$

$$\text{s.t. } M \in \{-1, 0, +1\}^{L \times d_e} \tag{2.5}$$

$$\sum_{i=1}^{L} I(M_{il} = +1) \geq 1, l = 1, \ldots, d_e \tag{2.6}$$

$$\sum_{i=1}^{L} I(M_{il} = -1) \geq 1, l = 1, \ldots, d_e \tag{2.7}$$

$$\sum_{i=1}^{d_e} I(M_{il} \neq 0) \geq 1, l = 1, \ldots, L. \tag{2.8}$$

where I is the indicator function. $F_b(M)$ is a function that measures the separability of each binary partition problem associated with columns of M, and reflects the expected accuracy of the dichotomizers. $F_r(M)$ measures the correlation between the codewords, therefore minimizing it increases the error correcting ability of the resulting matrix. The $L_2 - norm$ regularization of each column m_i controls

the complexity of the binary induced problems. The hyper-parameters λ_r and λ_b control the relative importance of the competing objectives. The first integrity constraint 2.5 imposes the solution to be a set of ternary codewords. It is the main difficulty of the problem since it makes it NP-hard. The last two constraints ensure respectively that (1) for each of the binary induced problems the sets $\mathcal{D}_+$ and $\mathcal{D}_-$ are never empty so that trivial problems are avoided and (2) each label of the original problem appears at least in one induced problem. The authors instantiate the functions $F_b(M)$ and $F_r(M)$ and propose relaxation of the integer which allow them to efficiently solve this problem despite its apparent difficulty. Even though the final optimization problem requires sophisticated techniques because of the non-convexity of the problem and results in an approximate solution, this method achieves convincing performances compared to One-versus-All baseline and randomly generated codewords.

Error Correcting Output Codes are an interesting option for extreme single label classification for several reasons. First, they enjoy the main feature of reduction based methods which is that the binary classifiers can be trained independently in parallel. Second, the codeword size can be much smaller than the number of labels when the latter is large ((Allwein et al., 2001) suggest to use codewords of size $O(10 \log L)$ for binary coding matrices). Moreover, even the randomly generated codewords have demonstrated good performances specially on very under-represented classes because of the way the binary problems are created (Dietterich and Bakiri, 1995).

2.2.1.4 Discussion

Reducing multiclass to binary is an elegant and natural way of tackling classification problems in that we do not need to reinvent the wheel. Indeed, efficient solvers exist for binary classification and that resource can be leveraged to tackle extreme classification provided efficient and tractable ways of combining binary classifiers with theoretical guarantees on the final performance based on the binary classifiers' performances. This is an active research trend [5].

[5] http://hunch.net/ jl/projects/reductions/reductions.html

2.2.2 Embedding approaches

Nearest Neighbor (NN) approaches are powerful non parametric methods that can model complex non-linear decision surfaces (Bishop, 2006). They have achieved state of the art performances when the distance metric used for nearest neighbor search is learned as for large margin nearest neighbor classifiers (Weinberger and Saul, 2009). However, because of their linear computational complexity in the number of examples they do not scale to extreme classification problems unless specialized data-structures are used. For example, *kd-trees* (Bentley, 1975) can speed up the nearest neighbor search and reduce it to $O(d \log m)$ (where d is the size of the input space and m is the number of examples) at the cost of small performance drop. An improved version of this approach consist in learning a distance metric together with a dimensionality reduction. This method was championed by large margin component analysis (LCA) (Torresani and chih Lee, 2007) which projects the data in a lower dimensional space of size d_e ($d_e << d$) while learning the distance metric. Overall, this method results in a final complexity of $O(d_e(d + m))$ (or $O(d_e(d + \log m))$ when *kd-trees* are used) and empirically gives better performances.

Another workaround for scaling nearest neighbor approaches to extreme classification is *nearest centroid classifier* (NC). Given a set of training examples $\{(x_i, y_i)\}_{1 \leq i \leq m}$, this approach consist in computing class centroids $\mu_j = 1/c_j \sum_{i \in c_j} x_i$ (where c_j is the set of examples belonging to class j). At inference each example is assign to the class whose centroid is closest $y = \text{argmin}_j ||u_j - x||$. This method has been widely adopted in the text classification community where it is known as the *Rocchio classifier*. It scales better than the Nearest Neighbor approach since its complexity is $O(dn)$ rather than $O(dm)$. Despite this improved complexity, NC classifiers remain costly in extreme classification setting where both the number of classes and the dimensionality of the data are large. Moreover, the high dimensionality of the data generally leads to poor performances when euclidean distance for example is used for nearest mean search (Aggarwal et al., 2001, Beyer et al., 1999).

The combination of distance metric learning with dimensionality reduction has shown effective for to reduce the inference complexity of Nearest Neighbor classifiers while improving classification performances. On the other hand, Nearest Centroid classifiers allow complexity reduction up to linear dependance with in the number of labels rather than linear dependance in the number of examples as is the case for NN classifiers. *Embedding methods* are designed to take the best of both worlds. They project the data and the labels into a joint low dimensional space of size d_e ($d_e << d$ and $d_e << n$) where a distance can be computed between the prototypes and the examples. The final complexity is $O(d_e(d+n))$ and can be reduced to $O(d_e(d+\log n))$ by using kd-trees even though one has to trade some performance to have the logarithmic dependence in the number of labels.

The label embedding procedure consist in learning two projection matrices $W \in \mathbb{R}^{n \times d_e}$ and $V \in \mathbb{R}^{d \times d_e}$ such that each example is close to its corresponding label prototype in the latent space. If each label y is represented in the original label space by a one-hot coded vector of size n composed of zeros at all positions except at y^{th} position: $\phi(y) = (0, \ldots, 1, \ldots, 0)$, the latent representation of the label y is given by $V\phi(y)$. Similarly, the latent representation of each example x is given by $z = Wx$. Classification is then achieved by assigning each transformed test example to the label corresponding to its closest prototype:

$$f_{embed}(x) = \underset{1,\ldots,n}{\operatorname{argmax}} S(Wx, V\phi(y_i)) \tag{2.9}$$

where $S(\cdot, \cdot)$ is some similarity measure generally chosen to be the negative euclidean distance or the inner product. The two matrices can be learned either independently (in which case the corresponding problem can formulated as sequence of convex problems) or jointly (where the corresponding optimization problem is non-convex). The label embedding matrix V can also be constructed using prior information such as the similarity between labels. In (Bengio et al., 2010) such similarity information is obtained via the confusion matrix of a previously trained one-versus-rest classifier while in (Weinberger and Chapelle, 2008) the similarity matrix is calculated using the distance between the labels in a given hierarchy. We detail next the two possibilities for learning the embedding matrices.

2.2.2.1 Sequence of Convex Problems

Solving a sequence of convex problems is the first proposed approach to learning label embeddings (Weinberger and Chapelle, 2008). It is a two step procedure in which prior information is supposed available. To learn the V matrix given a similarity matrix A in which A_{ij} represents the similarity between the labels y_i and y_j, a spectral embedding problem is solved in other to project similar labels in neighboring regions of the new low dimensional space. This problem corresponds to minimizing the following objective: $\sum_{i,j=1}^{n} A_{ij}||V_i - V_j||^2$ subject to the constraints $V^T DV = I$ where $D_{ii} = \sum_j A_{ij}$. This is the same problem solved by the laplacian eigenmaps (Belkin and Niyogi, 2003). Other equivalent formulations such as spectral embedding or locally linear embedding can be used as explained in (Bengio et al., 2004).

Given the matrix V, the second step is to learn the matrix W to project the examples in the same space as the labels. To that end, the following large margin problem is posed:

$$\text{minimize: } \gamma||W||_{FRO} + \frac{1}{m}\sum_{i=1}^{m}\xi_i \tag{2.10}$$

$$\text{s.t. } ||Wx_i - V\phi(i)||^2 \leq ||Wx_i - V\phi(j)||^2 - 1 + \xi_i \quad (\forall j \neq i) \tag{2.11}$$

$$\xi_i \geq 0, i = 1, \ldots, m. \tag{2.12}$$

This problem convex since the constraints 2.21 are linear. At inference, the negative euclidean distance $S(z, z') = -||z - z'||$ is used as similarity measure in equation 2.9. This rather simple approach yields better performances than the classical NC and NN classifiers.

2.2.2.2 Joint Non Convex Embeding

The second way of learning learning label embeddings is to jointly seek for the matrices is V and W without using any prior information. Even though such procedure poses an "egg and chicken" problem as pointed out by (Weinberger and

Chapelle, 2008), there is a simple joint optimization problem proposed in (Bengio et al., 2010)

$$\text{minimize: } \gamma||W||_{FRO} + \frac{1}{m}\sum_{i=1}^{m}\xi_i \tag{2.13}$$

$$\text{s.t. } (Wx_i)^T V\phi(i) \geq (Wx_i)^T V\phi(j) + 1 - \xi_i \quad (\forall j \neq i) \tag{2.14}$$

$$||V_i|| \leq 1, i = 1, \ldots, n \tag{2.15}$$

$$\xi_i \geq 0, i = 1, \ldots, m. \tag{2.16}$$

Here, no prior information is used and the problem is non-convex because of the constraints 2.14. However, learning can be efficiently achieved by using stochastic gradient descent with randomly initialized weights. Also, a different similarity measure can be used at inference $S(z, z') = z^T z'$. Despite the apparent difficulty of this problem, its solution has yielded better performances than the previous convex one in real world extreme classification problems (Bengio et al., 2010) presumably because the class of functions explored is larger and the stochastic gradient descent algorithm used is effective.

2.2.2.3 Discussion

Label embedding is an appealing framework for accurate extreme classification. It is both flexible, simple and allows the incorporation of prior knowledge as has been demonstrated in (Weinberger and Chapelle, 2008). It can also be used as a building block for designing more complex systems such as label embedding trees (Bengio et al., 2010). Moreover, its use goes beyond classification as it has successfully been applied to ranking problems throughout the *WSABIE* system (Weston et al., 2011).

2.2.3 Conclusion

Flat approaches to single label extreme classification achieve training/inference complexity reduction by changing the representation of the labels. While embedding methods project the labels and data in a joint low dimensional continuous space where fast nearest neighbor operations can be computed, machine learning reductions (mainly ECOC) though similar in spirit, use instead binary representation of the labels that allow independent training of the base classifiers and the use of existing off-the-shelf solvers to tackle induced problems. In both cases, the final inference complexity is $O(d_e(d+L))$ where d_e is the size of the new representation of the labels, d is the dimensionality of the data and L is the number of labels. Both approaches has achieved good performances in extreme single label classification benchmarks. However, the gold standard among flat techniques to which new extreme single label classification methods should be compared remains the one-versus-all (OVA) approach because of its competitive performances despite its simplicity.

2.3 Hierarchical Approaches

Given a test data $x \in \mathcal{X} = \mathbb{R}^n$, efficiently predicting its relevant class $y \in \mathcal{Y}$ among many is an instance of a search problem. As such, it can be tackled by a divide and conquer strategy. Hierarchical classifiers (Liu et al., 2005b, Silla and Freitas, 2011) are an important instance of this widely used method in the context of classification in presence of a large number of categories. The popularity of these methods is due to both accuracy and efficiency reasons. Indeed, most of the real world extreme classification problems come with an accompanying taxonomy that carries semantic relationship between the classes. For example, DMOZ [6] is a comprehensive directory of the web with a strong hierarchical backbone organization. Similarly, the MESH [7] directory is organized as a hierarchy of medical topics used for indexing PubMed. Hence, exploiting the hierarchical information can lead to important performance improvements in classification (Bennett and

[6] http://www.dmoz.org/
[7] http://www.ncbi.nlm.nih.gov/mesh

Nguyen, 2009, Koller and Sahami, 1997, Weigend et al., 1999). Moreover, most of the flat approaches are computationally prohibitive for real world applications: *"flat SVMs cannot be used in very large-scale real-world applications, due to their high computational complexity (an average response time, with 10 powerful machines running in parallel, of 0.69s for one single document is not acceptable for large-scale online classification)"* (Liu et al., 2005b). This is in contrast with hierarchical approaches which potentially allow logarithmic time prediction when hierarchical structures such as balanced trees are used for example (Beygelzimer et al., 2009b, Deng et al., 2011). This last feature is very desirable in extreme classification and justifies the increasingly large body of work devoted to hierarchical structure learning for extreme classification (Bengio et al., 2010, Deng et al., 2011, Griffin and Perona, 2008, Marszalek and Schmid, 2008).

A generic hierarchical classifier (also called label tree) is a function $h : \mathcal{X} \to \mathcal{Y}$ where the classes $\mathcal{Y}$ are arranged in a rooted hierarchy $T \in \mathcal{T}$ associated with a set of classifiers F chosen from a particular class of functions $\mathcal{F}$. It can hence be denoted as a tuple $(T, F) \in \mathcal{T} \times \mathcal{F}$. The set of allowed structures for the hierarchy (defined by a set of nodes N and edges E) generally corresponds to trees, in which a node has a single parent, or directed acyclic graphs (DAGs) where many parents are allowed for a single node. For each node $v \in N$ of the hierarchy, the following notions are of importance:

- The set of its parents : $\mathcal{A}(v)$
- The set of its siblings : $\mathcal{S}(i)$
- The set of its children : $\mathcal{C}(v) = \{u \in N, \mathcal{A}(u) = v\}$
- The set of its ith level ancestors : $\mathcal{A}^{(i)}(v)$ with $\mathcal{A}^{(0)}(v) = v$ and $\mathcal{A}^{(i)}(v) = \mathcal{A}(\mathcal{A}^{(i-1)}(v))$
- The set of nodes on the path from the root to node v: $\mathcal{P}(v) = \left\{u \in \mathcal{Y} : \exists i, u = \mathcal{A}^{(i)}(v)\right\}$
- The set of leaves of the sub-hierarchy rooted at v: $\mathcal{L}(T_v)$

In extreme single label classification, the set of allowed classes at a given node v of the hierarchy is often reduced to the leaves of the sub-hierarchy rooted at the

node v, $\mathcal{L}(T_v)$. This setup is called *mandatory leaf node classification* (Silla and Freitas, 2011) and reduces the class of functions to $h : \mathcal{X} \rightarrow \mathcal{L}(T)$. In this case, each node v is associated with a local classifier f_v trained to discriminate between the classes corresponding to the leaves of the sub-hierarchy rooted at the current node $l_v = \mathcal{L}(T_v)$ and some other classes. We consider linear (or kernel) classifiers of the form $f_i(x) = w_i^T \phi(x)$ and refer to the set of classifiers associated with the nodes of the hierarchy as $F = \{f_v\}_{v \in N}$. Learning a hierarchical classifier then equals to finding the optimal tuple $h^* = (T^*, F^*) \in \mathcal{T} \times \mathcal{F}$ minimizing a specific loss function.

Various types of loss functions have been used to learn and evaluate hierarchical classifiers (Kosmopoulos et al., 2013, Sun and Lim, 2001). When the hierarchy represents semantic relationships between the classes (as for ontologies), evaluation measures such as the *hierarchy induced loss* are relevant. For any pair of classes $u, v \in \mathcal{Y}$, the hierarchy induced loss $\gamma(u, v)$ counts the number of edges along the shortest path from u to v in the hierarchy T. Hence, it quantitatively answers the question *"how semantically related are the predicted class and the actual class ?"* for a given test instance. However, when the hierarchy is learned from the data for efficiency reasons, the relationship between its nodes can be semantically meaningless. In this case, the evaluation measure of choice remains the classical 0/1 loss which counts an error every time a wrong class is predicted. Directly learning a hierarchical classifier optimizing either of these two losses can be a difficult task (Bengio et al., 2010). In practice, easier to optimize proxies are generally used to learn the classifiers which are also evaluated using other performance measures such as hierarchical versions of the F-measure to gain more insights in the behavior of the methods .

Given a learned hierarchical classifier h, prediction is achieved by applying Algorithm 1. The process is a depth first search (DFS) based on the scores of the local classifiers. It starts at the root node and selects at each round among the current node's children the one whose associated classifier has the largest score. The same process is repeated until a leaf node is reached. The class corresponding to that final leaf node is then predicted.

1em

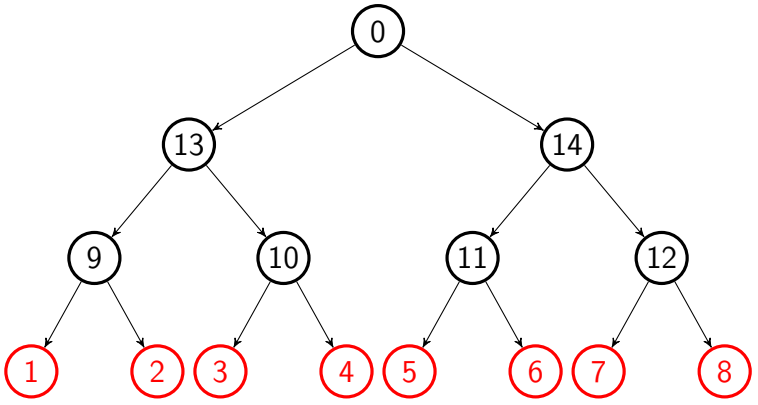


FIGURE 2.3: Example of hierarchical classifier h. The structure of the hierarchy T is a tree. Each node (besides the root node) is associated with a local classifier f_i. The set of classifiers is $\{f_i\}_{i=1}^{14}$ and the allowed classes are the leaves (in red) $\mathcal{Y} = \mathcal{L}(\mathcal{T}) = \{c_1, \ldots, c_8\}$. Also, $\mathcal{P}(7) = \{0, 9, 13\}$ and children of node 9 are $\mathcal{C}(9) = \{1, 2\}$.

Algorithm 1: Hierarchical Prediction Algorithm

Input: test example x, $h = (T, F)$
Let $s = 0$ (root node)
repeat
  Let $s = \text{argmax}_{v \in \mathcal{C}(s)} f_v(x)$
until $|\mathcal{C}(s)| = 0$;
return l_s

Learning accurate hierarchical classifiers poses a number of challenges relative to the *structure of the hierarchy* T itself on one hand and to the local classifiers associated to it, F, on the other (Bennett and Nguyen, 2009, Silla and Freitas, 2011, Yang et al., 2003). First, the data does not usually come with a hierarchical structure of the classes and when a hierarchy is available, its shape usually makes it computationally inefficient or the internal organization of the classes makes its discriminative capabilities unsatisfactory. In either case, a new hierarchy is to be learned from the data (Bengio et al., 2010, Beygelzimer et al., 2009a, Deng et al., 2011, Gao and Koller, 2011a, Griffin and Perona, 2008, Marszalek and Schmid, 2007). Second, to improve the discriminative capabilities of the models associated to the internal nodes of the hierarchy, it is necessary to cope with the *error propagation problem* which occurs when the actual distribution of examples a classifier is predicting over changes from that used during training due to errors at

higher levels of the hierarchy (Bennett and Nguyen, 2009, Gao and Koller, 2011a). It is also equally important to deal with the *requirement for complex decision surfaces at the higher nodes of the hierarchy*. In fact, the induced classification problems solved at the higher levels of the hierarchy involve very generic sometimes meaningless concepts. Solving these problems calls for powerful learning machines and enriched representation of the data (Bengio et al., 2010, Bennett and Nguyen, 2009). A last concern to which many researchers attribute the poor performances of hierarchical classifiers is the *sparsity of labeled data* mainly at the lower levels of the hierarchy where many classes are not statistically enough represented to allow learning accurate classifiers with classical methods (Bennett and Nguyen, 2009, Gopal et al., 2012, Japkowicz and Stephen, 2002, Liu et al., 2005b). For example, 72% of the classes in the Open Directory Project has less than 4 positive instances [8]. Such severe data scarcity causes overfitting problems that lead to poor local classifiers.

In the sequel, we describe the main methods proposed in the literature to tackle the above mentioned problems. These methods can be divided into three main categories. The first group proposes algorithms to learn discriminative hierarchies from the data regardless of the classifiers to be associated with the internal nodes. The second line of work is interested in learning accurate classifiers associated with the internal nodes of a given hierarchy. The last and more recent family of methods jointly learns the structure of the hierarchy and the local classifiers. The general idea behind all these methods however boils down to the same goal of learning efficient models while maintaining competitive performances in comparison to flat methods.

2.3.1 Hierarchical Structure Learning

Most of the traditional approaches to classification such as flat single machine SVMs (Weston and Watkins, 1998), one-versus-one and one-versus-rest reductions (Rifkin and Klautau, 2004a) to name a few, have linear or quadratic inference complexity in the number of classes. They therefore do not scale well to

[8] http://www.dmoz.org/

extreme classification. For the sake of inference efficiency, several authors have early proposed to learn discriminative class hierarchies (Chen et al., 2004, Liu et al., 2005a,c, Vural and Dy, 2004, Zhang et al., 2010). While the validity of this idea has become an opinio communis in the machine learning community (Bengio et al., 2010, Griffin and Perona, 2008, Marszalek and Schmid, 2008), the methods proposed to tackle the challenge of learning discriminative classifiers can be very different. For example, (Marszalek and Schmid, 2007) learn a class hierarchy by exploiting the semantics of the classes and some additional knowledge about inter-class relationships such as Wordnet[9]. In another line of work (Griffin and Perona, 2008) have introduced an approach relying on a *recursive top-down partitioning* of the set of classes to build hierarchies while (Liu et al., 2005c) use a *bottom-up agglomerative clustering*. Similarly, (Liu et al., 2005a) use a method based on K-means clustering conversely to (Zhang et al., 2010) who randomly sample the structure of the class hierarchy by cross-validation. However, The superiority of top-down approaches over bottom-up methods as well as that of learned hierarchies over randomly created ones have been empirically demonstrated (Bengio et al., 2010, Griffin and Perona, 2008). Therefore, we believe the most discriminative feature between the best performing methods , as far as generalization performance and inference speed are concerned, is the type of the learned class hierarchy. The main competitors here are *tree structured class hierarchies*, in which there is a single path from the root node to a given class node, and the *directed acyclic graph* (DAG) which allows many paths from a root to a class node. Because, the best performing methods (irrespective of the type of hierarchy) use spectral clustering (Luxburg, 2007, Ng et al., 2001) to partition the classes, we first describe its formalism before discussing its use to build discriminative tree and DAG structured class hierarchies.

2.3.1.1 Spectral Clustering

Clustering algorithms such as K-means distribute given items into different groups such that items that are similar to each other are in the same group. The similarity between the items is defined in terms of some distance measure. For K-means, the

[9] http://wordnet.princeton.edu/

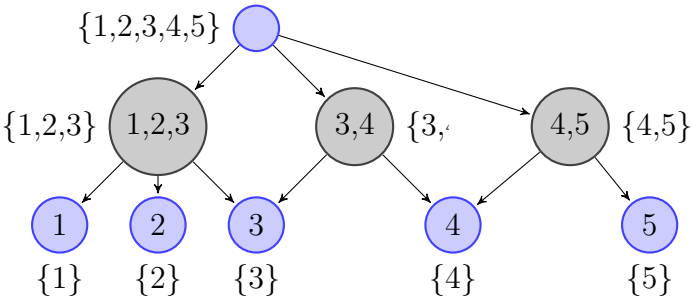


FIGURE 2.4: Directed Acyclic Graph

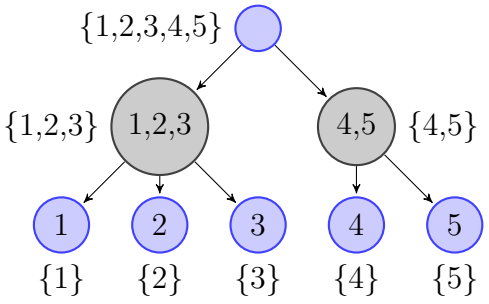


FIGURE 2.5: K-way Tree

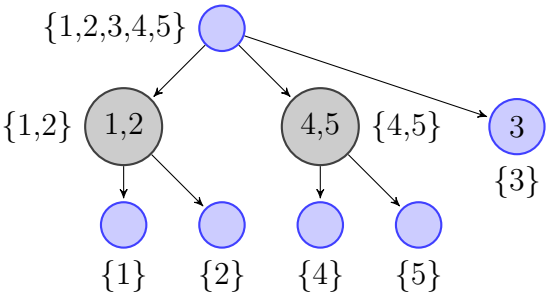


FIGURE 2.6: Binary Tree

overall process reduces to optimizing the within cluster sum of square distances even though other distortion functions can be used. Despite its popularity, some problems have been shown to be notably difficult for K-means (Luxburg, 2007, Ng et al., 2001). Spectral clustering algorithms are powerful alternatives to K-means. The first step of spectral clustering consist in representing the set of items to cluster as a graph in which the vertices N are the items and the edges E represents the similarity between them. We denote the similarity matrix W. Various types

of similarity measures can be used and result in different types of graph. For example, when only the k nearest neighbors (according to some distance measure) of a given item are considered similar, all the nodes of the graph are of degree k and the edges are not weighted ($w_{ij} = 1, \forall (i,j) \in E$). Conversely, dense graphs are obtained when for every two items $(u, v) \in E \times E$, the gaussian kernel is used as a similarity measure: $w_{ij} = \exp(||u - v||^2 / 2\sigma^2)$.

The second step is to find a non-trivial partitioning of the items such that the sum of the weights of edges linking vertices belonging to different clusters is minimized (in the basic form of the algorithm). For two clusters A and B, this quantity is called the *cut*: $cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$. (Ng et al., 2001) propose to minimize instead the normalized cut which leads to more balanced clusters. The corresponding objective function is:

$$J_N = \frac{cut(A,B)}{Vol(A)} + \frac{cut(A,B)}{Vol(B)} \tag{2.17}$$

where $Vol(A) = \sum_{i \in A, j \in N} w_{ij}$ and is the volume of the cluster A. It can be proven that the above problem can equivalently be written as a generalized eigenvalue problem: $(D - W)y = \lambda D y$ where D is the diagonal matrix with $d_{ii} = \sum_j w_{ij}$ and y is the indicator vector of vertices belonging to clusters A ($y = 1$) and B ($y = -1$). This problem is known to be NP-hard. However, when relaxing from binary to continuous values, the solution is obtained by the second eigenvector of the normalized laplacian $L = D^{-1/2}(D - W)D^{-1/2}$.

The last step of the process is to infer a clustering from the solution of the previous eigenvalue problem. If the second eigenvector p^* is considered, a clustering is obtained by a simple thresholding rule: $A = \{i : p^*_i > 0\}$ and $B = \{i : p^*_i < 0\}$. However, if the top-l eigenvectors are considered ($l \geq 2$) as recommended by (Ng et al., 2001), a clustering is obtained by using a K-means algorithm on the spectral representation of the items (in this case, if we have s items, p^* is a $(s \times k)$ matrix whose rows are the new representations of the items).

2.3.1.2 Learning Class Hierarchies

Among the recently proposed class hierarchy building algorithms, the most successful ones are based on a recursive partitioning of the set of classes using spectral clustering (Bengio et al., 2010, Chen et al., 2004, Griffin and Perona, 2008, Marszalek and Schmid, 2008). However, while the power of spectral clustering partly justifies this success, the structure of the learned hierarchy has shown to be equally as important.

Tree structured class hierarchies When using spectral clustering to build discriminative tree structured class hierarchies, the items (the nodes of the graph) correspond to the classes. Since the goal is to partition the classes into easily separable clusters, the affinity matrix used is the symmetrized confusion matrix between the classes (Godbole et al., 2002). The underlying idea is that two class are similar if one's test instances are often classified as belonging to the other class. The confusion matrix can be obtained from a previously trained surrogate classifier such as One-vs-Rest. A common practice is to average different confusion matrices using k-fold cross validation to guarantee more stability. Given a fixed depth and branching factor for the hierarchy, (Bengio et al., 2010) recursively solve graph cut problems (with a previously built affinity matrix) until the desired shape is obtained. A slightly different variant is proposed by (Griffin and Perona, 2008) who, instead of fixing the branching factor, use self-tuning spectral clustering (Zelnik-manor and Perona, 2004) to automatically find the number of clusters at each step.

DAG structured class hierarchies Conversely to trees, the nodes in the directed acyclic graph based approaches correspond to the instances of the classes rather than the classes themselves (Marszalek and Schmid, 2008). The affinity matrix is computed using some adequate similarity measure between the instances depending on the problem of interest. As with trees, the hierarchy is built recursively. At each step, the examples $S = \{(x_i, y_i)\}_{1 \leq i \leq m}$ are then partitioned into two clusters R and L. Further, whenever a class c has one of its instances belonging to a cluster (cluster R for example), it is considered as belonging to that cluster's

set of classes. We denote the set of classes belonging to clusters R (respectively L) as $\mathcal{R}$ (respectively $\mathcal{L}$) and define them formally as $\mathcal{R} = \{y : \exists(x, y) \in R\}$ (respectively $\mathcal{L} = \{y : \exists(x, y) \in L\}$). This way of grouping the classes necessarily results in overlapping clusters of classes ($\mathcal{R} \cap \mathcal{L} \neq \emptyset$) since it is very unlikely that every class has all its instances belonging to a single cluster. In practice, (Marszalek and Schmid, 2008) propose to relax the the partitioning rule in order to have shallower hierarchies. This is done by allowing a class to exclusively belong to a cluster even when a small number of its instances are in another cluster and to belong to two clusters only if many of its instances are both clusters. Moreover, since the instances are used rather than the classes, the resulting graph is much larger and the complexity of finding the solution of the eigenvalue problem via the eigenvectors of the normalized laplacian is cubic in the number of instances involved $O(m^3)$. Nonetheless, if only the second eigenvector of the laplacian is used, optimized algorithms can be used to reduce the overall complexity.

2.3.1.3 Discussion

Empirically, DAGs structured class hierarchies have proven to be more accurate than tree structured ones (Marszalek and Schmid, 2008). The main reason for this is the existence of several paths from the root node to a given leaf node in the hierarchy. *Choosing DAG structured class hierarchy is a natural way of fighting the early confusion problem.* Indeed, binary problems induced by the partitions at the higher levels of the hierarchy are easier with DAGs since they only involve classes that are likely to be separable because the most confusing classes are allowed to belong to either of the two clusters. This illustrates the fact that even a simple design choice (type of hierarchy used) can be an important part in solving one of the fundamental problems of hierarchical classification (error propagation problem for instance). However, choosing DAGs instead of trees is always done at the cost of some computational efficiency: balanced trees guarantee logarithmic time prediction conversely to DAGs. The choice of the best hierarchical structure for a specific problem is therefore equivalent to finding an optimal tradeoff between accuracy and efficiency.

2.3.2 Discriminative Models Learning

2.3.2.1 Independent Optimization of Models: Pachinko Machines

Early work in hierarchical classification has focused on learning recursive classifiers trained independently at each node of a given hierarchy. The so called *pachinko machines* (Liu et al., 2005b, Yang et al., 2003) are very simple since the hierarchy is only used to partition the training data for learning local classifiers. At each node v, the the positive data consist of the training data labelled as belonging to the set of leaf nodes $\mathcal{L}(T_v)$ and the negative data is all the the data belonging to the current node's parents that do not belong to the current node. The overall training set at node v writes $S_v = \{(x_i, y_i) : y_i \in l_{\mathcal{A}(v)}\}$. Slight variations of this idea have been proposed. For example, (Koller and Sahami, 1997) use a small subset of relevant features at each node based on the observation that set of most discriminative features varies across the nodes of the hierarchy while (Dumais and Chen, 2000) propose to use different slack variables at different levels of the hierarchy (assuming large margin classifiers are used). However, modest performance improvements have been reported as resulting from the application of these methods.

Recently, (Bengio et al., 2010) showed that the optimization problem solved by the pachinko machines is a poor approximation of the 0/1 loss if a tree structured hierarchy is used. Indeed, observing that any misclassification in the hierarchy leads to a final wrong prediction and denoting $b_j(x)$ the index of the best node in the hierarchy at depth h, we have:

$$R_{emp}(h) = \frac{1}{m}\sum_{i=1}^{m} \max_{j \in B(x)} I(y_i \notin l_j) \leq G_h = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} I(sgn(f_j(x_i)) = C_j(y_i))$$

where $B(x) = \{b_1(x), \ldots, b_{D(x)(x)}\}$ and $D(x)$ is the depth in the tree of the final prediction for x. Here, because the max is approximated by a sum in G_h, this surrogate can be much larger than the actual loss of interest (0/1 loss) since it adds all the wrong predictions in the tree. By further replacing the indicator function in G_h with hinge loss, they end up with the following optimization problem:

$$\sum_{j=1}^{n}\left(\gamma||w_j||^2+\frac{1}{m}\sum_{i=1}^{m}\xi_{ij}\right) \quad \text{s.t. } \forall i,j, \begin{cases} C_j(y_i)f_j(x_i)\geq 1-\xi_{ij} \\ \xi_{ij}\geq 0 \end{cases} \tag{2.18}$$

where $C_j(y) = 1$ if $y \in l_j$ and -1 otherwise. The final optimization problem is fully decomposable since the parameters of the local classifiers do not interact in the objective function. Therefore, the local classifiers can be trained in parallel. This is a very desirable property in a large scale setting and has been the main reason for the success of these models at the early stages of hierarchical classification. However, with the many recently introduced large scale optimization tools and frameworks, exploiting the hierarchical information by jointly training the local classifiers has become feasible and yields superior performances to pachinko machines as reported in (Bengio et al., 2010, Gopal and Yang, 2013) for example.

2.3.2.2 Joint Optimization of Models

Conversely to the pachinko machines, many authors have recently proposed to consider the hierarchical classifier as a whole and jointly learn all its parameters. Learning the local classifiers jointly would make individual errors impact the updates of all the parameters. The interaction through the updates is expected to avoid the error propagation problem and also to improve classification performance on small classes thanks to the transferred information during the learning process. Indeed, the process can be understood as learning multiple tasks jointly which has been proven to be a promising solution to the data scarcity problem (Argyriou et al., 2008, Caruana, 1997, Widmer et al., 2010). Moreover, jointly learning the parameters of a hierarchical classifier allows to optimize tighter upper bounds of the actual losses of interest (0/1 loss or hierarchical loss) than the general graphical loss G_h optimized by most of the pachinko machines. When learning an arbitrary hierarchical classifier, the set of parameters $\mathbf{w} = (\mathbf{w_i})_{i\leq n}$ of the model is the solution of the generic problem:

$$\arg\min_{\mathbf{w}} \mathcal{R}(\mathbf{w}) + \mathbf{C} \times \mathcal{L}_{\mathbf{emp}}(\mathbf{w}) \tag{2.19}$$

where $\mathcal{L}_{emp}$ is the empirical loss on the training dataset, $\mathcal{R}(\mathbf{w})$ is the regularization term and C is a constant hyper-parameter controlling the trade-off between the two terms. For the pachinko machine, both the regularization term and the empirical loss are fully decomposable and no interaction exists between the parameters of the local classifiers. However, dependence between the local classifiers can be enforced through the regularization term or through the empirical loss term in equation 2.19. We present in this section approaches modeling the dependence via the empirical loss term and defer the regularization based methods to the next section.

The first jointly learned hierarchical classifier has arguably been introduced by (Cai and Hofmann, 2004) as a generalization of the classical multiclass support vector machine formulation to hierarchical tasks. In their framework, the M-SVM problem is reformulated as:

$$\text{minimize: } \gamma \sum_{j=1}^{n} ||w_j||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \tag{2.20}$$

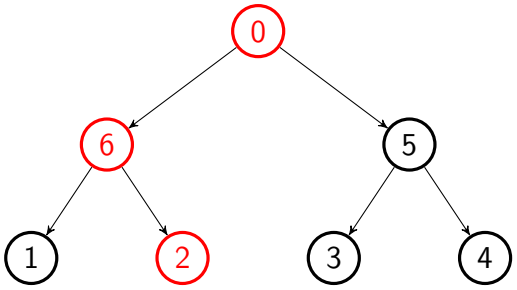
$$\text{s.t. } F(x_i, y_i; w_{y_i}) - F(x_i, y_j; w_{y_j}) \geq 1 - \xi_i \quad (\forall j \neq i) \tag{2.21}$$

$$\xi_i \geq 0, i = 1, \ldots, m. \tag{2.22}$$

where $F(x, y_i; w_i)$ is the linear discriminant function corresponding to the class y usually reduced simple dot product $\langle w_i, x \rangle$. The authors first observe that if the classes are characterized by attribute vectors rather than just arbitrary numbers, the linear discriminant functions can more generally be expressed as:

$$F(x, y; \mathbf{w}) = \langle \mathbf{w}, \Phi(x, y) \rangle = \sum_{r=1}^{n} \lambda_r(y) \langle w_r, x \rangle \tag{2.23}$$

Here, $\Phi(x, y) = \Lambda(y) \otimes x$ and $\otimes$ is the tensor dot product. If $\lambda_r(y) = \delta_{yr}$, each class is interpreted as a binary attribute (one hot coding scheme) and the classical formulation is recovered. It is possible to use instead class attributes reflecting the hierarchical nature of the problem. In (Cai and Hofmann, 2004), for each node v



$$\Lambda(2) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \Phi(x, 2) = \begin{pmatrix} 0 \\ x \\ 0 \\ 0 \\ 0 \\ x \end{pmatrix}$$

FIGURE 2.7: Illustrating the use of class attributes reflecting the hierarchical structure. Class attribute vectors have the same dimension as the number of nodes in the hierarchy except root node. For the class 2, the components corresponding the nodes 6 and 2 are set to one since they are on the path from the root to node 2. The class attribute vectors for 2 and 1 have hamming distance of one from because they are siblings.

of a DAG structured hierarchy authors proposed to use class attributes composed of ones for every component whose corresponding node is in a path from the root to node v and zeros everywhere else. This can be summarized as follows:

$$\lambda_r(y) = \begin{cases} 1 & \text{if} \quad r \in \mathcal{P}(y) \\ 0 & \text{otherwise} \end{cases} \qquad \Phi(x, y) = \Lambda(y) \otimes \mathbf{x} = \begin{pmatrix} \lambda_1(y) \cdot x \\ \lambda_2(y) \cdot x \\ \cdots \\ \lambda_n(y) \cdot x \end{pmatrix}$$

As depicted in figure 2.7, the use of class attributes helps capturing the semantics of the hierarchy. The siblings 1 and 2 for example will have closer attribute vectors than 1 and 3 which are far away from each other in the hierarchy. This clearly has an impact in the constraints because they are less likely to be violated when the classes being compared have close attribute vectors. The resulting discriminant functions are structure aware and decompose into contributions from different levels of the hierarchy. The same idea has first been coined in (Dumais and Chen, 2000) even though the parameters of the hierarchical classifier proposed in that earlier work are not jointly learned

Another benefit of this way of formulating the hierarchical classification problem is that it can be straightforwardly turned into a hierarchical loss optimization

problem. To understand this, it just suffices to penalize the margin violations involving an incorrect class with high loss more severely. This can be done by scaling the margin violation penalties proportionally to the hierarchical loss. In the optimization problem above, it is equivalent to replacing the slack variables in the margin violation constraints ξ_i by $\frac{\xi_i}{\gamma(i,j)}$. The final problem is a good proxy of the actual hierarchical loss as stated by the following proposition.

Proposition 2.1. *(Cai and Hofmann, 2004) Denote by* $(\hat{\mathbf{w}}, \hat{\xi})$ *a feasible solution of the quadratic program in equation then* $\frac{1}{m}\sum_{i=1}^{m}\hat{\xi}_i$ *is an upper bound on the empirical loss* $\hat{\gamma} \equiv \frac{1}{m}\sum_{i=1}^{m}\gamma(y_i, h(x_i))$.

In the realm of extreme classification, very large taxonomies with hundreds of thousands of nodes are ubiquitous. The class attributes vectors are both very high dimensional and extremely sparse and the number of active constraints is potentially very large. which can be a serious computational bottleneck for solving the above quadratic programs. This is the main criticism faced by the hierarchical classifier proposed (Cai and Hofmann, 2004) despite the efficient variable-selection strategy based algorithm for solving this problem and the significant performance improvements reported over flat classifiers and pachinko machines. Moreover, in case of mandatory leaf node classification problem, as far as the 0/1 loss is concerned, satisfying all the constraints (2.21) is not a requirement. Indeed, at inference, we would just count one mistake every time the path followed by the prediction algorithm 1 does not lead to the relevant leaf node. The following remark by (Cesa-Bianchi et al., 2006) explains this intuition: "*If an algorithm fails to label a document with the class SPORTS, then it should not be charged more loss because it also failed to label the same document with the subclass SOCCER*[10] *and the subclass CHAMPIONS LEAGUE*". For a given hierarchy, a data point to predict $\{x, y\}$ and a predicted class $\hat{y}$, this can be written as the following loss function :

$$l_H(y, \hat{y}) = \max(\{|\{i \notin \mathcal{P}(y)\}|\forall i \in \mathcal{P}(\hat{y})\}) \tag{2.24}$$

[10] the name of this game is actually FOOTBALL even though some dumb ass persist in calling it soccer

When learning the Pachinko machines, the optimized proxy was the loss which counts the errors at all the nodes of the hierarchy (denoted l_G). The previously described hierarchical SVM (Cai and Hofmann, 2004) optimizes a surrogate of the hierarchy induced loss l_Δ although it can be used for the 0/1 loss. Compared to these losses, The hierarchical loss l_h 2.24 is a tighter surrogate since it can be easily shown that $l_{0/1} \leq l_H \leq l_\Delta \leq l_G$. Therefore, using path-wise constraints would be more suitable for optimizing the 0/1 loss than the graph-wise contraints 2.21 used so far. Following this idea, (Bengio et al., 2010) proposed to solve this problem:

$$\text{minimize } \gamma \sum_{j=1}^{n} ||w_j||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i^\alpha \tag{2.25}$$

$$\text{s.t. } f_r(x_i) \geq f_s(x_i) - \xi_i, \forall r, s : (y_i \in l_r \wedge y_i \notin l_s), r \in \mathcal{S}(s) \tag{2.26}$$

$$\xi_i \geq 0, i = 1, \ldots, m. \tag{2.27}$$

The constraints 2.26 in this problem express that at each level of the hierarchy, we examine all the siblings of the relevant node (the one at this level which is on the path from the root to the relevant label) and count an error if any of these has a higher score. Therefore, the number of counted error is an upper bound of the the final 0/1 loss since it is 0 when the correct label is predicted but can be as large as the length of the path from the root to the relevant node in the worst case multiplied by the branching factor of the tree. Also, because path-wise constraints are used (with only one slack variable per example), there is at most one active constraint. This results in an important computational complexity advantage compared to the approach by (Cai and Hofmann, 2004) where the number of active constraints is potentially as large as the number of nodes the hierarchy. Moreover, This method also gives competitive performances compared to flat approaches and is superior to pachinko machines.

Although the method by (Bengio et al., 2010) is both accurate and computationally appealing, it is (in its original form) limited to tree structured hierarchies. For directed acyclic graphs, (Cesa-Bianchi et al., 2006) propose a theoretically grounded algorithm that incrementally learns a linear threshold classifier at each

node of the hierarchy. More importantly, their approach is not restricted to the mandatory leaf node classification setup and allows partial-path labeling. Overall, enforcing hierarchical dependences between local classifiers through the empirical loss function has been shown to be an effective way of improving classification performances compared to pachinko machines (Bengio et al., 2010, Cai and Hofmann, 2004, Cesa-Bianchi et al., 2006) presumably because it fights both error propagation problem and the labelled data scarcity problem through the interaction between local classifiers during learning. However, as previously seen, the right method to chose among the many proposed in the literature depends on the final evaluation measure of the hierarchical classifier and the computational budget. For the hierarchy induced loss, the algorithm by (Cai and Hofmann, 2004) is a good candidate while the label tree (Bengio et al., 2010) should be preferred to it when the evaluation measure of interest is the 0/1 loss (mainly for computational reasons). Moreover, it is important pointing out that the loss function used depends on whether the hierarchies carries semantic information, as it is for ontologies, or it is just built for efficiency reasons.

2.3.2.3 Regularization Based Approaches

Modeling local classifiers' dependence through constraints has successfully been achieved by the methods described in the previous section. However, another way of modeling these dependences is through regularization. This is an active area of research which has generated lots of new solutions that can be divided in two subgroups at the light of the type of dependence being modelled:

1. The first line of work originates from the multi-task and transfer learning community (Argyriou et al., 2008, Caruana, 1997, Evgeniou and Pontil, 2004). It aims at enforcing similarity between the classifiers of adjacent nodes of the hierarchy to avoid error propagation and overfitting (Dekel et al., 2004, Gopal and Yang, 2013, Gopal et al., 2012).

2. The second line of work introduces regularization terms inducing dissimilarity between each internal node's local classifier and its children's local

classifiers (Hwang et al., 2011, Xiao et al., 2011). This enhances discriminative features detection to improve classification of closely related classes.

Next, we elaborate on the similarity based recursive classification regularization and the orthogonal transfer regularization which are respectively representative of the above two subgroups. We give more details on the underlying ideas of these approaches and discuss their specificities.

Similarity based dependence modeling: Enforcing local classifiers of neighboring nodes of a given hierarchy to be similar is an intuitive idea first applied through careful design of class attributes reflecting the hierarchical structure (Cai and Hofmann, 2004). Recently, some studies have introduced new approaches based on the regularization (Dekel et al., 2004, Gopal and Yang, 2013, Gopal et al., 2012). The main argument of the proponents of these approaches is to help classes leverage information from nearby classes while estimating local classifiers' parameters. For small classes, this transferred information prevents over-fitting problems that may be faced otherwise. In this line of work, (Dekel et al., 2004) first proposed a principled approach tailored for hierarchy induced loss optimization. They also proposed an efficient online dual based algorithm to solve the problem and confirm the effectivity of their approach with experiments on large scale real world datasets. (Gopal et al., 2012) introduced a hierarchical bayesian method modeling dependencies among nodes using multivariate logistic regression. They model parent-child relationships by placing a hierarchical prior over the children nodes centered around the parameters of their parents; thereby encouraging classes nearby in the hierarchy to share similar local classifiers. The main bottleneck of their approach is the inference tractability issues generally arising in bayesian methods despite the proposed parallel variational inference algorithm.

More recently, (Gopal and Yang, 2013) proposed a recursive regularization framework that can be applied to model arbitrary graphical dependencies. In the context of hierarchical classification, they formulate their approach as follows:

$$\text{minimize:} \quad \frac{1}{2}\sum_{j=1}^{n}||w_j - w_{\mathcal{A}(n)}|| + \gamma \sum_{j\in\mathcal{L}(T)}\sum_{i=1}^{m} L(y_{ij}, x_i, w_j) \tag{2.28}$$

$$L(y_{ij}, x_j, w_i) = \begin{cases} [1 - y_{ij} w_j^T x_i]_+ & \text{support vector machine} \\ \log(1 + \exp(-y_{ij} w_j^T x_i)) & \text{logistic regression loss} \end{cases} \quad (2.29)$$

The loss function used depends on local classifiers being logistic regression or support vector machines (see equation 2.29). The empirical part of the objective function 2.28 is fully decomposable since it only involves the leaves of the hierarchy and there is no interaction between them. However, the interactions through the regularization term obliges to jointly learn the parameters. Moreover, since the interactions between local classifiers are only through parents and children, the optimization of 2.28 can be parallelized. Indeed for a given node of the hierarchy, when the parameters of its children and parents are fixed, the local classifier can be optimized independently of the rest of the parameters. For large hierarchies whose parameters may not fit in memory, this is an attractive feature. This, with the state of the art performances reported by the authors on large scale real world datasets, makes this method very appealing for solving extreme classification problems.

Dissimilarity based dependence modeling: While the similarity based regularization leverages the transferred semantic relatedness information to improve classification, dissimilarity based approaches rely on exploiting specificities of each class to improve local classifiers hence ameliorating the overall accuracy of the hierarchical classifier. An important contribution following this idea is the orthogonal regularization framework by (Xiao et al., 2011). At the heart of their work is the observation that the type of relationships conveyed in a hierarchical structure are generally of the type *generalization/specialization*. This suggests that classes can be better distinguished from their ancestors by their particularities. This observation is in line with a previous remark by (Koller and Sahami, 1997) who point of that when classifying between documents on *sports* and *computer science*, the word computer is a very discriminative feature while in contrast, the two subclasses *system* and *compiler* can be more accurately differenciated by the feature *parsing*.

To account for this fact, (Xiao et al., 2011) propose to add to the classical hinge loss a regularization term encouraging the weight vector of each local classifier f_i to be as much different as possible to those of its ancestors $\{f_j : y_j \in \mathcal{A}(y_i)\}$. For a set of training examples $\{(x_i, y_i), \ldots, (x_m, y_m)\}$ and a tree structured hierarchy, the final hierarchical classification problem can then be formulated as follows:

$$\text{minimize: } \frac{1}{2}\sum_{i=1}^{n} \mathbf{K_{ii}}||w_i|| + \sum_{i=1}^{n}\sum_{j\in\mathcal{P}(i)} \mathbf{K_{ij}}|w_i^T w_j| + \frac{C}{m}\sum_{k=1}^{m}\xi_k \tag{2.30}$$

$$\text{s.t. } \langle w_i, x_i\rangle - \langle w_j, x_i\rangle \geq 1 - \xi_i \quad (\forall j \in \mathcal{S}(i), \forall i \in \mathcal{P}(y_k), \forall k = 1, \ldots, m) \tag{2.31}$$

$$\xi_i \geq 0, i = 1, \ldots, m. \tag{2.32}$$

where $\mathcal{P}(y_k)$ is set of nodes on the path from the root of the hierarchy to the node representing the label y_k. The new elements in this optimization problem (in comparison with the classical multiclass SVM formulation by (Crammer and Singer, 2002)) are shown in the second term of the objective function and the first constraints. The terms $\mathbf{K}_{ij}|w_i^T w_j|$ encourage orthogonality among weight vectors of local classifiers that are on the same path from the root the relevant class by penalizing the absolute values of their inner products. The terms $\mathbf{K}_{ij}$ are non-negative entries of a symmetric matrix $\mathbf{K}$ the authors suggest to choose as in equation 2.33.

$$\mathbf{K_{ij}} = \begin{cases} |\mathcal{C}(i)| + 1 & \text{if} \quad i = j \\ \alpha & \text{if} \quad i \in \mathcal{A}(j) \\ 0 & \text{else} \end{cases} \tag{2.33}$$

where $\mathcal{C}(i)$ is the set of children of node i and α is a positive parameter that can be set to 1 to make the problem convex (Xiao et al., 2011). However, although convex, the problem cannot be directly cast in any known standard conic program optimization (Boyd and Vandenberghe, 2004). Nonetheless, the authors propose

an efficient regularized dual averaging based method (Nesterov, 2009) for tractably solving this problem.

2.3.2.4 Sequential Learning of Models

Pachinko machines are computationally appealing because they allow parallelization without much effort since the local classifiers are totally independent (Bengio et al., 2010, Yang et al., 2003). However, not modeling the dependences at all narrows down the capacity of a hierarchical classifier to reach state of the art performances (Liu et al., 2005b, McCallum et al., 1998). On the other hand, jointly learned models have achieved record breaking performances in many studies (Bengio et al., 2010, Cai and Hofmann, 2004, Gopal and Yang, 2013). However, training them efficiently is challenging because of the number of dependences to account for in an extreme classification context and the difficulty to parallelize the learning process. An intermediate approach is to leverage the available hierarchical information by training local classifiers sequentially. This makes training more feasible and allows parallelization since all the local classifiers at a given level of the hierarchy are independent. There are three main contributions in this line of work: Filter trees (Beygelzimer et al., 2009b) and the refinement procedure (Bennett and Nguyen, 2009) are based on using biased training distributions to learn local classifiers while refined experts (Bennett and Nguyen, 2009) use expert information as additional features. We elaborate more on these methods in the sequel.

Filter Trees (Beygelzimer et al., 2009b) introduced a hierarchical classifier trained in a bottom-up procedure that reduces a given multiclass classification problem into a set of binary problems sequentially. The structure of the hierarchy is a binary tree that can either be learned or randomly generated. The main trick in this algorithm is the bottom up learning process of the local classifiers at internal nodes of the tree. Starting from the leaf nodes, at each internal node v, the local classifier is learned with a training data formed conditionally to the predictions of the local classifiers of the current node's children. Indeed, Only relevant examples that have been correctly classified at all the previous rounds of the subtree rooted at v are involved in the training data used at the node v. In

case of a leaf node, all the relevant data is used. The local training set can then be summarized as: $S_v^{FT} = \{(x_i, y_i) : \exists k \in \mathcal{C}(v) : y_i \in S_k^{FT} \wedge f_k(x_i) > 0\}$. Hence, the training distribution at each node is said to be filtered by the local classifiers of the node's children. Therefore, the Filter Tree is a way of solving the error propagation problem by preventing false positives mainly at the upper levels since the constraint is harder as one goes upper in the hierarchy. Moreover, it comes with an accompanying regret bound showing that it is a consistent reduction.

Although it has theoretically and empirically been shown to be an effective reduction, the Filter Tree has several drawbacks. First, the binary structure of the hierarchy which guarantees logarithmic time inference may not be discriminative enough as shown in (Griffin and Perona, 2008). Moreover, the rule governing the local training set creation can lead to data scarcity problems. Indeed, the set of examples correctly classified by all the local classifiers from the current node to the relevant leaf nodes can be very small. Even worse is when the hierarchy is randomly generated since in this case, the learnability issues at most of the nodes would result in difficult induced problems at internal nodes implying poor local classifiers.

Refinement (Bennett and Nguyen, 2009) has proposed an approach, called *refinement*, also based on biasing the training distribution of local classifiers conditionally to the neighboring nodes' local predictions. Like Filter Trees, their method aims at improving the generalization ability of the hierarchical classifier by reducing the probability of false positive. However, the two approaches are rather different in practice. While the the Filter Trees training procedure is bottom-up, Refinement rests on a top down strategy that employs cross validation to obtain predictions for the examples of the training data. The predicted labels are then used to filter the training data. At a node v, all the examples that have been predicted as belonging to this node by the ancestor's local classifier are involved in the local training set: $S_v^R = \{(x_i, y_i) : f_{\mathcal{A}(v)}(x_i) > 0\}$. This training data can further be decomposed into positive and negative examples. The former are the examples in S_v^R whose actual labels correspond to a leaves of the subtree rooted at v: $S_{v+}^R = \{(x_i, y_i) \in S_v^R : y_i \in l_v\}$. The latter are the complementary of the

positive examples in S_v^R, $S_{v-}^R = \{(x_i, y_i) \in S_v^R : y_i \notin l_v\}$. Using the ancestors' local classifiers to build the training sets at every node aligns the training distribution with what will likely happen at test time. The resulting local classifiers are expected to better distinguish positive examples to propagate down from negatives examples that may arrive at the current node (due to errors in the upper levels) therefore halting their progression.

Due to potential false negative errors at the upper levels of the hierarchy, the local classifiers are prone to overfitting when the refinement procedure is applied in its basic form. Indeed, positive data scarcity (Japkowicz and Stephen, 2002) is an important problem that is here exacerbated by the hierarchical nature of the problem as in Filter Trees. To overcome this difficulty, (Bennett and Nguyen, 2009) propose a slight modification to the original refinement procedure which consist in using as training data at each node v, the union of the distribution created with the ancestor's predictions and the actual distribution. The resulting local training set at a node v writes $S_v^{R+} = \{(x_i, y_i) : (f_{\mathcal{A}(v)}(x_i) > 0) \vee (y_i \in l_v)\}$. This new refinement procedure has been empirically shown to be better than both the refinement approach and the classical Pachinko machines.

Refined Experts Filter Trees and refinement tackle the error propagation problem by learning local linear classifiers which are robust to false positive errors. However, hierarchical classifiers also suffer from false negative errors occurring (mainly) at the upper level internal nodes. Solving this problem requires complex non linear decision surfaces. Although the literature on this latter topic is rich, most of the existing methods (kernel machines, neural networks, etc) are computationally prohibitive in an extreme classification context. In (Bennett and Nguyen, 2009) an approach inspired from *meta-classification and combination of classifiers* (Bennett et al., 2002) is proposed to create non linear local classifiers. At a given node n, their method uses predictions (optionally passed through a calibrated sigmoid) of the children's local classifiers as additional features to the representation of examples for training the current nodes local classifier. The local training set at node v using bottom-up expert information can be written as : $S_v^E = \{(z_i, y_i) : (y_i \in \mathcal{A}(v)); z_i = [x_i; (f_k(x_i))_{k \in \mathcal{C}(v)}]\}$. This idea stems from

TABLE 2.1: Sequential models learning approaches. The key differences between them is the local training set construction and the order in which the models are learned. The pachinko machine is given for comparison.

Algorithms	Local Distribution	Training order
Pachinko	$S_v^P = \{(x_i, y_i) : y_i \in l_{\mathcal{A}(v)}\}$	Independent
Filter Tree	$S_v^{FT} = \{(x_i, y_i) : \exists k \in \mathcal{C}(v) : y_i \in S_k^{FT} \wedge f_k(x_i) > 0\}$	Bottom up
Refinement	$S_v^R = \{(x_i, y_i) : f_{\mathcal{A}(v)}(x_i) > 0\}$	Top down
Refinement+	$S_v^{R+} = \{(x_i, y_i) : (f_{\mathcal{A}(v)}(x_i) > 0) \vee (y_i \in l_v)\}$	Top down
Expert Information	$S_v^E = \{(z_i, y_i) : (y_i \in \mathcal{A}(v)); z_i = [x_i; (f_k(x_i))_{k \in \mathcal{C}(v)}]\}$	Bottom up

the observation that the union of linear surfaces generally results in a non linear surface (Klivans and Sherstov, 2006). As in refinement, the predictions on the training examples are obtained using cross validation. However, the process starts here from the leaves and is repeated until the root node is reached. At each round, the bottom-up propagated expert information is a strong signal for the upper classifier to pull a given test instance down the correct subtree hence avoiding false negative errors. Combining refinement with the use of expert information results in a two step procedure which consist in a bottom-up procedure aiming at reducing false negatives at test time, followed by a top-down pass refinement that prevents false positive. The overall process, called *refined experts* (Bennett and Nguyen, 2009), yields state of the art performances compared to classical approaches such as hierarchical SVMs (HSVM) and flat One Versus Rest (OVR).

Table 2.1 summarizes the sequential approaches to learning local classifiers given a hierarchical structure. All these methods build on the same idea of biasing the local training distribution at the different nodes of the hierarchy to fight the error propagation problem. Even though refinement and refined experts can arguably be more accurate than Filter Tree, the additional training time required for cross validation is a potential bottleneck. Nonetheless, all these methods are interesting alternatives to jointly learned models in case one has limited resources for training.

2.3.3 Joint Learning of Models and Hierarchical Structure

To achieve the goal of efficient and accurate classification using hierarchical methods, one needs both a discriminative class hierarchy (T^*) and accurate local classifiers (F^*). Various methods have proposed to satisfy these two requirements independently. In general, a global solution (T^*, F^*) can be obtained by first learning a class hierarchy (for example using the method (Bengio et al., 2010) or (Marszalek and Schmid, 2008)) before learning its associated local classifiers (Bengio et al., 2010, Bennett and Nguyen, 2009, Gopal and Yang, 2013). While this sequential strategy has yielded good performances in previous studies (Bengio et al., 2010), it can be suboptimal. Indeed, the difficulty of the classification problems locally induced at the nodes of the hierarchy governs the accuracy of the local classifiers which is the key factor of the hierarchical classifier's global performance. Therefore, jointly learning the class hierarchy and its associated local classifiers would result in a better solution since it deals with the interplay between the two components of the hierarchical classifier. This idea has arguably first been proposed in (Beygelzimer et al., 2009a) where the authors learn online a hierarchical structure with local probability estimators. Even though their approach can be used for classification purposes (by greedily traversing the tree), it is first intended for conditional probability estimation. More recently, two successful approaches have been proposed (almost simultaneously) to tackle the joint hierarchical classifier learning challenge (Deng et al., 2011, Gao and Koller, 2011a). In the rest of this section, we present the respective formulation of these approaches before discussing their similarities and specificities.

2.3.3.1 Fast and Balanced Hierarchies (Deng et al., 2011)

The fast and balanced hierarchical classifier is learned by processing one node at a time. At each node s the learning algorithm searches for a an optimal split of the local set of classes l_s together with the parameters of the local classifiers f_s. This process can be formulated as local classifier's accuracy maximization subject to efficiency constraints. Therefore, the feasible set of parameters of the classifier f_s are required to satisfy the efficiency constraints. For a given example x, The

efficiency measure considered here at node s is called *ambiguity* and is defined as the size of the label set of the child $c \in \mathcal{C}(s)$ that the example follows relative to its parent's size $|l_s|$.

Formally, at the current node s, let Q be the specified branching factor (number of children per node) and $K = |l_s|$. The splits at this node can be represented by a *partition matrix* $P \in \{0,1\}^{Q \times K}$ in which $P_{qk} = 1$ if class k appears in the label set corresponding to child q l_q, and $P_{pk} = 0$ otherwise. To each child $q \in \mathcal{C}(s)$ corresponds a binary classifier. At node s, the set of parameters of the children's binary classifiers is represented by a matrix $\mathbf{w}$ whose columns represent the weight vectors $(w_i)_{1 \leq i \leq Q}$.

At node s, for a given example (x, y) with $y \in l_s$, let $\hat{q} = \text{argmax}_{q \in \mathcal{C}(s)} f_q(x)$ be the winning child. Given the parameters $\mathbf{w}$ and P, the loss at the current node s is $L(\mathbf{w}, x, y, P) = 1 - P_{\hat{q}y}$. When the set of partitions is held fixed, optimizing for $\mathbf{w}$ reduces to a multiclass classification problem. Therefore, the following convex relaxation to L is proposed:

$$\tilde{L}(\mathbf{w}, x_i, y_i, P) = \max\{0, 1 + \max_{q \in A_i, r \in B_i} \{w_r^T x_i - w_q^T x_i\}\} \tag{2.34}$$

Solving this problem encourages the local classifier to give a higher scores to the children whose set of labels contain y_i ($q \in A_i$) compared to those whose that do not contain it ($r \in B_i$). Regarding the efficiency of the hierarchy, it is enforced through *ambiguity constraints* i.e the average proportion of classes pruned away at every step when traversing the hierarchy top-down for classification. If the partitions are balanced, $(Q-1)\cdot(K/Q)$ classes are pruned every time a classification decision is made at a node of the hierarchy. For a given example (x, y) and the parameters $(P, \mathbf{w})$, *ambiguity* is formulated as:

$$A(\mathbf{w}, x, P) = \frac{1}{K} \sum_{k=1}^{K} P(\tilde{q}, k) \tag{2.35}$$

The final global optimization problem consisting into local accuracy maximization subject to efficiency constraints summarized below:

$$\begin{aligned} \underset{\mathbf{w},P}{\text{minimize}} \quad & \lambda||\mathbf{w}||^2 + \frac{1}{m}\sum_{i=1}^{m} L(\mathbf{w}, x_i, y_i, P) \\ \text{subject to} \quad & \frac{1}{m}\sum_{i=1}^{m} A(\mathbf{w}, x_i, P) \leq \epsilon \\ & P \in \{0,1\}^{Q\times K} \end{aligned} \tag{2.36}$$

This algorithm proposed by the authors to solve this problem alternatively minimizes the classification error and the ambiguity at each node of the hierarchy. The integrity constraints of the partition matrix's entries make this problem NP-hard. Nonetheless, solving the continuous relaxation and rounding the resulting solution is theoretically proven to yield good performances. Overall, the fast and balanced hierarchy is proven to be superior to the hierarchical classifiers learned independently from the local classifiers (Bengio et al., 2010) and One-versus-Rest flat classifiers.

2.3.3.2 Relaxed Discriminant Hierarchies (Gao and Koller, 2011a)

The relaxed discriminative hierarchical classifier is another recent approach that learns the hierarchical structure and the local classifiers jointly. The general learning strategy also consists in processing one node of the hierarchy at a time by splitting the local label set and learning a local classifier. As for the fast and balanced hierarchy (Deng et al., 2011), the structure is a directed acyclic graph (DAG) to fight the error propagation problem (Bennett and Nguyen, 2009). The two main ingredients in this approach are the way the local induced problems are created and the the way efficiency is enforced during learning.

To achieve low classification error at each node s, the locally induced problems are *binary*. Moreover, they are relaxed to avoid the requirement for complex decision surfaces at the top of the hierarchy. That is, at a given node s, not all the classes of l_s are involved in the binary induced problem. Indeed, the more classes involved in the binary induced problem at node s, the more difficult the problem is. Therefore,

the authors propose to consider only a subset of classes that can be discriminated easily (the set of positive classes is denote S_y^+ and the set of negative ones is S_y^-) and ignore the other classes S_y^0. Hence, given a training data $\{(x_i, y_i)\}_{i=1}^m$ where $y_i \in \mathcal{Y} = \{1, \ldots, n\}$, the local training set is split into $S_x^+ = \{x_i : y_i \in S_y^+\}$ and $S_x^- = \{x_i : y_i \in S_y^-\}$. To identify the group each class belongs to, coloring variables μ_k taking their values from $\{-1, 0, +1\}$ are introduced.

Trivial solutions such as considering a very large set of classes as belonging to S_y^0 can give good solutions since the induced problems become very simple (the extreme case is when $|S_y^+| = |S_y^-| = 1$). However, the number of pruned classes at every step is very small and classification complexity becomes almost linear. Moreover, the height of the hierarchy is minimized when the partitions S_y^+ and S_y^- are balanced at each node. Therefore it is necessary maintain non-trivial and balanced partitions at every node of the hierarchy. To achieve these goals, the following optimization problem is solved at each node:

$$\begin{aligned}
\underset{\mathbf{w},b,\{\mu_k\},\{\xi_j\}}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}|\mu_{y_i}|\xi_i - A\sum_{i=1}^{m}|\mu_{y_i}| \\
\text{subject to} \quad & \mu_i \in \{-1, 0, +1\}, \forall k \in \mathcal{Y} \\
& \mu_{y_i}(\mathbf{w}^T x_i + b) \geq 1 - \xi_i, \forall i \\
& \xi_i \geq 0, \forall i \\
& -B \leq \sum_{k=1}^{|\mathcal{Y}|}\mu_k \leq B \\
& \sum_{k=1}^{|\mathcal{Y}|}\mathbf{1}\{\mu_k > 0\} \geq 1 \text{ and } \sum_{k=1}^{|\mathcal{Y}|}\mathbf{1}\{\mu_k < 0\} \geq 1
\end{aligned} \tag{2.37}$$

The first components of the objective function, $1/2||\mathbf{w}||^2 + C\sum_{k=1}^m |\mu_{y_k}|\xi_k$ correspond to a classical binary SVM problem. The third component, $-A\sum_{k=1}^m |\mu_{y_k}|$, prevents trivial solutions by encouraging most classes to be involved in the binary problem ($|\mu_k| = 1$). Compared to a classical binary SVM formulation, the last two constraints are unusual. The third constraint is intended to enforce balance splits while the last one requires each split contain at least one positive class and one negative class (non-triviality). Given a fixed partition, the problem reduces to a

binary SVM while given a parameter vector $\mathbf{w}$, finding a good partition is done by solving a coloring problem. An alternating method is hence proposed to solve the global optimization problem. The authors also provide theoretical guarantees regarding the generalization performance of the algorithm. In practice, convincing results are presented among which improvements compared to the hierarchy learning approaches by (Marszalek and Schmid, 2008) and the One-versus-Rest flat baseline.

2.3.3.3 Discussion

The two approaches presented above are very similar in spirit. They draw from a set of intuitions and best practices resulting from previous studies (Bennett and Nguyen, 2009, Griffin and Perona, 2008, Marszalek and Schmid, 2008). In hierarchical classification, efficiency (speed) mainly depends on the type of hierarchy chosen which is generally either a tree or a DAG. However, DAGs provide better discriminative capabilities since they suffer less from the error propagation problem (Marszalek and Schmid, 2008). Moreover, the efficiency loss compared to trees can be rather small (Gao and Koller, 2011a). Therefore, both of the methods are designed to produce DAG structured class hierarchies. Regarding the accuracy of hierarchical classifier, it has been shown that it can be greatly improved when the dependencies between the local classifiers are modeled (Bengio et al., 2010, Bennett and Nguyen, 2009, Cai and Hofmann, 2004, Gopal and Yang, 2013) since it helps solving problems such as the scarcity of labelled data at the leaf nodes. Even though the local classifiers are not jointly learned in these methods conversely to (Bengio et al., 2010, Cai and Hofmann, 2004), the training data at each node is filtered by the local classifiers of its ancestors. As shown in (Bennett and Nguyen, 2009, Beygelzimer et al., 2009b), this is a good way of encoding dependencies between the local classifiers of a given hierarchy.

Despite all their similarities, there are fundamental differences between the relaxed discriminant hierarchies (Gao and Koller, 2011a) and the balanced hierarchies by (Deng et al., 2011). First, (Deng et al., 2011) allow the k-way ($k \geq 2$) DAG structured class hierarchies while the formulation of (Gao and Koller, 2011a) restrict the branching factor of the internal nodes to 2. While the former may lead

to more efficient structures because at each step a larger subset of candidate classes can be pruned away, we believe the latter is a better option in terms of accuracy because only one binary classification problem is solved at each node. Moreover, in relaxed discriminant hierachies (Gao and Koller, 2011a), all the classes are guaranteed to have a corresponding leaf node in the class hierarchy. Such bijection does not exist in (Deng et al., 2011) where at each step of the hierarchy building process, it is legal to have a class that is not associated to any sub-cluster. When the final performance evaluation measure is the 0/1 loss, this does not have much impact since most of the classes that are *forgotten* will correspond to small classes. However, it can be dramatic if measures such as the *Macro-F1* are used because this type of performance evaluation measure considers all the class equally regardless of their size.

2.3.4 Conclusion

The rationale underlying the best among the presented solutions for improving hierarchical classification is to accommodate with the semantics imposed by a given hierarchy (Cai and Hofmann, 2004) or to encode the dependencies between the nodes of a learned hierarchy (Bengio et al., 2010). To that end, learning is generally achieved with various types of constraints aiming at modeling the dependences between the local classifiers of the hierarchy. All these methods are empirically proven to give good performances but the reason of this success is not fully understood. As previously stated, authors generally agree on the four main problems to solve in hierarchical classification (Bennett and Nguyen, 2009) : (1) the need for an efficient hierarchical structure, (2) the error propagation problem, (3) the requirement for complex decision surfaces at the top the hierarchy and (4) the labelled data scarcity at the leaf nodes. There is clearly an interplay between these problems. For instance, when a hierarchy is learned such that the problems induced at its nodes are more natural (hence more easily solvable), the need for complex decision surfaces at the top of the hierarchy is attenuated (Griffin and Perona, 2008, Marszalek and Schmid, 2008). Similarly if an efficient structure such as a DAG is chosen it also improves the accuracy since it contributes in solving the error propagation problem (Deng et al., 2011, Gao and Koller, 2011a).

Unfortunately, even though the authors usually mention the specific problem they intend to solve at first to improve hierarchical classification performances, the proposed methods are rarely accompanied with an analysis of how it impacts the other three problems of hierarchical classification. In such situation, performance improvements can rapidly be acknowledged to the wrong facts as the impact of the side effects on the final performances measure can be superior to that of solving solely the initially considered problem. Thus, we believe more ablation studies as in (Bennett and Nguyen, 2009) would be of great help towards a better understanding of the nature of the hierarchical classification problem and also for classification of the solutions proposed so far according to the specific problem they solve and their potential side effects.

Chapter 3

Extreme Single Label Classification with Compact Ouput Coding

3.1 Introduction

Scalable approaches to extreme single label classification are divided into flat and hierarchical methods as discussed in the previous chapter. Hierarchical approaches have demonstrated their ability to reduce inference complexity up to $O(\log L)$ (L being the number of labels) while maintaining competitive performances compared to the traditional one-versus-rest classifier (Bengio et al., 2010, Deng et al., 2011, Gao and Koller, 2011b). This is achieved by either learning a hierarchy or by using an existing one as it is case in most extreme classification problems. However, hierarchical approaches require training and storing as many classifiers as the number of nodes in the given hierarchy. This can result in a computational and memory burden specially when the input space in very high dimensional which is common in extreme classification problems (the wikipedia dataset used in the LSHTC challenges[1] has more than 300K features). Alternatively, some flat approaches such as Embedding methods (Bengio et al., 2010, Weinberger and Chapelle, 2008) and

[1] http://lshtc.iit.demokritos.gr

Error Correcting Output Codes (Dietterich and Bakiri, 1995) have better training and inference complexity than OVR. While Embedding methods can exploit existing hierarchical information to improve classification performance (Weinberger and Chapelle, 2008), they are slower at inference (for a same code size) than binary ECOCs which can be speed up thanks to existing fast hamming distance computing procedures (Pappalardo et al., 2009). Unfortunately, binary codes in ECOCs are generally randomly chosen and hence do not leverage existing hierarchical information to improve performances. Moreover, large code size are often required in order to reach competitive performances.

In this chapter, we present an approach that takes the best of both worlds. It exploits existing hierarchical information to learn compact binary codes. Exploiting the hierarchical information improves performances while the use of compact binary codes allows fast inference as with Error Correcting Output Codes. The approach we develop relies on first learning binary class codes using a similarity information between classes, a class will then be represented as a l-dimensional binary code with values in $\{-1, +1\}$, and second in training l binary classifiers, each will predict one bit of the class code. The *dichotomizer* for the j^{th} bit of the code will be trained to distinguish between the samples of all classes whose j^{th} bit is 1 and those whose j^{th} bit is -1. A test example will then be categorized according to a simple nearest neighbour rule between the code computed for this example and class learned codes. The novelty of this two step strategy is an efficient procedure for learning compact binary class codes of size l such that $B << L$ where L stands for the number of classes. The size of the learned distributed representation of the classes may be set so as to achieve a compromise between complexity and accuracy.

We first present the method used to learned the distributed representation of the classes in section 3.2. We discuss its relationship to Error Correcting Output Codes and propose a study of the method's complexity. The experimental section 3.3 follows. There, We show that the code size required for reaching OVR performance scales sub-linearly with the number of classes and that increasing the complexity of the method (i.e. B) allows outperforming OVR. We also provide an experimental

comparison, with respect to performance and runtimes, of our method with baselines, including OVR, on datasets up to 10 000 classes built from the 2010 Large Scale Hierarchical Text Classification challenge datasets[2]. Moreover, we investigate the ability of the proposed approach for *zeroshot learning* (Palatucci et al., 2009) which is the problem of discriminating between classes labels for which no examples were encountered during training. We show that providing the similarity information for new classes allows recognizing samples from theses classes even in the case when no training samples are available.

3.2 Learning Distributed Representation of Classes (LDR)

3.2.1 Principle

We aim here at building a method that allows, both fast inference and high accuracy. To reach this goal we propose a method called Learned Distributed Representation (LDR) that first learns binary low dimensional class codes, then uses binary classifiers to learn each bit of the codes, as in ECOC.

A key issue is to take into account the available relationships between classes (e.g. a hierarchical or a graph organization of classes). We propose to compute low dimensional binary class codes that reflect these relationships. In order to do that we first represent a class as a vector of similarities between the class and all other classes, $\mathbf{s}_i = [s(C_i, C_1), ..., s(C_i, C_L)]$ (see section 3.3 for an example). Different similarity measures may be used. It may be computed from a hierarchy of classes or from a similarity between samples of the two classes. Then, we learn short class codes that reflect these relationships between classes, by transforming these high k-dimensional representations of classes ($\mathbf{s}_i$) into lower l-dimensional codes ($\mathbf{h}_i$) via a dimension reduction algorithm. This step is explained in details in section 3.2.2. Once low dimensional (say B-dimensional, with $B << L$) binary class representations are learned, we train l binary classifiers, one for every bit.

[2] http://lshtc.iit.demokritos.gr

The binary classifier for the j^{th} bit is a dichotomizer that is learned to separate samples of all classes whose class code has the j^{th} bit set to 1 from the samples of all classes whose class code has the j^{th} bit set to -1. All these binary classifiers are then learned with all training samples from all classes.

Finally at test time, when one wants to decide the class of an input sample x, we use the B classifiers on x to compute a B-length binary word $\mathbf{m} = (m_1, ..., m_B)$ which is compared to the L class codes $\{\mathbf{h}_i, i = 1..L\}$ to find the nearest neighbor.

3.2.2 Learning Compact Binary Class-codes

We propose to learn compact class codes with autoencoders which have been widely used for feature extraction and dimensionality reduction (Vincent et al., 2008). Among many existing dimension reduction methods the advantage of autoencoders lies in the flexibility of the optimization criterion that allows us including additional terms related to class codes separation. An autoencoder is trained by minimizing a squared reconstruction error between the input (here a class representation $\mathbf{s}_i$) and its reconstruction at the output of the autoencoder, $\widehat{\mathbf{s}}_i$. It may be viewed as an encoder (input $\rightarrow$ hidden layer) followed by a decoder (hidden $\rightarrow$ output layer). Usually it is required that encoding and decoding weights are tied (Vincent et al., 2008), both for linear and non linear encoders, so that if $\mathbf{w}$ is the coding matrix, $\mathbf{w}^T$ is the decoding matrix. We used this strategy here. Training an autoencoder writes (omitting bias terms):

$$\underset{W}{\operatorname{argmin}} \sum_{i=1}^{L} ||\mathbf{s}_i - \mathbf{w}^T \times f(\mathbf{w} \times \mathbf{s}_i)||^2 \tag{3.1}$$

where $||.||$ is the euclidean distance. The activation function in hidden units f may be a linear function, then the projection learned by the autoencoder is similar to the one learned by a principal component analysis. One can expect to learn more interesting features by using nonlinearities on hidden units, using sigmoid or hyperbolic tangent activation functions (in our implementation, we use hyperbolic tangent activation function hidden units). To perform dimensionality reduction

one uses a narrow hidden layer which forces to learn non trivial regularities from the inputs, hence interesting and compact codes on the hidden layer. The vector of activation of hidden units is the learned encoding function. Here the new class code for class C_i is then $\mathbf{h}_i = f(\mathbf{w} \times \mathbf{s}_i)$.

Ideally, new class codes should satisfy two properties. First, similar classes (according to the cost-sensitive information and/or to similar examples) should have close codes $\mathbf{h}_i$. Second, class codes for any pair of classes should be significantly different to ensure accurate classification at the end. The first property is naturally satisfied since an autoencoder actually learns hidden codes that preserve distances in the original space. Next, to ensure minimal separation between class codes we propose to look for a solution of the following constrained problem:

$$\begin{aligned} &\underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{L} ||\mathbf{s}_i - \mathbf{w}^T \times f(\mathbf{w} \times \mathbf{s}_i)||^2 \\ &\text{s.t. } \forall (i,j), i \neq j : ||f(\mathbf{w} \times \mathbf{s}_i) - f(\mathbf{w} \times \mathbf{s}_j)|| \geq b \end{aligned} \tag{3.2}$$

The constraints are inspired from margin based learning and yield to maximize the distance between any pair of class codes up to a given threshold b. We solve this optimization problem by stochastic gradient descent using the unconstrained regularized form:

$$\begin{aligned} \underset{\mathbf{w}}{\operatorname{argmin}}\ \alpha &\sum_{i=1}^{L} ||\mathbf{s}_i - \mathbf{w}^T \times f(\mathbf{w} \times \mathbf{s}_i)||^2 \\ &+ \beta \sum_{i,j=1}^{k} \max(0, b - ||f(\mathbf{w} \times \mathbf{s}_i) - f(\mathbf{w} \times \mathbf{s}_j)||) \\ &+ \frac{\lambda}{2} ||\mathbf{w}||^2 \end{aligned} \tag{3.3}$$

where α and β weight the respective importance of the reconstruction error term and of the margin terms, and $||\mathbf{w}||^2$ is a regularization term. Note that α, β, and b (which tunes the margin between two class codes) are set by cross validation.

We learn the autoencoder using stochastic gradient descent by iteratively picking two training samples i and j at random and making a gradient step. Figure 3.1 illustrates the training process which recalls somehow Siamese architectures used in the past for vision tasks (Bromley et al., 1993). At the end, in order to get

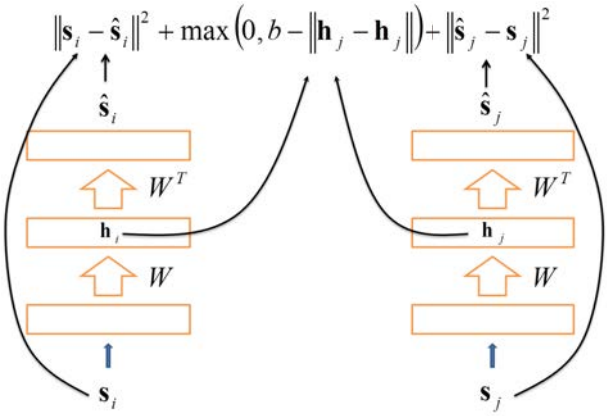


FIGURE 3.1: Learning the autoencoder from pairs of input samples (here α and β are considered equal to 1). See Algorithm 2 for details.

binary class codes, we threshold the learned real valued class codes. This means that the j^{th} component of all class codes $\mathbf{h}_i$ are set to $\mathbf{h}_i(j) = -1$ if $\mathbf{h}_i(j) < \theta_j$, and $\mathbf{h}_i(j) = +1$ otherwise. The threshold value θ_j is chosen so that the prior probability of the j^{th} bit of a class code be $+1$ is equal to 0.5, and this is done by setting θ_j to the median of $\{\mathbf{h}_i(j)|i = 1 \dots B\}$. Although this cut-off it is not learned to optimize classification accuracy, it should be noted that it is defined according to the usual property in ECOC (firing with probability 0.5). Also since similar classes should have close class codes, it is expected that the obtained two class classification problem (i.e. for the j^{th} bit of class codes, separating samples of all classes with $\mathbf{h}_i(j) = +1$ from the samples of all classes with $\mathbf{h}_i(j) = -1$) should be easier to solve than any random two class problem as those defined in traditional ECOC. We will come back to this point in the next section. Algorithm 2 describes the whole algorithm.

Algorithm 2: Learning Compact Binary Class Codes

Input : similarity vectors $\{\mathbf{s}_i\}$;
Hyper parameters $\alpha, \lambda, \beta, b$;
repeat
  Pick randomly two samples $(\mathbf{s}_i, \mathbf{s}_j)$;
  Make a gradient step : $\mathbf{w} = \mathbf{w} - \epsilon \partial L_{\mathbf{w}}(\mathbf{s}_i, \mathbf{s}_j)/\partial \mathbf{w}$;
  with: $L_{\mathbf{w}}(\mathbf{s}_i, \mathbf{s}_j) = \frac{1}{2} \sum_{k \in \{i,j\}} \alpha ||\mathbf{s}_k - \mathbf{w}^T \times f(\mathbf{w} \times \mathbf{s}_k)||^2 + \lambda ||\mathbf{w}||^2 +$
  $\beta \max(0, b - ||f(\mathbf{w} \times \mathbf{s}_i) - f(\mathbf{w} \times \mathbf{s}_j)||)$
until *convergence criterion is met*;

3.2.3 Relations to ECOC

Because each element in the class codes has probability 1/2 of being either +1 or −1, our method bares some similarities with the standard dense random ECOC. However, there are two fundamental differences.

The first difference is that by construction, our learned distributed representation is intended to have a reduced tree induced loss compared to randomly generated methods because the autoencoder projects classes that are close in the hierarchy in the same area of the latent space. The second difference, which is somehow related to the first one, is that the binary classification problems induced by the learned class codes should be easier than in random ECOC. Indeed, since similar classes should have close class codes, it is likely that for similar classes most bits are equal. This means that a particular dichotomizer is trained with samples for class +1 and for class -1 that are more homogeneous than if the partitioning of classes was random, as in traditional ECOCs. At the end, if dichotomizers reach higher accuracy, the overall accuracy of the multiclass classifier should also be higher.

An ECOC coding scheme closer to our method is the discriminative ECOC (DECOC) which learns a discriminative coding matrix by hierarchically partitioning the classes according to a discriminative criteria (Escalera et al., 2010). The hierarchy is built so as to maximize the mutual information between the data in each partition and the corresponding labels. Our method differs from this in that we are seeking codewords having a sub-linear dependency on the number of classes L while the DECOC method creates codewords of length $L - 1$.

3.2.4 Training and inference complexity

We focus here on complexity issues with respect to the number of classes L, the number of training samples N, the dimension of samples d, and the length of the learned class codes B. Let us denote by $C_T(N)$ the complexity of training one binary classifier with N training samples, and by C_I the complexity of inference for a binary classifier. All complexities in the following will be expressed as a function of C_T and C_I.

We start with our method. Training consists in learning the class codes of length B, then in learning B classifiers. Learning class codes is done by gradient descent whose complexity depends on the number of iterations. Yet since class codes are binarized at the end, one can expect that the method will not be very sensitive to accurate convergence of the autoencoder and one can reasonably assume a fixed and limited number of iterations $\mathcal{I}$ so that learning the autoencoder requires $O(\mathcal{I} \times L^2 \times B)$ (B iterations with L samples every iteration whose forward and backward pass costs roughly $O(L \times B)$). Next, learning the B binary classifiers requires $O(B \times C_T(N))$. At the end training complexity is in $O(\mathcal{I} \times L^2 \times B + B \times C_T(N))$. Inference consists in finding the class code which is most similar (wrt. Hamming distance) to the output code computed for this input sample. Computing the output code requires using the l classifiers, hence $O(l \times C_I)$. Next, using fast nearest neighbour search methods such as ball trees or kd-trees for finding the closest class code may be done (in practice) in $O(\log L)$ comparisons (Bentley, 1975), where each comparison costs $O(B)$. Overall, the inference complexity is then $O(B \times (\log L + C_I))$.

We compare these costs to those of the OVR method which is the most accurate technique for large scale classification (Bengio et al., 2010) (see Table 3.1). Training in OVR method requires $O(L \times C_T(N))$ since one uses k classifiers that are all trained with all training samples, while inference requires $O(L \times C_I)$.

It clearly appears from this discussion that OVR does not extend easily to VLC due to its inference complexity that scales linearly with the number of classes. Compared to these baselines, our method exhibits interesting features. As we will argue from experimental results, it may outperform OVR for $B << L$ and the

TABLE 3.1: Comparison of training and inference complexity for our method and for standard methods, OVR and ECOC, as a function of the number of classes L, the dimension of the data d, the size of the class codes l, the learning complexity of a binary classifier with N training samples $C_T(N)$, the inference complexity of a binary classifier C_I, and the number of training iterations I of the autoencoder (LDR method).

	Training	**Inference**
OVR	$O(LC_T(N))$	$O(LC_I)$
ECOC(B)	$O(BC_T(N))$	$O\left(BC_I + B\log L\right)$
LDR(B)	$O\left(B\mathcal{I}L^2 + BC_T(N)\right)$	$O\left(BC_I + B\log L\right)$

minimal length B for such a behavior seems to scale strongly sublinearly with L. Furthermore although the training complexity includes a term in $O(L^2)$, it must be clear that in experimental settings such as the ones we investigate in this paper (large number of samples and high dimensionality), the overall training complexity in $O\left(B\mathcal{I}L^2 + BC_T(N)\right)$ is dominated by the second term $O(BC_T(N))$.

3.3 Experiments

We performed experiments on three large scale multi-class single label datasets. The proposed method (LDR) is compared to two coding methods, spectral embedding (SPE) and traditional error correcting output coding (ECOC), and to a standard OVR baseline. We first present the datasets, then we explain our experimental setup and finally we present results and analysis.

3.3.1 Datasets

We used datasets with respectively 1000, 5000 and 10000 classes. Each dataset was created by randomly selecting the corresponding classes from a large scale dataset released for the first PASCAL large scale hierarchical text classification challenge[3]. This dataset was extracted from the open Mozilla directory DMOZ (www.dmoz.org). The classes are organized in a tree hierarchy, classes being at

[3]http://lshtc.iit.demokritos.gr

the leaves of the hierarchy and internal nodes being not instantiated classes. Hierarchies are of depth 5.

The documents were provided as word counts, and then transformed into normalized TF/IDF feature vectors. Considering that for large multi-class text classification every new class is likely to bring specific new words, we did not performed any feature selection although all datasets have very high dimensional feature spaces.

Statistics of the datasets are detailed in Table 3.2. Each dataset is split into training, validation and testing sets (see Table 3.2).

We exploited a similarity measure between classes i and j, which is defined as a function of the distance $d_{i,j}$ between the two classes in the hierarchy measured by the length of the shortest path in the tree between the two classes: $\mathbf{s}_i(j) = s(C_i, C_j) = \exp(-d_{i,j}^2/2\sigma^2)$. The tree path distance between two classes is also used in the tree loss used as a classification measure in section 3.3.3. We systematically used $\sigma = 1$ in our experiments.

TABLE 3.2: Statistics of the dataset used in the experiments

Statistics	1K classes	5K classes	10K classes
Nb. training docs	8119	36926	76417
Nb. validation docs	3005	13855	28443
Nb. testing docs	3006	13771	28387
Nb. features	347 255	347 255	347 255

3.3.2 Experimental setup

Three strategies were used as baselines: OVR, random ECOC and a Spectral Embedding technique.

Besides ECOC classifiers, we also compared our method to a spectral embedding technique (SPE) which can be used for learning class codes from a similarity matrix and is an alternative to our auto-associator method. Spectral embedding is widely used as a preprocessing step before applying k-means in clustering applications.

It has also been used recently for hashing and we exploit a similar idea here. In (Weiss et al., 2008) the authors propose to embed the data for fast retrieval by binarizing the components of the eigenvectors of the similarity matrix Laplacian. This process aims at mapping similar examples in the same regions of a target space. The training complexity of the method is $O(L^3 + BC_T(N))$, which is much larger than LDR or ECOC, and is due to the high complexity of the eigen-decomposition. This method is similar in spirit to LDR and ECOC and is a natural candidate for comparison. The classes here play the same role as data do in spectral hashing.

We use logistic regression as a base classifier (dichotomizers) for all methods, but any other binary classifier could be used as well. The binary classifiers were trained with a regularization parameter selected from $\lambda \in \{0.001, 0.0005, ..., 10^{-6}\}$ using the validation set.

To train random ECOC classifiers, for a given code length B and a number of class L, we generated several $L \times B$ matrices and discarded those having equal or complementary rows. We then used the coding matrices with best error correcting property (the top 25 matrices for 10^3 classes and the top 10 for $5 * 10^3$ and 10^4 classes) to train an ECOC classifier. Then we kept the model that reached the best performance on the validation set for evaluation on the test set.

We compare the methods using accuracy and tree induced loss which is defined as the average of the length of the shortest path in the hierarchy between the correct class and the predicted class. The tree induced loss measures the ability of the classifier to take into account the hierarchical nature of the classification problem, and the class proximity according to this metric. A low tree loss means that confusions are made between neighboring classes, while a high tree loss signifies that confusions occur among distant classes.

3.3.3 Comparison of the methods

We investigate here the behavior of the different methods on the three datasets and explore how the performance evolves with respect to the class code length.

Comparisons with all methods are performed on the $1K$ and $5K$ classes corpora, while on the larger $10K$ classes dataset, only OVR vs LDR were tested. Figure 3.3 reports accuracies on the first two datasets for code length in {200, 300, 400, 500, 600}. First it can be seen that LDR outperforms systematically the two other coding methods (SPE and ECOC) whatever the dataset, and whatever the class code length. Second, the performance of the three coding methods (LDR, SPE and ECOC) increases, with some fluctuation, with the code length. A higher code is needed when the number of classes increases. This behavior is intuitive. Finally one can see that LDR reaches and even exceeds the performance of OVR on these two datasets, while ECOC and SPE stay under the performance of OVR, even when increasing the code length B.

Table 3.3 compares the different methods using their best accuracy score[4], and the corresponding tree induced loss on the same two datasets. It can be seen that the best performance of the different methods are quite close, LDR being systematically higher and providing a clear speedup wrt OVR. For example, for 1 000 classes, with a code length of 200 LDR achieves an accuracy of 67.49% while OVR's accuracy is 66.50%. In this case, the number of classifiers used by the OVR method is 5 times that of LDR.

We come back to our previous observation that LDR is consistently better than random error correcting output coding (ECOC) (Figure 3.3), which holds whatever the code length. Our main explanation of this phenomenon is that the binary problems are probably easier to solve with LDR. It has been observed since the early use of ECOCs (Dietterich and Bakiri, 1995) that the dichotomies induced by the codes where more difficult to solve than the initial OVR dichotomies. Here, neighbour classes in the tree, are forced to have similar codes. The data for these classes are often closer one to the other than that of distant classes, so that similar inputs will most often be required to be classified similarly. On the opposite, classical ECOCs where codes are designed at random do not share this property. To investigate this, we compared the mean accuracy of the binary classifiers induced by our method to the mean accuracy of classifiers in a random ECOC scheme 3.3. The mean accuracy remains between 72% and 75% for LDR while it is almost

[4]For each method, one uses the parameterization, including the value of l, leading to the best score.

constant at about 69% for ECOC when the code size varies between 64 and 1024. This confirms the hypothesis that learned dichotomizers induce easier problems as depicted in figure. Also we think that the learning criteria of the autoencoder helps creating better class codes than those produced by the spectral embedding method.

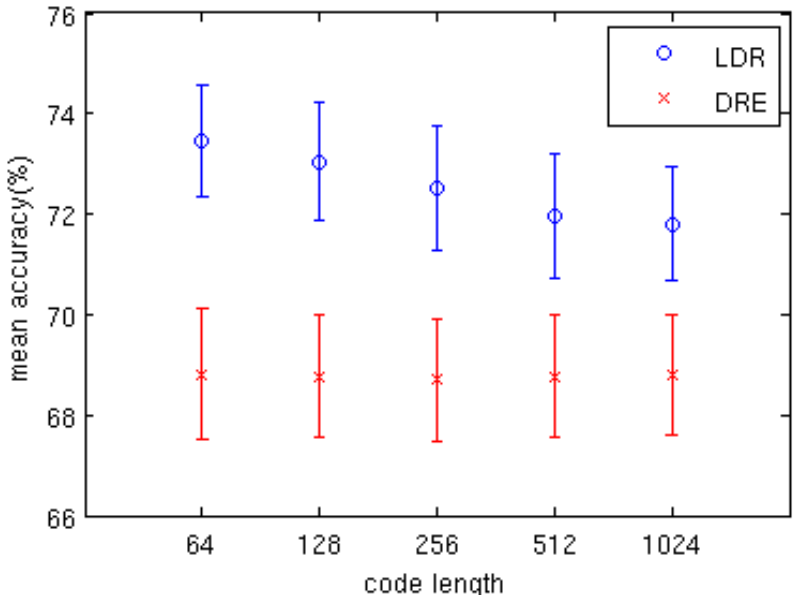


FIGURE 3.2: Comparing the mean accuracy of dichotomizers for binary problems induced by the learned distributed representation and those induced by random ECOCs on the $1K$ dataset with various code size. The binary problems induced the learned representation are easier

At last we compare LDR and OVR on classification tasks with up to 10 000 classes. Figure 3.4 shows the performance of LDR vs OVR for the three datasets ($1K$, $5K$ and $10K$ classes) for a code length of size 500. LDR outperforms OVR whatever the number of classes. Speedup are more and more important as the number of classes increases. For $10K$ classes LDR achieves an accuracy of 36.81% (with a code length of 500) while the OVR's performance is 35.20%. This performance is achieved while using 20 times less classifiers than the number of classes. This corresponds to a speedup of 46 wrt OVR (measured by runtimes). Such a speedup is not only due to the smaller number of classifiers used by LDR, but also to fast bitcounts routines that exploit the binary representation of codes for nearest neighbour search.

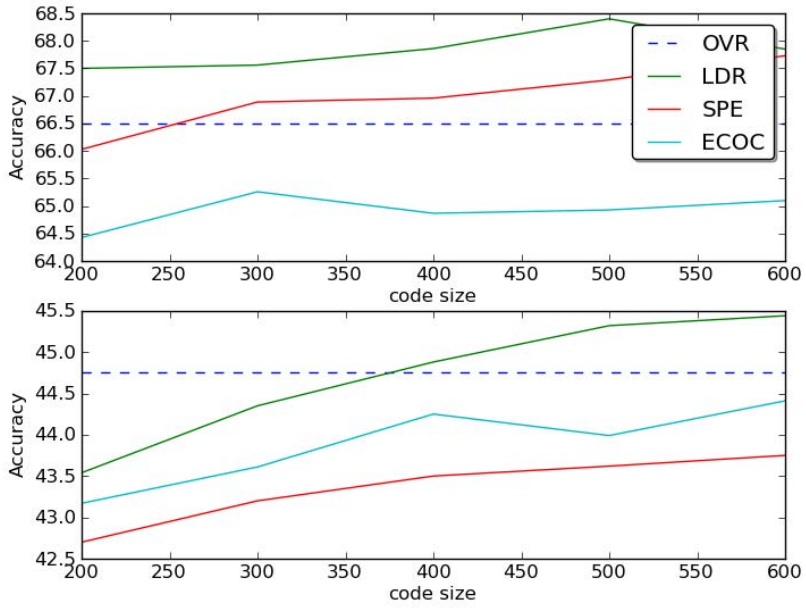


FIGURE 3.3: Accuracy of our method (LDR), random ECOC (ECOC), Spectral Embedding (SPE), and OVR as a function of code length on datasets with 1 000 classes (top) and with 5 000 classes.

3.3.4 Zero-shot learning

A few approaches have been proposed in the literature to answer the *zero-shot learning* problem (Larochelle et al., 2008, Palatucci et al., 2009), i.e. designing a classifier that is able to discriminate between classes for which we do not have instances in the training set. One particular approach proposes the use of a rich semantic encoding of the classes (Palatucci et al., 2009). Our approach is close to this idea since the codes of classes (computed by the autoencoder) are vectors that encode some semantic information on classes.

To explore empirically how our model is able to achieve zero-shot learning, we performed the following experiment on the 1000 classes dataset. We learned the class codes on the 1000 class representations (similarity vectors) computed from the hierarchy, s_i. Then we selected randomly a number of classes (10 to 50) and removed all training samples of these classes from the training set. The dichotomizers were then trained with this reduced training set. At test time,

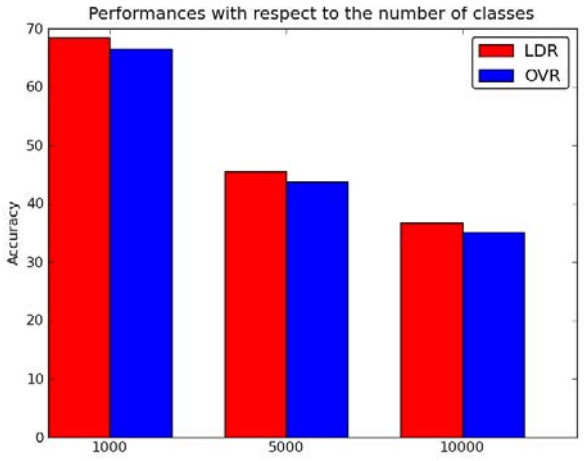


FIGURE 3.4: Accuracy of our method (LDR) and OVR on datasets with 1 000, 5 000 and 10 000 classes. Whatever the dataset LDR exploits class codes of length $l = 500$.

Classifiers	1K classes			5K classes		
	Accuracy	T.I.L	Speed	Accuracy	T.I.L	Speed
One-vs-rest	66.50%	2.63	×1	44.76%	3.98	×1
Random ECOC	65.10%	2.74	×2	44.41%	4.12	×12
SPE	67.73%	2.51	×2	43.75%	4.30	×12
LDR (first)	67.49%	2.54	×5	44.88%	3.98	×17
LDR(best)	**68.40%**	**2.46**	×2	**45.44%**	**3.93**	×12

TABLE 3.3: Comparative results of OVR, Random ECOC, Spectral Embedding, and LDR, on datasets with 1000 and 5000 classes with respect to accuracy, tree induced loss, and inference runtime. The runtimes are given as speed-up factors compared to OVR (×2 means twice as fast as OVR). Reported results are the best ones obtained on the datasets whatever the class code length. For LDR, we also provide the performance reached for a minimal B yielding performance at least equal to that of OVR, denoted as LDR (first), to stress the speed-up. LDR(best) is the best performance LDR based model regardless of the speed.

following the approach in (Larochelle et al., 2008), we use the learned classifier to discriminate between the classes whose training samples were not present in the training set. Results are given in Table 3.4 for a class code length equal to 200. One can see that the accuracy achieved by LDR on classes that have not been learned is significantly greater than a random guess although it is naturally

TABLE 3.4: Average accuracy (and standard deviation) of LDR ($B = 200$) for zero-shot learning tasks. Results are averaged over 10 runs with removal of different random sets of classes.

# classes removed	10	20	30	40	50
Accuracy (std)	25.64(12.20)	24.45(6.34)	16.76(4.24)	14.31(3.18)	12.76(2.48)

lower than the accuracy obtained on classes that were actually represented in the training set as reported in previous section.

Note also that one could go one step further than the zero-shot paradigm and try to recognize samples from a new class which was even not used for learning the class codes, provided one gets its similarity with all classes in the training stage. This would fit with many large multi-class problems where the set of classes is not closed (for instance new classes appear periodically in the DMOZ repository). Preliminary results show a similar performance as above provided the number of new classes remains small. This is a perspective of our work.

3.4 Conclusion

Learning compact distributed representation of classes combines the accuracy of flat methods and the fast inference of hierarchical methods. It relies on building distributed compact binary class codes that preserve class similarities. The main features of the method lies in its inference complexity that scales sub-linearly with the number of classes while outperforming the standard OVR and Error Correcting Output Codes techniques on problems up to 10 000 classes. Interestingly it also allows, to some extent, considering the addition of new classes in the hierarchy without providing training samples, an instance of the zero-shot learning problem. Zeroshot classification is a plausible scenario in extreme classification and deserves a closer look. Also, in order to improve the classification performances, binary class codes can be learned jointly with the binary dichotomizers. This would result in a more complex training procedure but can yield better representations than those learned separately.

Chapter 4

Extreme Multilabel Classification

4.1 Introduction

The largest part of the extreme classification literature is devoted to the single label case where each instance only belongs to one class (Bengio et al., 2010, Rifkin and Klautau, 2004b). However, the multilabel setting in which many labels can be associated to a given instance is receiving increasing attention in many domains such as text categorization[1], image classification[2] and genomics[3]. For example, a wikipedia document about the soccer world cup can be considered relevant for sports as well as for economics. Similarly, many objects of interest may appear in a single image. Also, a specific gene can potentially belong to many different functional groups. In MLC, sets of labels are generally represented as binary vectors $\mathbf{y}$ of length L (L is the number of labels) in which each position y_i is set to 1 if its corresponding label belongs to the set of relevant labels. The task of multilabel classification (MLC) is to a learn from a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ a classifier that predicts for a given instance the set relevant labels $\mathbf{h} : \mathcal{X} \to \{0,1\}^L$.

MLC is not an overlooked topic in machine learning (Tsoumakas and Katakis, 2007). However, it has received an increased interest because of the large number

[1] http://lshtc.iit.demokritos.gr/
[2] http://www.imageclef.org
[3] http://bioasq.org/

of labels in many real world applications[4]. For example, there are about $20K$ labels in the current version of imagenet and about $+300K$ labels in Wikipedia dataset used in the pascal LSTHC challenge. The number of labels can even grow up to millions in the context of bid phrase prediction for computational advertising (Agrawal et al., 2013). Predicting multiple labels in such large output spaces is called extreme MLC in contrast to traditional MLC problems where the size of the output space rarely exceeds 100 (e.g. there are only 14 labels in the widely used yeast dataset[5]).

Predicting multiple labels when the number of labels is large calls for scalable approaches taking into account the specific features inherent to problems of this size. For example in extreme MLC, the *label cardinality*, defined as the average number of labels per instance is typically very small compared to the number of labels available. Hence the magnitude of the *label density* (label cardinality divided by the number of labels) is also very small in extreme MLC compared to small scale problems: for the yeast dataset which has only 14 labels, the label density is 0.3 while it is about 10^{-5} for the large Wikipedia dataset ($+300K$ labels).

The large number of labels and the *small label density* have consequences on the evaluation measures used to assess the validity of the proposed methods and also algorithmic implications on the design of efficient extreme MLC methods. Indeed, multilabel classification algorithms are tailored for specific loss functions some which are less relevant when the number of labels is very large. We discuss in section 4.2 the main loss functions used in MLC and their relevance in the extreme setting.

As in extreme single label classification, many labels are very underrepresented. This emphasizes the need for multitask/transfer learning approaches (Hariharan et al., 2010) or the use of additional information in order to achieve good performances (Godbole and Sarawagi, 2004). In MLC, the main available additional information is the label dependence (Dembczynski et al., 2012). It has been exploited

[4] http://research.microsoft.com/en-us/um/people/manik/events/xc13/index.html
[5] http://mulan.sourceforge.net/datasets.html

in various ways to improve classification performances by early approaches (Dembczynski et al., 2010a, Godbole and Sarawagi, 2004, Read et al., 2009). After presenting the classical one versus all approach called Binary Relevance when used for MLC in section 4.3, we describe the main approaches designed to exploit label dependence in section 4.4. Most of these early attempts to exploit label dependence do not scale to the extreme classification setting. Therefore, new scalable methods have to be derived to solve extreme MLC. We present the most important contributions in this last line of work in section 4.5 before concluding the chapter.

4.2 In defense of Hamming Loss

Choosing the right loss function is an important yet overlooked step in solving MLC problems (Dembczynski et al., 2010a, 2012). Depending on the problem considered, one may be interested in learning a multilabel classifier that minimizes the number of disagreements between the predicted label set and the target one. The loss function corresponding to this setup is called the *Hamming Loss* L_H and is formally defined for a given classifier $\mathbf{h}$ and instance $(\mathbf{x}, \mathbf{y})$ as :

$$L_H(\mathbf{y}, \mathbf{h}(\mathbf{x})) = \frac{\mathbf{1}}{\mathbf{L}} \sum_{\mathbf{i=1}}^{\mathbf{L}} \mathbf{I}\{\mathbf{y_i} \neq \mathbf{h_i}(\mathbf{x})\} \tag{4.1}$$

where $I(\cdot)$ is the indicator function and h_i is the prediction corresponding to the i-th position in the vector $\mathbf{h}(\mathbf{x})$. The Hamming Loss is very intuitive since in practice, for a document whose exact set of relevant labels is $\{ball, beach, soccer, tournament\}$, it appears reasonable to penalize more severely a system predicting the set of labels $\{ball, karate, politics, soccer\}$ than another one that predicts $\{ball, beach, soccer\}$ since the latter's prediction is closer to the actual label set. Nonetheless, in some MLC applications people find interesting to penalize equally all the predicted label sets that do not exactly match the target label set. The corresponding loss function, called the subset 0/1 loss $L_{0/1}$, is different from the Hamming loss even though they are known to coincide in some situations according the following result:

Proposition 4.1. *(Dembczynski et al., 2012) The Hamming loss and the subset* $0/1$ *loss have the same risk minimizer, i.e* $\mathbf{h}^*_H(\mathbf{x}) = \mathbf{h}^*_{\mathbf{0/1}}(\mathbf{x})$*, for any* $\mathbf{x}$ *if one of the following conditions holds:*

1. *labels* $y_1, \ldots, y_n$ *are conditionally independent, i.e* $\mathbf{P}(\mathbf{y}|\mathbf{x}) = \mathbf{\Pi^L_{i=1}} \mathbf{P}(\mathbf{y_i}|\mathbf{x})$.
2. *the probability of the mode of the joint probability distribution is greater than or equal to* 0.5*, i.e* $\mathbf{P}(\mathbf{h}^*_{0/1}(\mathbf{x})|\mathbf{x}) \geq \mathbf{0.5}$.

In extreme classification, it is unlikely that any of the two conditions of the previous proposition hold. First, labels are necessarily dependent in real world applications (Hariharan et al., 2010). For instance, odds are high that the labels *sun* and *boat* are relevant for a given image if the label *beach* is relevant for the same image. Conversely, it is unlikely to find *human babies* and *lions* on the same real picture. Therefore, labels cannot reasonably be considered independent in real world applications. Also, because ground truth is often obtained from a collaborative labelling process, target label sets can suffer from the presence of noise. Hence, only rarely there will be a clear winner (with probability greater than 0.5) between them. Therefore, one has to choose the specific criteria of interest when designing an algorithm to solve MLC problems as suggested by (Dembczynski et al., 2012).

In the context of extreme MLC, we argue that the subset zero-one loss is not a good performance measure. Many arguments support this claim among which the size of the output space that can cause noisy labelling. Indeed there are potentially $O(2^L)$ label sets for a problem with L labels even though in practice, the number of label sets present in the dataset is generally much smaller but remain larger than the number of labels (Tsoumakas et al., 2010). For example the Wiki1K dataset (which will be later used in our experiments) has 1K labels and +40K sets of labels. Similarly, the RCV-industries dataset [6] has 303 labels and 4470 label sets. In these situations, missing labels can correspond to relevant labels that was not considered as such during the labelling procedure due to the large size of the output space.

[6] http://lshtc.iit.demokritos.gr/

Learning is also very difficult because most of the label sets are not enough represented. Moreover, in most real world MLC applications (mainly in document/image classification tasks), we are more interested into predicting a set of relevant labels in a reasonable time than predicting the exact set of labels. Therefore, algorithmic contributions to extreme MLC are more oriented towards optimizing Hamming loss (Dekel and Shamir, 2010, Hsu et al., 2009, Tai and Lin, 2012) rather than zero-one subset loss or other losses from information retrieval such as the F-measures. Nonetheless, these losses are used in addition to Hamming loss to have a better understanding of the methods' behaviour.

4.3 On Binary Relevance

A straightforward approach to MLC is to decompose the initial problem into many binary classification problems. One binary classifier is trained for each label and used to predict whether for a given instance this label is relevant or not. This one-versus-all method (Rifkin and Klautau, 2004b) called *Binary Relevance* (BR) in the context of MLC can be directly applied even when the number of labels is large thanks to several desirable properties. First it is simple and consistent with the hamming loss because the optimal classifier is obtained when the individual binary classifiers are optimal: $\mathbf{h}^*(\mathbf{x}) = (\mathbf{h}_\mathbf{1}^*(\mathbf{x}), \ldots, \mathbf{h}_\mathbf{L}^*(\mathbf{x}))$ (Dembczynski et al., 2012). Secondly, it is readily parallelizable since the binary problems can be trained independently. Finally it has achieved state of the art performances on several benchmarks (Dembczynski et al., 2012, Tsoumakas and Katakis, 2007).

However, it ignores the dependences that naturally exist between the labels in MLC problems and has linear training and inference complexity. These are strong limitations since label dependence can be leveraged to improve classification performances and linear time training and inference complexity can be prohibitive in extreme MLC problems (unless some effort is put into parallelizing training and inference) if the number of labels grows linearly with the number of examples (i.e $L \geq \Omega(n)$) as shown in (Dekel and Shamir, 2010). Most of the methods proposed to efficiently solve extreme MLC tackle these two limitations.

4.4 Early approaches to MLC

It is a common belief that multilabel classifiers' performances can be improved when the label dependence information is exploited. Most of the early works on MLC have focused on this challenge (Dembczynski et al., 2010a, Godbole and Sarawagi, 2004, Read et al., 2009). The main contributions in this line of work have demonstrated performance improvements over Binary Relevance on many classical small scale benchmark datasets (both on Hamming loss and subset 0/1 loss). However, in most cases the improvement is obtained at the expense of increased inference complexity leading to scalabitlity issues when applied to extreme multilabel classification. Nonetheless, these early approaches remain important contributions to MLC because they laid the foundations of label dependence exploitation and inspired several recently introduced approaches. We describe next the main methods that have been proposed in this line of work.

4.4.1 Stacking Binary Relevance

While the classical Binary Relevance approach completely ignores the potential correlations between the labels (Rifkin and Klautau, 2004b, Tsoumakas and Katakis, 2007), it is possible to exploit the binary classifiers predictions as additional information to learn improved classifiers in a second step. This is the main idea behind *Stacking* (Godbole and Sarawagi, 2004) which replaces the original features by the predictions, obtained after learning every label separately in a first step. There are various ways of using the initial predictions even though none of them has proven to be consistently superior to others. For example, binary classifiers used in the first step can provide probabilities. Therefore, it is possible to use either hard 0/1 predictions or probability estimations. One can also use the predictions as additional features combined with the original ones as new input to the second levels classifiers (Dembczynski et al., 2012). In the first case, stacking can be interpreted as a regularization procedure while in the latter it can be seen as feature expansion. Both methods have contributed to improve Binary Relevance in terms of 0/1 subset loss $L_{0/1}$ and *Hamming Loss* L_H (Dembczynski et al., 2012).

4.4.2 Classifier Chains (CC)

Classifier Chains were first introduced in (Read et al., 2009) as an improved version of Binary Relevance exploiting the correlation between the labels by learning as many classifiers as labels in augmented input spaces. However, conversely to Stacking (Godbole and Sarawagi, 2004) the classifiers are trained sequentially and each one $h_i(\cdot)$ only uses the predictions of the classifiers trained before it $(h_j)_{1:i-1}$ as additional features therefore having:

$$\begin{aligned} h_i : \mathcal{X} \times \{0,1\}^{i-1} &\rightarrow [0,1] \\ (x, y_1, \ldots, y_{i-1}) &\rightarrow [0,1] \end{aligned}$$

In case the base learners are probabilistic classifiers such as logistic regression, (Dembczynski et al., 2010a) have interpreted the approach described in (Read et al., 2009) as being a simplified inference procedure of a more general approach called *Probabilistic Classifier Chains* (PCC) whose inference procedure would produce a probability for each label combination using the product rule of probability, and hence cause an exponential complexity. PCC is a consistent method for optimizing the zero-one subset loss $L_{0/1}$. The inference procedure of PCC consist in finding the mode of the joint distribution of the labels which is intractable unless when the number of labels is very small. Different heuristics using *beam search* (Kumar et al., 2012) or *monte carlo sampling* (Read et al., 2014) have been proposed to reduce the complexity of PCC's inference process. However, these methods are still more costly than the original CC method even though they improve the performances. Moreover, the order in which the classifiers are trained can have an important impact on the final performances. Therefore, both (Dembczynski et al., 2010a, Read et al., 2009) recommend to use ensemble of classifier chains to achieve good performances and to be less sensitive to permutation considered when training the classifiers.

4.4.3 Label Powerset and friends

The Label Powerset (LP) appoach (Tsoumakas and Katakis, 2007) reduces the multilabel classification to a multiclass single label classification problem in which the meta-classes to predict are combination of labels. The number of combination of labels can be as large as $O(2^L)$ even though only the combinations of labels that are present in the dataset are usually considered. The remaining number of meta-classes is still larger than the number of labels in general. In (Dembczynski et al., 2012), it is shown that LP is tailored for the subset zero-one loss $L_{0/1}$ since predicting the optimal meta-class corresponds to predicting the mode of the joint label distribution. Despite its good performances (mainly on the $L_{0/1}$ loss), LP is not a feasible option in an extreme classification context because of its computational burden. Several attempts such as the *Rake*l approach (Tsoumakas et al., 2010) have been proposed to make LP applicable to large scale settings. This latter ensemble method trains an LP-like classifier on several meta-classes defined on random subsets of labels. The method is parameterized by the number label subsets to consider and the size of these subsets. Despite Rakel's good performances on several benchmark datasets (Tsoumakas et al., 2010) with a lower complexity compared to LP, its complexity is still high and does not allow its use in an extreme classification setting.

4.5 Scalable approaches to Extreme MLC

The increasingly large output spaces in MLC problems have shifted the focus of research in this topic from performance improvement via label dependence exploitation (Dembczynski et al., 2012, Read et al., 2009) to the challenge of improving the scalability of the proposed methods (Hsu et al., 2009, Tai and Lin, 2012). More precisely, most of the recent work in extreme MLC is about deriving new methods with reduced inference complexity while maintaining competitive performances compared to BR, which is the classical baseline. Indeed when the number of labels is very large, it is often reasonable to trade some accuracy for better scalability because speed is an important factor in many real world problems.

The underlying idea of the most important contributions in this line of work is to first embed the label sets into a low dimensional representation. Then, a function is learned to predict for every instance the latent representation corresponding to its relevant set of labels. Finally, a mapping designed according to the encoding function is used to recover the original sets of labels from the previously predicted low dimensional representation. This three steps procedure can be formally unified into the general framework of learning the following composition of functions:

$$\begin{aligned} H: &\quad D(\hat{\mathbf{e}}(\cdot)) \\ \mathbf{e}: &\quad \mathcal{Y} \to F^p \\ \hat{\mathbf{e}}: &\quad \mathcal{X}^d \to F^p \\ D: &\quad F^p \to \mathcal{Y} \end{aligned}$$

The embedding function $\mathbf{e}$ is used only at training time to encode sets of labels. At inference time, for each instance, the function $H(\cdot)$ is applied. That is, the latent representation is inferred using $\hat{\mathbf{e}}$ and the set of relevant labels is recovered with the decoding function D. Compared to linear time approaches such as Binary Relevance, the complexity gains at inference can be very important because the size of the embedding space p can be as small as $O(\log L)$. Combining the prediction of the latent representation from the data (whose complexity is $O(d \log L)$) with the decoding step ($O(L \log L)$) yields an overall inference complexity of $O((L + d) \log L)$.

The main approaches in this line of work can be divided into two subgroups. The methods of the first group, which we call label selection approaches, are based on selecting F^p to be a subset of $\mathcal{Y}$ such that $p << L$. The methods of the second group, the label transformation approaches, use non trivial transformations to project the label sets into a lower dimensional space. Conversely to the label space pruning approaches, the components of this embedding space are not necessarily interpretable. Next we describe in more details the most representative contributions in each of these groups.

4.5.1 Label Selection Methods

4.5.1.1 Label Space Pruning

In extreme classification, most classes are very rare because the label distribution usually follows power law as has been previously observed in several dataset such as Wikipedia[7]. This causes sample complexity problems since for some classes, there is not enough data to learn accurate classifiers. This observation inspires the pruning procedure proposed in (Dekel and Shamir, 2010). The authors of this study argue that when the number of classes grows linearly with the number of examples (i.e $L \geq \Omega(n)$), and when the final loss of interest is the Hamming Loss, then it is possible to improve the performances of the Binary Relevance classifier by identifying and removing a set of labels that act as distractors. These labels are identified offline using a very simple rule. In a first round, BR classifiers are learned on a training set and their false positive (FP) and true positive (TP) rates are evaluated on a separate validation set. Then, every classifier for which the ratio $FP/TP > (1-\gamma)/\gamma$ (where γ is a parameter controlling the importance of false positives versus false negatives and equals 1/2 for the Hamming loss) is removed and will not be evaluated at inference time. The authors give both theoretical and empirical evidence supporting the validity of this approach. To cast this strategy into the previously defined framework, it suffices to consider the pruning step as the encoding function and the decoding function to be the identity. It is also possible to control the number of labels to remove by ranking the classifiers according to their ratio FP/TP and considering the k smallest labels. This approach can be applied as a first step to reduce the number of classifiers to evaluate at inference time. If the number of labels removed in this first step does not result in important complexity gains, more sophisticated methods such as Principal Label Space Transformation (Tai and Lin, 2012) can be used to further reduce the complexity.

[7] http://lshtc.iit.demokritos.gr/

4.5.1.2 Column Subset Selection Method

While (Dekel and Shamir, 2010) propose to select a subset of labels that will be predicted at inference hence removing the other labels which are considered as distractors, the authors of (Bi and Kwok, 2013) propose to select a subset of labels from which the rest of labels can be reconstructed. To that end, they propose a theoretically grounded method based on randomized sampling and building on previous work on Column Subset Selection by (Balasubramanian and Lebanon, 2012). Given the label matrix $Y \in \{0,1\}^{n \times L}$ and a positive integer k, the Column Subset Selection Problem (CSSP) consist in finding exactly k columns of Y that span Y as much as possible. This problem is equivalent to finding an index C with cardinality k such that $||Y - Y_C Y_C^{\dagger} Y||_F$ is minimized (where $Y^{\dagger}$ is the Moore-Penrose pseudo-inverse of matrix Y). To solve this problem, (Bi and Kwok, 2013) propose an efficient approximate algorithm. Given a selected subset of labels, their corresponding binary classifiers are trained as in BR. At inference, the latent representation $\mathbf{h}$ of each example is inferred (that is, the predictions of the subset of labels previously selected) and the actual label set is recovered using the mapping $\mathbf{h}^T Y_C^{\dagger} Y$. Therefore, the whole process follows the general function composition framework we defined. The CSSP method has yielded competitive performances on several benchmark datasets (Bi and Kwok, 2013).

4.5.2 Label Transformation Methods

4.5.2.1 Compressed Sensing

The Compressed Sensing (CS) approach to extreme MLC (Hsu et al., 2009) is based on the assumption that one can learn to predict compressed label sets instead of the actual labels and recover the labels accurately when the output space is sparse. The compression is achieved by multiplying the original label matrix with a random projection matrix satisfying the restricted isometry property (RIP) such as gaussian or Hadamard matrices. The code size (dimensionality of the latent space) can be as small as $O(\log L)$ in theory even though in practice, larger sizes of the latent space are needed to achieve competitive performances. According to

the previously described framework, linear regressors $(\hat{\mathbf{e}})_i$ are learned to predict the low dimensional representation of the data. To recover the original labels, a non-trivial pre-image problem has to be solved using a sparse recovery algorithm such as Orthogonal Matching Pursuit (OMP) or Lasso (Pati et al., 1993). In the best case (when correlation decoding procedure (Hsu et al., 2009) is used), the reconstruction's complexity is $\Omega(LC \log L)$ where C is the label cardinality. CS is a precursor among the embedding approach and has been widely adopted as a classical baseline. However, almost all the label transformation methods that have been proposed recently such as principled label space transformation (Tai and Lin, 2012) achieve better performances while having a lower inference complexity for a fixed code size. A bayesian compressed sensing method (Kapoor et al., 2012) has been proposed recently to improve the classical compressed sensing and also to handle missing labels. However, this method also suffers from the computational burden of its variational inference procedure in an extreme classification setting.

4.5.2.2 Principle Label Space Transformation

Rather than using a random projection matrix for embedding the label sets into a lower dimensional space, (Tai and Lin, 2012) use a singular value decomposition on the label matrix ($Y = U\Sigma V^*$). This has two main benefits: first it allows to exploit the correlations between the labels and more importantly, it naturally yields simple reconstruction procedure conversely to compressed sensing. Indeed the reconstruction step of the principle label space transformation (PLST) is reduced to simple matrix multiplication by U^T projecting the embedded inputs back to the original label space. The encoding function used by PLST results in easier regression problems compared to random embedding of CS (Tai and Lin, 2012). It has also been proven useful to jointly learn the regressors and the encoding of the labels as in Canonical Correlation Analysis (CCA) (Zhang and Schneider, 2011) and the Conditional Principle Label Space Transformation (CPLST) (Chen and Lin, 2012). However, the improvements resulting from this latter approach are modest and its training complexity is higher.

4.6 Conclusion

From the initial goal of performance improvement regardless of the methods' inference complexity (Godbole and Sarawagi, 2004, Read et al., 2009, Tsoumakas and Katakis, 2007)(as shown in early approaches description), the focus of research in MLC is blossoming into inference complexity reduction through label dependence exploitation (Chen and Lin, 2012, Dekel and Shamir, 2010, Hsu et al., 2009, Tai and Lin, 2012) to name a few. Overall, there is no empirical evidence to decide a clear winner between the label transformation and label selection methods. For instance, the results presented in the recent study (Bi and Kwok, 2013) suggest a tie between CSSP and PLST. Additionally to the embedding based approaches previously presented, there are recent hierarchical attempts to extreme MLC. The most notable of method in this new trend is arguably the random forest approach presented in (Agrawal et al., 2013). In this paper the authors present a promising approach to extreme MLC with sublinear inference complexity. Their method achieve very good performance on a bid recommendation problem and opens a new directions in hierarchical extreme MLC.

Chapter 5

Extreme Multilabel Classification with Bloom Filters

5.1 Introduction

In multi label classification most of the available approaches were not designed to scale to the extreme setting. To bridge the gap between the lack of scalable approaches and the ubiquity of extreme MLC problems some authors have proposed to learn an intermediate low dimensional representation of the labels (Chen and Lin, 2012, Hsu et al., 2009) as described in the previous chapter. Following that line of work, we propose to encode individual labels on K-sparse bit vectors of dimension B, where $B \ll L$, and use a disjunctive encoding of label sets (i.e. bitwise-OR of the codes of the labels that appear in the label set). Then, we learn one binary classifier for each of the B bits of the coding vector, similarly to BR (which corresponds to the special case $K = 1$ and $B = L$). By setting $K > 1$, one can encode individual labels on far less than L bits while keeping the disjunctive encoding unambiguous for a large number of label sets of small cardinality. Compared to BR, our scheme learns only B binary classifiers instead of L, while conserving the desirable property that the classifiers can be trained independently and thus in parallel, making our approach suitable for large-scale problems.

We propose two approaches that were inspired and motivated by Bloom filters (Bloom, 1970), a well-known space-efficient randomized data structure designed for approximate membership testing. Bloom filters use exactly the principle of encoding objects (in our case, labels) with K-sparse vectors and encode a set with the disjunctive encoding of its members. The filter can be queried in order to know whether or not a given item is present and the answer is correct up to a small error probability. Bloom Filters have been extensively used in database management systems and networking applications [1]. The data structure is randomized because the representative bits of each object are obtained by random hash functions; under uniform probability assumptions for the encoded set and the queries, the encoding size B of the Bloom filter is close to the information theoretic limit for the predefined error rate.

The first method we propose (Standard Bloom Filters) is a direct application of Bloom Filters to extreme MLC by randomly encoding the labels using a uniform sampling procedure to chose the representative bits (non-zero bit positions) of each label. It builds on the sparsity of the output space when the number of labels is very large similarly to (Hsu et al., 2009). We analyze the power of the proposed approach to reduce the complexity compared to Binary Relevance and the potential weaknesses inherent to the decoding process of Bloom Filters. Then we propose two decoding algorithms for this first approach. While the first one is simple because it is the standard application of the querying procedure of Bloom Filters, it is not robust to the potential mistakes of binary classifiers. The second decoding procedure aims at fixing this problem by exploiting predicted probabilities given by probabilistic binary classifiers (e.g logistic regression) in a softer way.

The second approach, which we call Robust Bloom Filters, also aims at building binary codes that are robust to mistakes of binary classifiers. It exploits a key structural property of extreme MLC problems: many labels never appear together. This property is called label clustering and is responsible for the improvements compared to the standard approach. These improvements are twofold. First, the encoding of "relevant" label sets are unambiguous with the disjunctive encoding.

[1] http://en.wikipedia.org/wiki/Bloomfilter

label	h_1	h_2	h_3	label	h_1	h_2	h_3
ℓ_1	2	3	5	ℓ_5	1	4	6
ℓ_2	2	4	5	ℓ_6	3	5	6
ℓ_3	1	2	5	ℓ_7	3	4	5
ℓ_4	1	5	6	ℓ_8	2	5	6

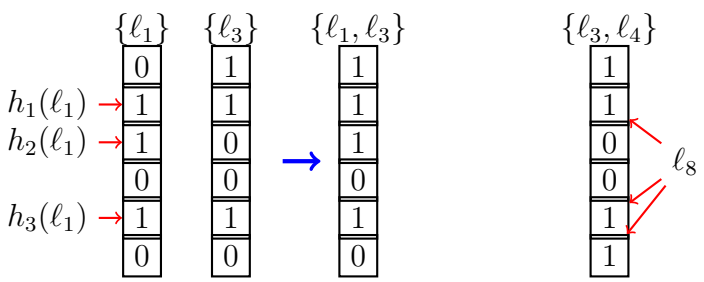


FIGURE 5.1: Examples of a Bloom filter for a set $\mathcal{L} = \{\ell_1, ..., \ell_8\}$ with 8 elements, using three hash functions (h_1, h_2, h_3 and 6 bits). *(left)* The table gives the hash values for each class. *(middle)* For each class, the hash functions give the index of the bits that are set to 1 in the 6-bit boolean vector. The examples of the representative vectors for ℓ_1 and ℓ_3 are given. Then, the subset $\{\ell_1, \ell_3\}$ is built by taking the bitwise OR of the vectors of ℓ_1 and ℓ_3. *(right)* Example of false positive: the representation of the subset $\{\ell_3, \ell_4\}$ is given ; all the representative bits the class ℓ_8 are set to 1, so the standard decoding algorithm considers that the vector encodes the set $\{\ell_3, \ell_4, \ell_8\}$ rather than the intended $\{\ell_3, \ell_4\}$.

Secondly, the decoding step, which recovers a label set from an encoding vector, is robust to prediction errors in the encoding vector; for instance, for $K = 2$, we prove that the number of incorrectly predicted labels is no more than twice the number of incorrectly predicted bits.

After presenting the main features of Bloom Filters, we present the proposed approaches to extreme MLC using Bloom Filters. Theoretical and empirical arguments are provided supporting the assumptions underlying these approaches. The chapter is concluded with experiments comparing the Bloom Filter based methods with state of the art methods.

5.2 Background on Bloom Filters

Bloom filters (Bloom, 1970) are compact data structures for probabilistic representation of a set in order to support membership queries (i.e. queries that ask:

"Is element X in the set"). They have been extensively used in many domain such as networks monitoring and database management systems. Since their inception, various types of Bloom Filters such as counting Bloom Filters and Attenuated Bloom Filters have been introduced for specific applications. Here we restrict our study to classical Bloom Filters and refer the interested reader to the comprehensive survey (Broder et al., 2002) and the references therein.

Let $\mathcal{L}$ be a set of L elements (in our case, $\mathcal{L}$ is the set of L possible labels). A Bloom filter of size B uses K *hash functions* from $\mathcal{L}$ to $\{1, ..., B\}$, which we denote $h_k : \mathcal{L} \to \{1, ..., B\}$ for $k \in \{1, ..., K\}$. These hash functions are used to represent each $\ell \in \mathcal{L}$ by a bit vector of size B with at most K non-zero bits, where each hash function gives the index of a nonzero bit in the bit vector (all other bits are set to zero). Then, the Bloom filter encodes a subset $y \subseteq \mathcal{L}$ by a bit vector of size B, defined by the bitwise OR of the bit vectors of the elements of y. Figure 5.2 (left and middle) gives an example of a Bloom filter and of the encoding step, where subsets of a set of 8 elements are represented on 6 bits, using 3 hash functions.

Given a bit vector of size B that encodes an (unknown) subset $y' \subseteq \mathcal{L}$, the Bloom filter can be queried to know if a specific element $\ell \in \mathcal{L}$ belongs to y': the answer is positive if all the indexes $h_k(\ell)$ are set to 1 in the bit vector of y', and negative otherwise. When queried, a Bloom filter always answers positively if $\ell \in y'$. However, the encoding of the subsets of $\mathcal{L}$ by Bloom filters is not injective, since it encodes 2^L elements on $B < L$ bits; it is thus a lossy compression scheme. The loss of information translates into a possibility of *false positive* answers: when queried with an element ℓ, the Bloom filter may answer positively even though $\ell \notin y'$. Figure 5.2 (right) gives an example of this situation.

Assuming random subsets of fixed size C, random query elements and perfect hash functions, the false positive rate of a Bloom filter behaves asymmptotically like $(1/2)^K$ when the number of hash functions K is equal to $\frac{B}{C} \ln 2$ (see e.g. (Christensen et al., 2010)). This error rate is, up to a multiplicative factor of $1/\ln(2)$, the information theoretic limit for a lossy compression scheme using B bits for encoding subsets of size C of a set of size $L \gg C$ (Carter et al., 1978). Since the error rate is independent of L and increases exponentially fast with C for fixed B, the asymptotic rate suggests that Bloom filters are most efficient for

encoding small subsets of a large set. For MLC, they should then be a method of choice for datasets when L is large and the label cardinality is relatively small, which is the usual situation in MLC datasets with a large number of labels. A detailed discussion on the achievable compression ratio in the context of MLC is presented in subsection 5.3.2.

5.3 Standard Bloom Filters for Multilabel Classification

As a reminder, the problem of multilabel classification can be described as follows: given a set of labels $\mathcal{L}$, one has to learn a prediction function g which, to each possible input x, predicts a subset $g(x) = \hat{y}$ of $\mathcal{L}$. Learning is carried out on a training set $((x_1, y_1), ..., (x_n, y_n))$ of inputs for which the desired label sets are known. The basic principle of our approach is to encode each label set y on a bit vector of size B, which we denote by $\mathbf{e}(y) = (e_1(y), ..., e_B(y)) \in \{0, 1\}^B$. Learning is carried out by (independently) training B binary classifiers $\hat{e}_1, ..., \hat{e}_B$, where each $\hat{e}_j$ is trained on $((x_1, e_j(y_1)), ..., (x_n, e_j(y_n)))$, so that given a test input x, we can predict the encoding of its label set by $\hat{\mathbf{e}}(x) = (\hat{e}_1(x), ..., \hat{e}_B(x))$. The final prediction $g(x)$ is obtained by decoding $\hat{\mathbf{e}}(x)$.

The basic principle described above is fairly common in MLC, since it takes Binary Relevance as a special case: BR consists in a disjunctive encoding of label sets (thus using $B = L$ bits), and the decoder is the inverse of the encoder since the latter is one-to-one. Our approach is intended to encode the label sets on $B \ll L$ bits, so that the computational cost of training and testing are L/B times lower for our approach than for BR. To that end, we use the encoding/decoding procedures of Bloom filters for label sets, which are at the same time simple and computationally efficient. In the next subsection 5.3.1, we describe the encoding and decoding steps of the standard Bloom Filter approach to MLC, before discussing the complexity properties (relatively to the features of the problem at hand such as number of labels, label cardinality, etc) of the proposed method in the subsection 5.3.2

label	h_1	h_2	h_3	label	h_1	h_2	h_3
ℓ_1	2	3	5	ℓ_5	1	4	6
ℓ_2	2	4	5	ℓ_6	3	5	6
ℓ_3	1	2	5	ℓ_7	3	4	5
ℓ_4	1	5	6	ℓ_8	2	5	6

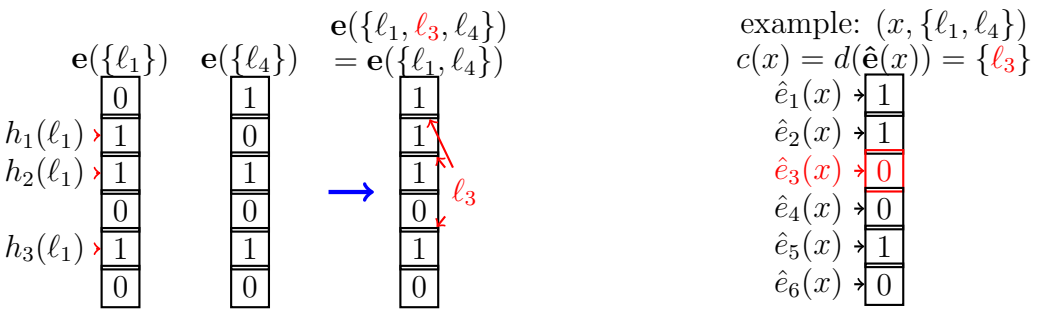


FIGURE 5.2: Examples of a Bloom filter for a set $\mathcal{L} = \{\ell_1, ..., \ell_8\}$ with 8 elements, using 3 hash functions and 6 bits). *(left)* The table gives the hash values for each label. *(middle-left)* For each label, the hash functions give the index of the bits that are set to 1 in the 6-bit boolean vector. The examples of the encodings for $\{\ell_1\}$ and $\{\ell_4\}$ are given. *(middle-right)* Example of a false positive: the representation of the subset $\{\ell_1, \ell_4\}$ includes all the representative bits of label ℓ_3 so that is ℓ_3 would be decoded erroneously. *(right)* Example of propagation of errors: a single erroneous bit in the label set encoding, together with a false positive, leads to three label errors in the final prediction.

Algorithm 3: Standard decoding algorithm.

Input : Test sample x;
Hash functions $h_1, ..., h_K$;
Binary classifiers $(\hat{e}_j)_{j=1}^B$;
Output: Label set $\hat{y} \subset \mathcal{L}$;
begin
  $\hat{y} \leftarrow \emptyset$;
  for $\ell \in \mathcal{L}$ **do**
    $s \leftarrow \sum_{k=1}^K \hat{e}_{h_k(\ell)}(x)$;
    if $s = K$ **then**
      $\hat{y} \leftarrow \hat{y} \cup \{\ell\}$

5.3.1 Encoding and Decoding

The basic principle of Bloom filters leads to an immediate training and inference algorithm for MLC. Indeed, the querying principle of the Bloom filter makes it possible to query any bit vector of size B, even if it is not a valid encoding of a

Algorithm 4: Correlation decoding algorithm.

Input : Test sample x;
Hash functions $h_1, ..., h_K$;
Probability estimators $(\hat{e}_j)_{j=1}^{B}$;
Thresholds $(t_\ell)_{\ell \in \mathcal{L}}$;
Output: Label set $\hat{y} \subset \mathcal{L}$;
begin
  $\hat{y} \leftarrow \emptyset$;
  for $\ell \in \mathcal{L}$ **do**
    $s \leftarrow \sum_{k=1}^{K} \hat{e}_{h_k(\ell)}(x)$;
    if $s > t_\ell$ **then**
      $\hat{y} \leftarrow \hat{y} \cup \{\ell\}$

subset of $\mathcal{L}$. Thus, training can be performed by learning a binary classifier for each bit of the filter, and decoding can be performed by querying the predicted Bloom filter for each possible label. The standard decoding is described in Algorithm 3.

This way of decoding may have the undesirable behavior that among the (at most) K bits that should be set to 1 for one class, a single error in $\hat{\mathbf{e}}(x)$ may result in the class not being predicted. Of course, this behavior allows to control the false positive rate of the Bloom filter, and is thus mostly intended. However, to get a better control on the final precision/recall of our approach, we propose a "soft" decoding method inspired from the loss based decoding proposed in (Allwein et al., 2001), which we call *correlation decoding*: instead of learning binary classifiers, we learn probability estimators for each bit of the Bloom filter (e.g. using logistic regression) and assign to each label the sum of the predicted posterior probabilities of its bit vector, instead of the sum of the binary decisions. Then, a label is added to the label set if this sum is greater than a threshold that is tuned (on the validation set, with the other hyperparameters) for each label. Note that the additional computational complexity of tuning a per-label decision threshold is negligible compared to learning the classifiers/probability estimators. This procedure is described in Algorithm4 Learning probability estimators also has the advantage of providing a ranking of the labels; even though our approach is not designed for ranking, we use this property in the experimental section to provide comparisons to the compressed sensing approach.

5.3.2 Computational Complexity

The overall computational cost of training is the sum of the costs of (1: preprocessing) encoding the label sets and (2: training) learning B binary classifiers. The preprocessing cost for n examples is in $O(nLK)$ with a small hidden constant. This is usually negligible compared to the training cost, which is at least $O(ndB)$ (with large hidden constants) for learning linear classifiers in dimension d, even if $B \ll LK$. For testing on a single input, the computational cost is the sum of (1) computing the predictions ($O(dB)$ for linear classifiers) and (2) decoding ($O(LK)$), which once again is usually negligible compared to the cost of the first step.

Thus, compared to Binary Relevance whose training and inference complexity are in $O(ndL)$ and $O(dL)$, the computational gain of our approach both for training and for testing is roughly a multiplicative factor of B/L. Compared to the compressed sensing approach of (Hsu et al., 2009), for which the training cost for linear functions is about $O(ndC \ln L)$ (with unknown hidden constant) and testing is at least in $O(dC \ln L)$ for predicting the encoding vector and at least $\Omega(LC \ln L)$ for decoding [2]. The relative complexity of training and testing between their approach and ours depends on how B compares to $C \ln L$ (up to a multiplicative factor), leaving aside that the decoding step is dramatically faster in our approach. We now give some theoretical and practical arguments that B indeed grows as $O(C \ln L)$.

Criterion under study As any lossy compression scheme, the parameters (B and K) should be chosen such that the loss of information, which we call *unrecoverable error* (the Hamming loss incurred by the false positive rate of the Bloom filter), is negligible compared to other sources of errors. Thus, given a probability measure $\mathbb{P}$ on the possible label sets (e.g. the marginal distribution of the label sets), the admissible region for B and K is such that the false positive

[2] This is the complexity of their simplest "correlation decoding", which consists in a multiplication of an $L \times O(C \ln L)$ matrix (the transpose of the coding matrix) and a vector of dimension $O(C \ln L)$ (the predicted encoding). The complexity is much higher with more sophisticated decoding algorithms such as orthogonal matching pursuit.

(FP) rate $fp(B, \mathbb{P}, K)$ of the Bloom filter is negligible compared to the overall prediction error. In practice, the optimal Hamming loss is usually roughly equal to some fraction of the label density C/L. This observation can easily be explained, since (1) the label density is the "baseline" Hamming loss, i.e. the Hamming loss obtained for the trivial classifier that never predicts any label, and (2) a Hamming loss of $\approx 10\%$ of the label density already corresponds to very high precision/recall values because the vast majority of labels appear very rarely. The admissible values for B and K of our approach should then follow

$$fp(B, \mathbb{P}, K) < 2^r \frac{C}{L} \tag{5.1}$$

for some $r < 0$ which defines how much the FP rate should be small compared to the label density.

Asymptotic Behavior If we assume that all label sets have the same size (equal to the label cardinality C) and denoting $\mathbb{U}_C$ the uniform probability over label sets of size C, the asymptotic FP rate (i.e. when $B \gg 1$ and $B \gg C$) of Bloom filters is achieved for $K = \frac{B}{N} \ln 2$ and is given by (see e.g. (Carter et al., 1978, Christensen et al., 2010)):

$$fp_0(B, \mathbb{U}_C, K = \frac{B}{N} \ln 2) = (1/2)^K \,. \tag{5.2}$$

According to this equation, criterion (5.1) is satisfied when $B > C * \left(\frac{r}{\ln(2)} + \frac{\ln \frac{L}{C}}{\ln(2)^2} \right)$. We can thus expect the asymptotic behavior of our method to accept values of B in $O(C \ln \frac{L}{C})$ in favorable cases, and to lead to exponential gains in terms of computational resources compared to binary relevance, matching the encoding size of the compressed sensing approach (Hsu et al., 2009).

Non-Asymptotic Considerations On real datasets, such as the RCV1-industry dataset on which we perform experiments in the next section ($C = 1.3$ and $C = 303$, see Table 5.1 for more details), Equation (5.2) gives $B \approx 24$ for $r = -5$ (meaning that we want the FP rate to be about $2^{-5} \approx 3\%$ of the label density). This is a wide underestimation of the reasonable values of B on that dataset,

mainly for two reasons: First, the formula for the FP rates for such a small B is overoptimistic. Second (and more importantly) the size of the label sets are somewhat "heavy tailed": a non-negligible proportion of the examples have label sets of size > 10, so the asymptotic assumption $B \gg C$ is far from true on a large portion of the dataset.

In order to give a more realistic theoretical approximation and avoid the two drawbacks above, we consider another formula for the probability of the FP rate, using the exact formula (for fixed B, C and K) given in (Christensen et al., 2010, Equation 17):

$$fp_1(B, C, K) = \sum_{p=1}^{L} Q(KC, B, p) \left(\frac{p}{B}\right)^K \tag{5.3}$$

where $Q(a, b, c) = \sum_{q=1}^{c} (-1)^{c-q} (\frac{q}{b})^a \binom{b}{c} \binom{c}{q}$. To account for the full distribution in the label set sizes, we can then use probabilities p_c (estimated on the training set) that the label set of a random example has cardinality c on a particular dataset:

$$fp(B, \mathbb{P}, K) = \sum_{c=1}^{L} p_c fp_1(B, c, K)\,, \quad \text{where } \mathbb{P} = \sum_{c=1}^{L} p_c \mathbb{U}_c\,. \tag{5.4}$$

Equation (5.4) gives a distribution-dependent formula for the theoretical FP rate, assuming that label sets of a given size are drawn uniformly at random. When $\mathbb{P}$ is concentrated around its average and B is large enough, (5.4) is well approximated by (5.2) because the asymptotic formula is itself a good approximation of (5.3) (Christensen et al., 2010); more generally, numerical calculations easily show that for fixed $\mathbb{P}$ and optimal K, the FP rate given by (5.3) decreases exponentially fast with B for natural distributions $\mathbb{P}$ (e.g. power laws). This refined version still allows us to argue that $B \in O(C \ln L)$ is the correct order of magnitude for large L and small C.

Simulations on Real Datasets In order to assess that the theoretical calculations are meaningful in practice, we performed simulations on the two datasets used in our experiments: RCV1-Industries ($C = 1.3$, $L = 303$) and Wikipedia1k ($C = 1.11$, $L = 1000$). These datasets are detailed in the next section. Figure 5.3 shows the distributions on these two datasets of the label set size. One can

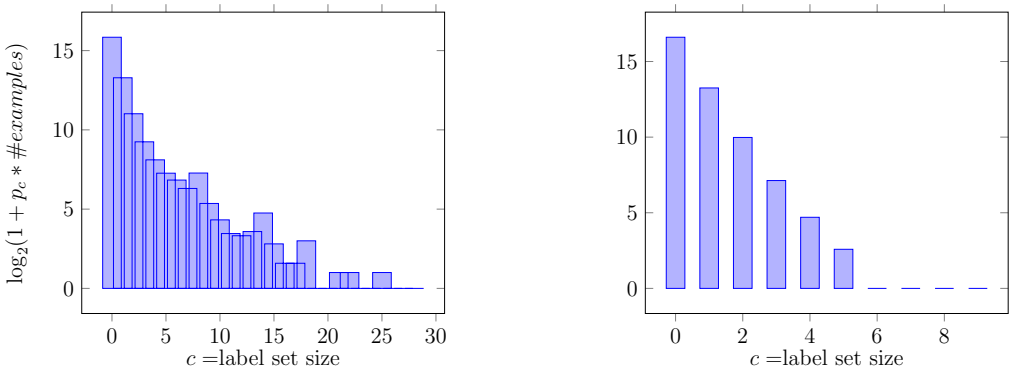


FIGURE 5.3: Distribution (in $\log_2$ scale) of the label set sizes on RCV1-Industries (left) and on Wikipedia1k (right). p_c is the probability of having an instance whose label set size is equal to c.

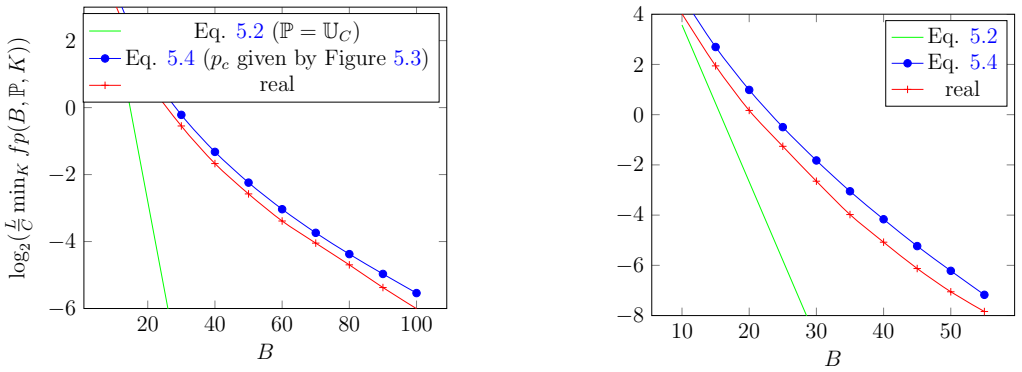


FIGURE 5.4: Theoretical and real unrecoverable Hamming loss (i.e. false positive rate) of the Bloom filter as a function of the size of the filter B (optimal K) on RCV1-Industries (left) and Wikipedia1k (right). Errors are printed in $\log_2$ scale relatively to the label density (the y-axis corresponds to the parameter r of Eq. 5.1).

see that while the label set size decreases exponentially fast on Wikipedia1k, the distribution on RCV1-Industries is much more heavy tailed.

Figure 5.4 shows the unrecoverable Hamming loss (i.e. the FP rate) of the Bloom filters on these datasets, as a function of B (the optimal value of K has been chosen for each B), both according to our theoretical approximations, and through direct simulation. We can first observe that while the asymptotic formula (5.2) dramatically underestimates the real FP rate, the refined formula (5.4) is very close to the real value (the difference between the two is because label sets of a given size are far from sampled uniformly in reality, which actually seems beneficial

on average in terms of FP rate). In the end, we can clearly observe that both the real and theoretical FP rates decrease exponentially fast with B, but also see that the RCV1-Industries dataset may actually need a Bloom filter as large (or maybe even larger) than Wikipedia1k. The reason is that the value of B in our approach has a logarithmic dependency in the number of labels , but a linear dependency with respect to the label set sizes, which tend to be larger on RCV1-Industries than on Wikipedia1k.

5.4 Extreme MLC with Robust Bloom Filters

The encoding and decoding schemes of BFs are appealing to define the encoder $\mathbf{e}$ and the decoder D in a reduction of MLC to binary classification, because as shown in section 5.5 the use of Bloom filters with random hash functions for MLC (denoted S-BF for Standard BF hereafter) leads to rather good results in practice.

Nonetheless, there is much room for improvement with respect to the standard approach above. The distribution of label sets in usual MLC datasets is far from uniform. This is an opportunity to make sure that false positive answers only occur in cases that are detectable from the observed distribution of label sets: if y is a label set and $\ell \notin y$ is a false positive given $\mathbf{e}(y)$, ℓ can be detected as a false positive if we know that ℓ never (or rarely) appears together with the labels in y. Second and more importantly, the decoding approach of BFs is far from robust to errors in the predicted representation. Indeed, BFs are able to encode subsets on $B \ll L$ bits because each bit is representative for several labels. In the context of MLC, the consequence is that any single bit incorrectly predicted may include in (or exclude from) the predicted label set all the labels for which it is representative. Figure 5.2 (right) gives an example of the situation, where a single error in the predicted encoding, added with a false positive, results in 3 errors in the final prediction. Our main contribution, which we detail in the next section, is to use the non-uniform distribution of label sets to design the hash functions and a decoding algorithm to make sure that any incorrectly predicted bit has a limited impact on the predicted label set.

We present a new method that we call Robust Bloom Filters (R-BF). It improves over random hash functions by relying on a structural feature of the label sets in MLC datasets: many labels are never observed in the same target set, or co-occur with a probability that is small enough to be neglected. We first formalize the structural feature we use, which is a notion of mutually exclusive clusters of labels, then we describe the hash functions and the robust decoding algorithm that we propose.

5.4.1 Label Clustering

The strict formal property on which our approach is based is the following: given a partition of $\mathcal{L}$ composed of P subsets $\mathcal{L}_1, ..., \mathcal{L}_P$ of $\mathcal{L}$, we say that $(\mathcal{L}_1, ..., \mathcal{L}_P)$ are *mutually exclusive clusters* if no target label set contains labels from more than one of each $\mathcal{L}_p, p = 1..P$, or, equivalently, if the following condition holds:

$$\forall p \in \{1, ..., P\}, \mathbb{P}_{y \sim \mathcal{D}_\mathcal{Y}}\Big(\quad \big(y \cap \mathcal{L}_p \neq \emptyset\big) \quad \text{and} \quad \big(y \cap \bigcup_{p' \neq p} \mathcal{L}_{p'} \neq \emptyset\big) \quad \Big) = 0\,. \tag{5.5}$$

where $\mathcal{D}_\mathcal{Y}$ is the marginal distribution over label sets. For the disjunctive encoding of Bloom filters, this assumption implies that if we design the hash functions such that the false positives for a label set y belong to a cluster that is mutually exclusive with (at least one) label in y, then the decoding step can detect and correct it. To that end, it is sufficient to ensure that for each bit of the Bloom filter, all the labels for which this bit is representative belong to mutually exclusive clusters. This will lead us to a simple two-step decoding algorithm (1) cluster identification, (2) label set prediction in the cluster. In terms of compression ratio $\frac{B}{L}$, we can directly see that the more mutually exclusive clusters, the more labels can share a single bit of the Bloom filter. Thus, more (balanced) mutually exclusive clusters will result in smaller encoding vectors B, making our method more efficient overall.

This notion of mutually exclusive clusters is much stronger than our basic observation that some pair of labels rarely or never co-occur with each other, and in practice it may be difficult to find a partition of $\mathcal{L}$ into mutually exclusive clusters because the co-occurrence graph of labels is connected. However, after removing

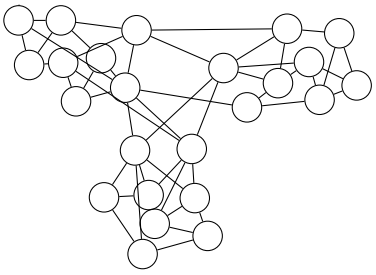


FIGURE 5.5: (a)

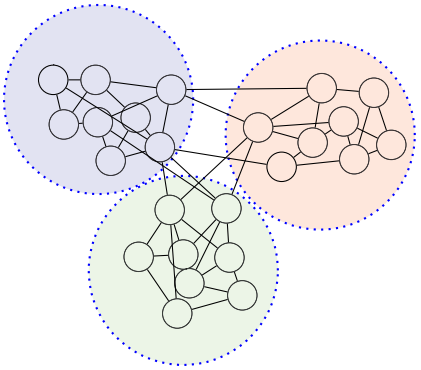


FIGURE 5.6: (b)

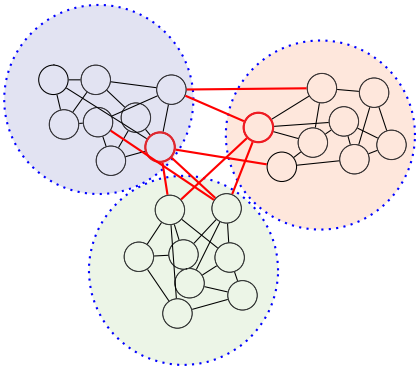


FIGURE 5.7: (c)

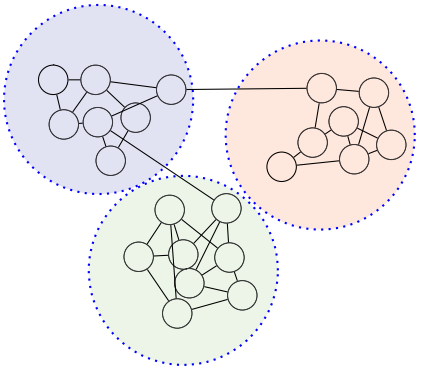


FIGURE 5.8: (d)

FIGURE 5.9: Illustration of the label clustering assumption in a practical situation: (a) The co-occurrence graph in which the labels are the nodes and the edges represent co-occurrence relations between the labels. In real world problems, the co-occurrence graph is a single connected component. (b) Even though the graph is connected, clusters of labels can be identified using a graph clustering algorithm. (c) Using the identified clusters as a partition of the set of labels results in unrecoverable loss represented by the edges in red. labels represented by nodes linked with red edges will never be predicted together. (d) The labels that are responsible for most of the unrecoverable loss correspond to the nodes which have the highest degree in the co-occurrence graph (nodes in red). We call them hubs. Removing these labels and treating them separately leaves the rest of labels approximately partitioned.

the few most central labels (which we call *hubs*, and in practice roughly correspond to the most frequent labels), the labels can be clustered into (almost) mutually exclusive labels using a standard clustering algorithm for weighted graphs. This process is illustrated in Figure 5.9.

In our approach, the hubs are dealt with outside the Bloom filter, with a standard binary relevance scheme. The prediction for the remaining labels is then constrained to predict labels from at most one of the clusters. From the point of view of prediction performance, we loose the possibility of predicting arbitrary label sets, but gain the possibility of correcting a non-negligible part of the incorrectly predicted bits. As we shall see in the experiments, the trade-off is very favorable. We would like to note at this point that dealing with the *hubs* or the most frequent labels with binary relevance may not particularly be a drawback of our approach: the occurrence probabilities of the labels is long-tailed, and the first few labels may be sufficiently important to deserve a special treatment. What really needs to be compressed is the large set of labels that occur rarely.

To find the label clustering, we first build the co-occurrence graph and remove the hubs using the degree centrality measure. The remaining labels are then clustered using Louvain algorithm (Blondel et al., 2008) which is a popular community detection algorithm widely used for social network analysis because of its scalability. It allows to find communities in graphs with thousands of nodes in only few seconds. To control the number of clusters, a maximum size is fixed and larger clusters are recursively clustered until they reach the desired size. Finally, to obtain (almost) balanced clusters, the smallest clusters are merged. Both the number of hubs and the cluster size are parameters of the algorithm, and, in the experiments, we show it is possible to choose them before training at negligible computational cost.

5.4.2 Encoding and decoding

From now on, we assume that we have access to a partition of $\mathcal{L}$ into mutually exclusive clusters (in practice, this corresponds to the labels that remain after removal of the hubs).

5.4.2.1 Encoding and Hash functions

Given the parameter K, constructing K-sparse encodings follows two conditions that allow cluster-wise unambiguity encoding and uniqueness assignment of binary codes:

1. two labels from the same cluster cannot share any representative bit;
2. two labels from different clusters can share at most $K-1$ representative bits.

Finding an encoding that satisfies the conditions above is not difficult if we consider, for each label, the set of its representative bits. In the rest of the paragraph, we say that a bit of the Bloom filter "is used for the encoding of a label" when this bit is a representative bit of the label. If the bit "is not used for the encoding of a label", then it cannot be a representative bit of the label.

Let us consider the P mutually exclusive label clusters, and denote by R the size of the largest cluster. To satisfy Condition 1., we find an encoding on $B = R.Q$ bits for $Q \geq K$ and $P \leq \binom{Q}{K}$ as follows. For a given $r \in \{1, ..., R\}$, the r-th batch of Q successive bits (i.e. the bits of index $(r-1)Q+1, (r-1)Q+2, ..., rQ$) is used only for the encoding of the r-th label of each cluster. That way, each batch of Q bits is used for the encoding of a single label per cluster (enforcing the first condition) but can be used for the encoding of P labels overall. For the Condition 2., we notice that given a batch of Q bits, there are $\binom{Q}{K}$ different subsets of K bits. We then injectively map the (at most) P labels to the subsets of size K to define the K representative bits of these labels. In the end, with a Bloom filter of size $B = R.Q$, we have K-sparse encodings that satisfy the two conditions above for $L \leq R.\binom{Q}{K}$ labels partitioned into $P \leq \binom{Q}{K}$ mutually exclusive clusters of size at most R.

Figure 5.10 gives an example of such an encoding. In the end, the scheme is most efficient (in terms of the compression ratio B/L) when the clusters are perfectly balanced and when P is exactly equal to $\binom{Q}{K}$ for some Q. For instance, for $K = 2$ that we use in our experiments, if $P = \frac{Q(Q+1)}{2}$ for some integer Q, and if the

bit index	representative for labels	bit index	representative for labels
1	$\{1, 2, 3, 4, 5\}$	7	$\{16, 17, 18, 19, 20\}$
2	$\{1, 6, 7, 8, 9\}$	8	$\{16, 21, 22, 23, 24\}$
3	$\{2, 6, 10, 11, 12\}$	9	$\{17, 21, 25, 26, 27\}$
4	$\{3, 7, 10, 13, 14\}$	10	$\{18, 22, 25, 28, 29\}$
5	$\{4, 8, 11, 13, 15\}$	11	$\{19, 23, 26, 28, 30\}$
6	$\{5, 9, 12, 14, 15\}$	12	$\{20, 24, 27, 29, 30\}$

cluster index	labels in cluster	cluster index	labels in cluster
1	$\{1, 15\}$	9	$\{9, 23\}$
2	$\{2, 16\}$	10	$\{10, 24\}$
3	$\{3, 17\}$	11	$\{11, 25\}$
4	$\{4, 18\}$	12	$\{12, 26\}$
5	$\{5, 19\}$	13	$\{13, 27\}$
6	$\{6, 20\}$	14	$\{14, 28\}$
7	$\{7, 21\}$	15	$\{15, 29\}$
8	$\{8, 22\}$		

FIGURE 5.10: Representative bits for 30 labels partitioned into $P = 15$ mutually exclusive label clusters of size $R = 2$, using $K = 2$ representative bits per label and batches of $Q = 6$ bits. The table on the right gives the label clustering. The injective mapping between labels and subsets of bits is defined by $\mathbf{g} : \ell \mapsto \{g_1(\ell) = (1 + \ell)/6, g_2(\ell) = 1 + \ell \mod 6\}$ for $\ell \in \{1, ..., 15\}$ and, for $\ell \in \{15, ..., 30\}$, it is defined by $\ell \mapsto \{(6 + g_1(\ell - 15), 6 + g_1(\ell - 15)\}$.

clusters are almost perfectly balanced, then $B/L \approx \sqrt{2/P}$. The ratio becomes more and more favorable as both Q increases and K increases up to $Q/2$, but the number of clusters P must also be large. Thus, the method should be most efficient on datasets with a very large number of labels, assuming that P increases with L in practice.

5.4.2.2 Decoding and Robustness

We now present the decoding algorithm, followed by a theoretical guarantee that each incorrectly predicted bit in the Bloom filter cannot imply more than 2 incorrectly predicted labels.

Given an example x and its predicted encoding $\hat{\mathbf{e}}(x)$, the predicted label set $d(\hat{\mathbf{e}}(x))$ is computed with the following two-step process, in which we say that a bit is "representative of one cluster" if it is a representative bit of one label in the cluster:

a. *(Cluster Identification)* For each cluster $\mathcal{L}_p$, compute its cluster score s_p defined as the number of its representative bits that are set to 1 in $\hat{\mathbf{e}}(x)$. Choose $\mathcal{L}_{\hat{p}}$ for $\hat{p} \in \underset{p \in \{1,\dots,P\}}{\operatorname{argmax}} s_p$;

b. *(Label Set Prediction)* For each label $\ell \in \mathcal{L}_{\hat{p}}$, let s'_ℓ be the number of representative bits of ℓ set to 1 in $\hat{\mathbf{e}}(x)$; add ℓ to $d(\hat{\mathbf{e}}(x))$ with probability $\frac{s'_\ell}{K}$.

In case of ties in the cluster identification, the tie-breaking rule can be arbitrary. For instance, in our experiments, we use logistic regression as base learners for binary classifiers, so we have access to posterior probabilities of being 1 for each bit of the Bloom filter. In case of ties in the cluster identification, we restrict our attention to the clusters that maximize the cluster score, and we recompute their cluster scores using the posterior probabilities instead of the binary decision. The cluster which maximizes the new cluster score is chosen. The choice of a randomized prediction for the labels avoids a single incorrectly predicted bit to result in too many incorrectly predicted labels. The robustness of the encoding/decoding scheme is proved below:

Theorem 5.1. *Let the label set $\mathcal{L}$, and let $(\mathcal{L}_1, ..., \mathcal{L}_P)$ be a partition of $\mathcal{L}$ satisfying (5.5). Assume that the encoding function satisfies Conditions 1. and 2., and that decoding is performed in the two-step process a.-b. Then, using the definitions of $\mathbb{H}^L$ and $\mathbb{H}^B$, we have:*

$$\mathbb{H}^L(d \circ \hat{\mathbf{e}}) \leq \frac{2B}{L} \mathbb{H}^B(\hat{\mathbf{e}})$$

for a K-sparse encoding, where the expectation in $\mathbb{H}^L$ is also taken over the randomized predictions.

Proof: Let (x, y) be an example. We compare the expected number of incorrectly predicted labels $H^L(y, d(\hat{\mathbf{e}}(x))) = \mathbb{E}\big[|d(\hat{\mathbf{e}}(x)) \,\Delta\, y|\big]$ (expectation taken

over the randomized prediction) and the number of incorrectly predicted bits $H^B(\hat{\mathbf{e}}(x), \mathbf{e}(y)) = \sum_{j=1}^{B} \mathbf{1}_{\{\hat{e}_j(x) \neq e_j(y)\}}$. Let us denote by p^* the index of the cluster in which y is included, and $\hat{p}$ the index of the cluster chosen in step a. We consider the two following cases:

Case 1 $\hat{p} = p^*$: if the cluster is correctly identified then each incorrectly predicted bit that is representative for the cluster costs $\frac{1}{K}$ in $H^L(y, d(\hat{\mathbf{e}}(x)))$. All other bits do not matter. We thus have $H^L(y, d(\hat{\mathbf{e}}(x))) \leq \frac{1}{K} H^B(\hat{\mathbf{e}}(x), \mathbf{e}(y))$.

Case 2 $\hat{p} \neq p^*$: If the cluster is not correctly identified, then $H^L(y, d(\hat{\mathbf{e}}(x)))$ is the sum of (1) the number of labels that should be predicted but are not ($|y|$), and (2) the labels that are in the predicted label set but that should not. To bound the ratio $\frac{H^L(y, d(\hat{\mathbf{e}}(x)))}{H^B(\hat{\mathbf{e}}(x), \mathbf{e}(y))}$, we first notice that there are at least as much representative bits predicted as 1 for $\mathcal{L}_{\hat{p}}$ than for $\mathcal{L}_{p^*}$. Since each label of $\mathcal{L}_{\hat{p}}$ shares at most $K-1$ representative bits with a label of $\mathcal{L}_{p^*}$, there are at least $|y|$ incorrect bits. Moreover, the maximum contribution to labels predicted in the incorrect cluster by correctly predicted bits is at most $\frac{K-1}{K}|y|$. Each additional contribution of $\frac{1}{K}$ in $H^L(y, d(\hat{\mathbf{e}}(x)))$ comes from a bit that is incorrectly predicted to 1 instead of 0 (and is representative for $\mathcal{L}_{\hat{p}}$). Let us denote by k the number of such contributions. Then, the most defavorable ratio $\frac{H^L(y, d(\hat{\mathbf{e}}(x)))}{H^B(\hat{\mathbf{e}}(x), \mathbf{e}(y))}$ is smaller than $\max_{k \geq 0} \frac{\frac{k}{K} + |y|(1 + \frac{K-1}{K})}{\max(|y|, k)} = \frac{\frac{|y|}{K} + |y|(1 + \frac{K-1}{K})}{|y|} = 2$.

Taking the expectation over (x, y) completes the proof ($\frac{B}{L}$ comes from normalization factors). □

5.5 Experiments

We first describe the datasets and the baselines to which we compare our work, then we investigate the behaviour of BF methods. Finally we report comparative results with state of the art baselines.

5.5.1 Datasets

We conducted experiments on the two following large scale real world datasets.

RCV-Industries is a subset of the widely used RCV dataset which only considers the industry categories. We used the first testing set file from the RCV1 site instead of the original training set since it is larger. The original RCV labels are organized as a hierarchy, here we consider leaf categories for prediction and hence reduce the number of labels to 303. We use TF/IDF features and normalize the vectors to have unity norm. This procedure has been proven to work well on text classification.

Wikipedia1k is a subsample of the wikipedia dataset release of the 2012 large scale hierarchical text classification challenge[3]. In the original dataset, roughly 10% of labels represent more than 60% of the samples and most the labels are not enough represented to allow learning accurate classifiers. We built a new dataset by retaining 1000 of the most represented labels. The features are originally word counts, again we converted the data to TF/IDF representation and normalized each data to have unity norm.

For both datasets, the testing and the validation sets have the same size while the training set is twice the size of the test set. Detailed statistics describing the main features of the datasets are presented in Table 5.1. Both datasets are publicly available online[4].

5.5.2 Evaluation metrics

We define here our few evaluation metrics using notations of section 5.3. We consider a dataset $D = \{(x_1, y_1), ..., (x_n, y_n)\}m$ where $\forall n, y_n \subset \mathcal{L}$. We note $g(x) \subset \mathcal{L}$ the output of the multilabel classifier g for an input sample x. The metrics are then defined according to (Δ is to the symmetric difference between sets):

[3]http://lshtc.iit.demokritos.gr/
[4]http://mulan.sourceforge.net/datasets.html

TABLE 5.1: Summary Statistics of the Datasets Used in the Experiments.

Statistics	**RCV-Industries**	**Wikipedia-1K**
Nb. Training examples	36167	55265
Nb. Testing examples	18084	27627
Nb. Validation examples	18084	27626
Nb. features	47236	346299
Nb. label sets	4470	47687
Nb. labels	303	1000
Label cardinality	1.3	1.11

$$HammingLoss(D, g) = \frac{1}{n}\sum_{i=1}^{n}\frac{|g(x_i)\Delta y_i|}{L}$$

$$Precision(D, g) = \frac{\sum_{i=1}^{n}|g(x_i) \cap y_i|}{\sum_{i=1}^{n}|g(x_i)|}$$

$$Recall(D, g) = \frac{\sum_{i=1}^{n}|g(x_i) \cap y_i|}{\sum_{i=1}^{n}|y_i|}$$

In addition we use F measures. The micro F measure (m-F1) is defined as:

$$microF(D, g) = \frac{2 \times Precision(D, g) \times Recall(D, g)}{Precision(D, g) + Recall(D, g)}$$

At last the macro F-measure (M-F1) is defined as the mean of the standard F measure of binary classifiers for all labels.

5.5.3 Baselines and experimental setup

We compared the three bloom filter (BF) strategies (standard Bloom Filter, standard Bloom Filter with correlation decoding and Robust Bloom Filters) to binary relevance (BR) and to three of the most notable approaches for dealing with large scale problems. The standard Bloom Filters strategies are trained similarly, but a different decoding method (and possibly different hyperparameters) is used. For BF-SD (standard decoding), Algorithm 3 is used to decode the predicted Bloom filter. Algorithm 4 is used in BF-CD (correlation decoding).

We remind the baselines used in this experimental study and previously described in the previous chapter. The first approach, which we name *BR-Dekel*, has been proposed by Dekel et al. in Dekel and Shamir (2010) to account for the fact that when the number of labels is very large many labels cannot be predicted accurately because there are not enough positive samples. *BR-Dekel* consists in first training a BR classifier, and then in applying a post-processing procedure that prunes out these noisy labels after having identified them using a validation set. Pruning means here to take the decision of never predicting a label, and thus to remove the corresponding classifier at testing time. In case one is interested in optimizing the hamming loss, the pruning rule is to eliminate any label for which the probability of a false positive (PFP) is smaller than the probability of a true positive (PTP). To fairly compare this approach to BF we operate as follows. Given a BF filter size equal to B we rank the labels based on the the ratio PTP/PFP and choose the top B labels only at inference. We use BR-Dekel as a baseline for hamming loss performance. Note that while reducing the inference complexity, this approach still has a linear complexity with respect to the number of labels in training.

The second baseline is the *Compressed Sensing* method (CS)(Hsu et al., 2009) which reduces the computational complexity by projecting the sets of labels on a low dimensional space using random matrix such as a Hadamard matrix. After learning regressors to predict the low dimensional representation, inference consists in solving a pre-image problem for each input, using a sparse recovery algorithm such as orthogonal matching pursuit (OMP) or Lasso. However, a parameter representing the number of non-zero elements to be recovered has to be passed to that algorithm, which makes this method unsuitable for Hamming loss optimization. It can however be evaluated in terms of precision-at-k, using the ranked list of outputs of the sparse recovery algorithm. In our experiments, we use OMP for decoding in our experiments and call this method CS-OMP.

As a third baseline we use the *Principle Label Space Transformation* method (PLST) which is based on SVD for dimensionality reduction rather than a random projection. Also, their is no pre-image problem to solve for recovering the original labels after predicting the low dimensional representation. PLST can exploit correlations between labels, and take classification decisions. However, this approach

is purely heuristic, and no theoretical guarantee is given.

All the methods under investigation here involve training binary classifiers or regression functions. We used the Liblinear[5] implementation of logistic regression as base binary classifier. The hyperparameters for all methods were tuned on the validation set. This corresponds to regularization parameters for each classifier in BR and our method, the number of hash functions for all BF approaches, the thresholds for BF-CD, the regularization factor and the computation of TP/FP rates for BR-Dekel, and the (L_2-)regularization factor of the ridge regressor for CS-OMP. Hyperparameters were selected to optimize the Hamming loss for all methods, except for CS-OMP for which precision@10 was used (the OMP sparsity parameter was set accordingly).

5.5.4 Parameter selection for Standard Bloom Filters

We first provide preliminary results that give some insights on the behaviour of the standard BF methods and allow to understand how the method behaves with repect to the number of hash functions K and to the size of Bloom filters B.

First we compare the behaviour of *BF-SD* and of *BF-CD* on the Wikipdia dataset. Table 5.2 reports the Hamming loss, Precision and Recall as a function of the number of hash functions K, for B fixed to 100. These results suggest that, B remaining fixed, there is an optimal tradeoff of K. As may be seen when K is small BF-SD gets a lower precision together with a higher recall. As K increases, the probability for predicting a label mechanically descreases (because the label is not predicted as soon as at least one of the corresponding K classifier incorrectly predicts 0), which makes the recall decrease and the precision increase. For BF-CD, the decoding strategy (i.e. the per-label thresholds) is optimized for minimizing the Hamming loss, and there are possibly several ways to achieve the objective. As a result precision does not steadily increase with K nor does the recall steadily decrease with K. Yet, tuning the decoding process as in BF-CD makes it possible to reach much better overall in most cases. A similar behavior of BF-CD/BF-SD with respect to K has been observed with all values of B.

[5] `http://www.csie.ntu.edu.tw/ cjlin/liblinear/`

TABLE 5.2: Comparison in (%) of Hamming Loss (HL), Precision (Prec.) and Recall (Rec.) of the two BF methods as a function of K on the Wikipedia dataset with filter size of 100.

K		2	3	4	6	8	9	10
BF-SD	HL	0.0848	0.0802	0.0818	0.0852	0.0877	0.0886	0.0889
	Prec.	75.17	91.04	92.73	93.26	93.85	93.88	94.10
	Rec.	35.24	30.80	28.58	25.00	22.48	21.61	21.23
BF-CD	HL	0.0980	0.0809	0.0797	0.0790	0.0791	0.0779	0.0784
	Prec.	61.79	89.75	81.70	82.49	82.32	86.53	82.95
	Rec.	30.04	30.35	36.09	36.29	36.30	35.12	36.67

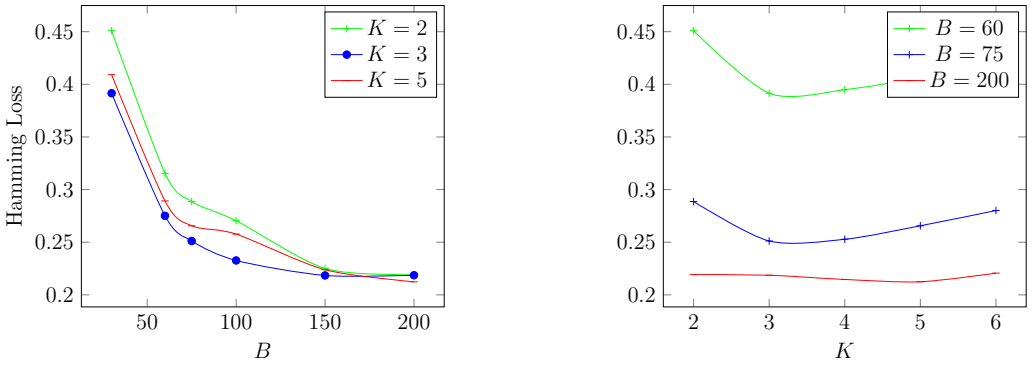


FIGURE 5.11: (left) Hamming loss as a function of the BF's size B for the *Industries* dataset. The curves correspond to various values of the number of hash function K. (right) Hamming loss as a function of the number of hash function K for the *Industries* dataset. The curves correspond to various values of BF's size B.

Next we show in figure 5.11 the joint influence of the number of hash functions K and of the size of BFs B in the performance of the system. These plots have been obtained with the BF-CD method for the *Industries* dataset. One sees that for small values (below 5), increasing K steadily increases performance whatever B (Figure 5.11 left). Yet as said above there is an optimal tradeoff (Figure 5.11 right).

5.5.5 Parameter selection for Robust Bloom Filters

The code size B can be freely set for all methods except for Robust BF, where different settings of the maximum cluster size and the number of hubs may lead

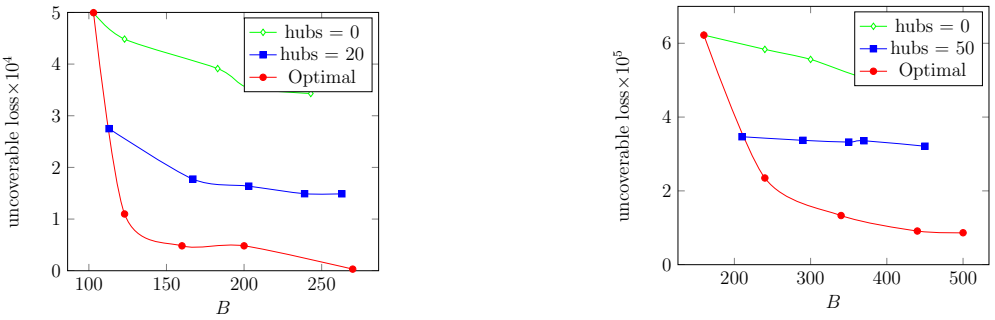


FIGURE 5.12: Unrecoverable Hamming loss (UHL) due to label clustering as a function of the code size B on *RCV-Industries (left)* and on *Wikipedia1k* dataset *(right)*. The optimal curve represents the best UHL over different settings (number of hubs,max cluster size) for a given code size. UHL decreases when the number of hubs in increased.

to the same code size. Since the use of a label clustering in R-BF leads to unrecoverable errors even if the classifiers perform perfectly well (because labels of different clusters cannot be predicted together), we chose the max cluster size among $\{10, 20, \ldots, 50\}$ and the number of hubs (among $\{0, 10, 20, 30, \ldots, 100\}$ for *RCV-Industries* and $\{0, 50, 100, \ldots, 300\}$ for *Wikipedia1k*) that minimize the resulting unrecoverable Hamming loss (UHL), computed on the train set. Figure 5.12 shows how the UHL naturally decreases when the number of hubs increases since then the method becomes closer to BR, but at the same time the overall code size B increases because it is the sum of the filter's size and the number of hubs. Nonetheless, we can observe on the figure that the UHL rapidly reaches a very low value, confirming that the label clustering assumption is reasonable in practice.

5.5.6 Correlation Decoding (CD) versus Standard Decoding (SD)

Tables 5.3 and 5.4 report comparative results with state of the art methods. Table 5.3 compares BF methods with BR and CS for the ranking loss optimized by CS, precision@k, while Table 5.4 compares BF methods to *BR* and *BR-Dekel* with respect to classification criterion, Hamming loss, micro and macro F-measures. The main comments that main be drawn from these results are the following.

TABLE 5.3: Precision @k (%) ($k \in \{1, 5, 10\}$) of Bloom Filter with Correlation Decoding (BF-CD), Compressed Sensing with Orthogonal Matching Pursuit as decoding procedure (CS-OMP) and Binary Relevance (BR) on the datasets. For each model, the number of regressors used is given in parenthesis.

Classifier	**NC**	**p@1**	**p@5**	**p@10**	**NC**	**p@1**	**p@5**	**p@10**
	RCV-Industries				**Wikipedia1K**			
BR	303	78.43	22.51	11.87	1000	66.82	17.78	09.40
CS-OMP	30	41.10	16.15	08.15	100	42.23	14.15	07.28
	60	54.78	17.49	09.07	200	57.09	15.95	08.42
	150	76.14	22.23	11.54	500	57.54	16.50	08.87
BF-CD	30	68.17	17.00	09.10	100	60.43	14.35	07.49
	60	76.41	18.73	09.91	200	62.21	14.41	07.43
	150	79.28	20.48	10.67	500	66.00	15.66	08.08

First, high precision can be achieved by BF-CD with low complexity. Indeed, it can be seen from table 5.3 that with only 100 classifiers BF-CD achieves 60% precision@1 when BR is 10% better with 1000 classifiers. For the same computational complexity (relatively to BR) of 10%, the performance of CS-OMP is 41.10% which is more than 30% less than the performance of BF-CD. In fact, in the high compression regime, the superiority of BF-CD over CS-OMP is consistent on both of the datasets for precision@k, $k \in \{1, 5, 10\}$ even though CS-OMP has been specially optimized for these values while the BF-CD were tuned to perform well in terms of hamming loss even though the two criteria can be correlated. This makes BF-CD an interesting alternative if one wants to trade some precision for much less computational complexity. It is even possible to achieve performances similar to or larger than that Binary Relevance with 50% computational complexity less. On RCV-Industries for example, BF-CD's precision@1 is 79.28% when that of BR is 78.43%. For precision@5 and precision@10, the performance of BF-CD is roughly only 2% lower than that of BR.

Correlation Decoding (CD) outperforms Standard Decoding (SD) when using Bloom Filters for multilabel classification. This appears clearly in Table 5.4 whatever the criteria and the dataset. This is particularly true in high compression settings. For example, with a hamming loss of 0.08 , BF-SD-(100)'s performance is about 13% worst than that of BR (0.0711) while BF-CD-(100)'s performance

TABLE 5.4: Test Hamming loss (HL, in %), micro (m-F1) and macro (M-F1) F1-scores. B is code size. The results of the significance test for a p-value less than 5% are denoted † to indicate the best performing method using the same B and $*$ to indicate the best performing method overall.

Classifier	B	**HL**	**m-F1**	**M-F1**	B	**HL**	**m-F1**	**M-F1**
	RCV-Industries				**Wikipedia1K**			
BR	303	0.200*	72.43*	47.82*	1000	0.0711	55.96	34.7
BR-Dekel	150	0.308	46.98	30.14	250	0.0984	22.18	12.16
	200	0.233	65.78	40.09	500	0.0868	38.33	24.52
BF-SD	150	0.223	67.45	40.29	250	0.0742	53.02	31.41
	200	0.217	68.32	40.95	500	0.0734	53.90	32.57
BF-CD	150	0.218	68.42	42.20	250	0.0726†	54.79	32.35
	200	0.212	70.07	43.37	500	0.0713	55.79	34.23
R-BF	150	0.210†	71.31†	43.44	240	0.0728†	55.85	34.65
	200	0.205†	71.86†	44.57	500	0.0705†*	57.31	36.85
CS-OMP	150	0.246	67.59	45.22†	250	0.0886	57.96†	41.84†
	200	0.245	67.71	45.82†	500	0.0875	58.46†*	42.52†*
PLST	150	0.226	68.87	32.36	250	0.0854	42.45	09.53
	200	0.221	70.35	40.78	500	0.0828	45.95	16.73

is 9% worst. We believe that as the the number of labels gets larger, the output space gets more sparse so that it is easer to reduce the computational complexity without hurting the performance. Moreover, as discussed previsouly although the performance of BF-CD and BF-SD may look similar their actual behavior is different. The restrictive decision rule of BF-SD makes its recision higher and its recall lower than that of BF-CD. Overall, BF-CD appears as a better compromise which in addition may be tuned for a particular target measure if needed.

At last it appears that removing labels does not allow as much complexity reduction as Bloom Filters. In all the experiments the performances achieved when removing labels are uniformly worst than that of BF methods. This can be explain by the fact that such variants of BR as BR-Dekel have been originally proposed to remove noisy labels. And determining the number of labels to remove in advance can result in pruning important labels (well represented) because of the criteria used to rank the labels.

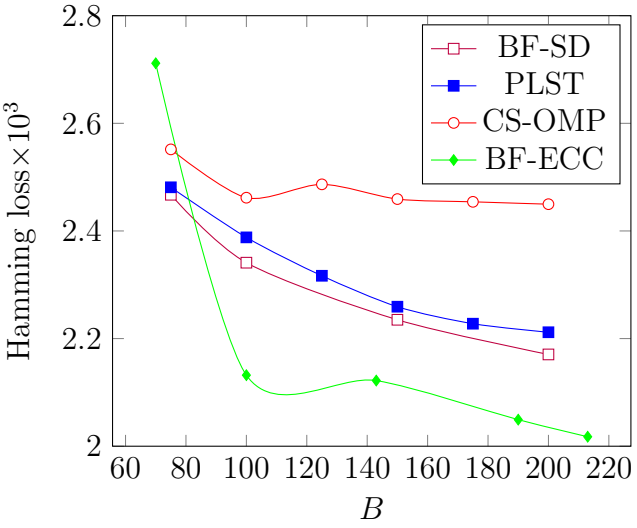


FIGURE 5.13: Hamming loss comparison of the the proposed method to the baselines while varying the code size. The Robust bloom filter is better than the other methods as the code size gets larger.

5.5.7 Comparative Results

Table 5.4 gives the test performances of all the methods on both datasets for different code sizes. We are mostly interested in the Hamming loss but we also provide the micro and macro F-measure. The results are averaged over 10 random splits of train/validation/test of the datasets, respectively containing 50%/25%/25% of the data. The standard deviations of the values are negligible (smaller than 10^{-3} times the value of the performance measure). Our Robust Bloom FIlter method seem to clearly outperform all other methods and yields significant improvements over Standard Bloom FIlter approaches. On *Wikipedia1k*, with 500 classifiers, the Hamming loss (in %) of R-BF is only 0.0705. This performance is similar to that of BR's (0.0711) which uses twice as many classifiers. The closer runner-up is consistently the standard Bloom FIlter approach with correlation decoding (BF-CD). The simple pruning strategy *BR-Dekel* is the worst baseline on both datasets, confirming that considering all classes is necessary on these datasets.

CS-OMP reaches a much higher Hamming loss (about 23% worst than BR on both datasets when using 50% less classifiers). CS-OMP achieves the best performance on the macro-F measure though. This is because the size of the predicted label sets

is fixed for CS, which increases recall but leads to poor precision. We used OMP as decoding procedure for CS since it seemed to perform better than Lasso and Correlation decoding (CD)(Hsu et al., 2009)(for instance, on *RCV-Industries* with a code size of 500, OMP achieves a Hamming loss of 0.0875 while the Hamming loss is 0.0894 for Lasso and 0.1005 for CD). PLST improves over CS-OMP but its performances are lower than those of S-BF (about 3.5% on RCV-industries and 13% and Wikipedia when using 50% less classifiers than BR). The macro F-measure indicates that PLST likely suffers from class imbalance (only the most frequent labels are predicted), probably because the label set matrix on which SVD is performed is dominated by the most frequent labels.

Figure 5.13 gives the general picture of the Hamming loss of the methods on a larger range of code sizes on the RCV-Industries dataset. Overall, R-BF has the best performances except for very small code sizes because the Unrecoverable Hamming Loss (UHL) becomes too high.

5.5.8 Runtime analysis

Experiments were performed on a computer with 24 intel Xeon 2.6 GHz CPUs. For all methods, the overall training time is dominated by the time to train the binary classifiers or regressors, which depends linearly on the code size. For test, the time is also dominated by the classifiers' predictions, and the decoding algorithm of R-BF is the fastest. For instance, on *Wikipedia1k*, training one binary classifier takes $12.35s$ on average, and inference with one classifier (for the whole test dataset) takes $3.18s$. Thus, BR requires about 206 minutes ($1000 \times 12.35s$) for training and $53m$ for testing on the whole test set. With $B = 500$, R-BF requires about half that time, including the selection of the number of hubs and the max. cluster size at training time, which is small (computing the UHL of a R-BF configuration takes $9.85s$, including the label clustering step, and we try less than 50 of them). For the same B, encoding for CS takes $6.24s$ and the SVD in PSLT takes $81.03s$, while decoding takes $24.39s$ at test time for CS and $7.86s$ for PSLT.

5.6 Conclusion

Standard Bloom Filters (with standard and correlation decoding) are a simple and efficient way of doing extreme MLC. Even though they do not come with performance guarantees and they can suffer from the mistakes of binary classifiers, their empirical performances are competitive to the main baselines proposed in extreme MLC literature. Robust Bloom Filters improve the standard Bloom Filters by exploiting the label correlation information. They provide a more robust encoding/decoding procedure together with performance guarantees. Moreover, their empirical performances compare favourably to those of the baselines and also to the performances of Binary Relevance.

In summary, Bloom Filters offer a nice a framework for Extreme MLC. They share the desirable properties of Binary Relevance such as simplicity, accuracy and ease of parallelization with the important additional property of scalability which is a main requirement in extreme classification. Hence they allow to take the best of both worlds. Still, there is room for many improvements. For example, some MLC problems with many labels have large label cardinality. This is the case of the Delicious dataset[6]. For this type of problem, the label clustering assumption does not hold because the label density is large (the label density is about 0.019 for Delicious which has 1000 labels while while it is about 10^{-4} for the wikipedia1K dataset which has the same number of labels). Other kinds of Bloom Filters such as counting Bloom Filters (Broder et al., 2002) could be better alternative for such.

[6]http://mulan.sourceforge.net/datasets.html

Chapter 6

Conclusion and Perspectives

In this thesis we have tackled the *extreme classification* problem. The presence of a large number of labels gives rise to a vast spectrum of challenging problems, however we have mainly been interested into reducing training and inference complexity. This is important as it makes learning feasible and allows fast inference hence making the models exploitable in real world applications. We have studied existing methods and proposed solutions for reducing training and inference complexity in both single label and multi label settings. For each of these two settings a review of the main approaches that have been proposed in the literature is presented (respectively in chapter 2 and chapter 4). These approaches are mainly of two kinds. On one hand, the embedding based methods rely on learning a low dimensional representation of the labels. Regressors are then learned to predict this new representation. At inference, for a given new example, original labels are recovered once its corresponding low dimensional representation is predicted. On the other hand, the hierarchical approaches learn hierarchical classifiers using a learned hierarchical structure or an existing one. Inference is then achieved by evaluating the classifiers using a depth first search procedure until a leaf node is reached.

Hierarchical classifiers are most popular in extreme single label classification. They yield competitive performances (Bengio et al., 2010, Bennett and Nguyen, 2009, Deng et al., 2011, Gao and Koller, 2011a) and allow logarithmic complexity (in

the number of labels) at prediction time when the hierarchy is a balanced tree. However, they suffer from training complexity burden because the number of classifiers to train (and store) equals the number of nodes in the hierarchy which can be very large for taxonomies such as the Wikipedia graph[1]. Moreover, they are less naturally suited for multi label classification.

Embedding based methods allow complexity reduction at both training and prediction time because the number of classifiers to train and to evaluate at prediction can be much smaller than the number of labels. Moreover, there are embedding based approaches for solving both single label (Langford and Beygelzimer, 2005, Weinberger and Chapelle, 2008) and multilabel (Hsu et al., 2009, Tai and Lin, 2012) extreme classification. They also allow the use of prior knowledge such as hierarchical information in single label setting and label dependence information in the multilabel setting. Therefore, they are a good compromise for complexity reduction in extreme classification problems.

We have first dealt with extreme single label classification by proposing to learn compact class codes that allow fast inference. Our approach leverages hierarchical information and learns low dimensional binary representation of the labels that were empirically proven to yield easier binary induced problem compared to randomly generated error correcting output codes. Overall, it results in competitive classification performances compared to classical baselines such as One versus Rest approach on large scale experiments. This work has been published at the European Conference of Machine Learning (ECML-PKDD) in 2012 (Cissé et al., 2012).

We have then tackled extreme multilabel classification by introducing a new framework based on Bloom Filters. Our contribution here is two fold. First, we used Standard Bloom Filters as a way to encode sets of labels hence ending with a sparse low dimensional binary representation of the labels and the sets of labels. As with previously presented approaches, regressors are learned to predict this low dimensional representation and two algorithms are presented for mapping a predicted binary vector to a set of relevant labels. The first approach gives promising results but suffers from its frailty to individual errors of binary classifiers. To overcome

[1] http://lshtc.iit.demokritos.gr

this problem, a second approach is proposed that exploits an important feature of extreme classification problems: many labels never appear together when the number of labels is very large. We termed this feature *label clustering* and exploited it to build new theoretically grounded encoding and decoding schemes for Bloom Filters that are robust to individual errors of binary classifiers. The resulting method, called robust Bloom Filter, compares favourably with other embedding based approaches such as Compressed Sensing (Hsu et al., 2009) and is competitive in comparison with the classical binary relevance method. Several experiments are also presented to gain more insight in the behaviour of the proposed approaches. Part of this work has been published at Neural Information Processing Systems conference (NIPS) 2013 (Cisse et al., 2013) and an extend version is in preparation for the Machine Leaning Journal.

Reducing training and inference complexity in extreme classification is of fundamental importance and has witnessed significant progress in the past few years. Several hierarchical and embedding based methods have been introduced that reduce the complexity and increase classification performances in both the single label and the multilabel setting. However, there are several other problems of importance that must be addressed in order to solve extreme classification. Discriminative representation learning, class imbalance and data scarcity and label synonymy are few examples of such problems. However, we have identified two problems that deserve more attention in our opinion. The first one is the design of new methods for efficiently harvesting labels for extreme classification datasets. Indeed, the generalization performance of the algorithms proposed depend heavily on the quality of the labelling process. Currently, these labels are mainly obtained from collaborative websites such as Wikipedia. While this process is cheaper than using mechanical turk[2] (AMT) for example it is prone to noisy labelling. The second problem is the analysis of existing performance measures and the design of new ones that are better suited for extreme classification problems. There have been some work in this direction (Kosmopoulos et al., 2013). However, this needs to be pushed further and will be the subject of our future research.

[2]https://www.mturk.com/mturk/welcome

Bibliography

Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Lecture Notes in Computer Science*, pages 420–434. Springer, 2001.

Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 13–24. International World Wide Web Conferences Steering Committee, 2013.

Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *J. Mach. Learn. Res.*, 1: 113–141, September 2001. ISSN 1532-4435. doi: 10.1162/15324430152733133. URL http://dx.doi.org/10.1162/15324430152733133.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Mach. Learn.*, 73(3):243–272, December 2008. ISSN 0885-6125. doi: 10.1007/s10994-007-5040-8. URL http://dx.doi.org/10.1007/s10994-007-5040-8.

Krishnakumar Balasubramanian and Guy Lebanon. The landmark selection method for multiple output prediction. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 983–990, New York, NY, USA, July 2012. Omnipress. ISBN 978-1-4503-1285-1.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396, June 2003.

ISSN 0899-7667. doi: 10.1162/089976603321780317. URL http://dx.doi.org/10.1162/089976603321780317.

Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 163–171. 2010.

Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009. ISSN 1935-8237.

Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux, Jean-François Paiement, Pascal Vincent, and Marie Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. *Neural Comput.*, 16(10):2197–2219, October 2004. ISSN 0899-7667. doi: 10.1162/0899766041732396. URL http://dx.doi.org/10.1162/0899766041732396.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. 2013.

Paul N. Bennett and Nam Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 11–18, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-483-6. doi: 10.1145/1571941.1571946. URL http://doi.acm.org/10.1145/1571941.1571946.

Paul N. Bennett, Susan T. Dumais, and Eric Horvitz. Probabilistic combination of text classifiers using reliability indicators: models and results. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 207–214, New York, NY, USA, 2002. ACM. ISBN 1-58113-561-0. doi: 10.1145/564376.564413. URL http://doi.acm.org/10.1145/564376.564413.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. volume 18, pages 509–517. ACM, September 1975.

Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *In Int. Conf. on Database Theory*, pages 217–235, 1999.

Alina Beygelzimer, John Langford, Yury Lifshits, Gregory B. Sorkin, and Alexander L. Strehl. Conditional probability tree estimation analysis and algorithms. *CoRR*, abs/0903.4217, 2009a.

Alina Beygelzimer, John Langford, and Pradeep D. Ravikumar. Error-correcting tournaments. *CoRR*, abs/0902.3176, 2009b.

Wei Bi and James Kwok. Efficient multi-label classification with many labels. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 405–413. JMLR Workshop and Conference Proceedings, May 2013.

Irving Biederman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94:115–147, 1987.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment.*, 10, 2008.

Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

Antoine Bordes, Léon Bottou, Patrick Gallinari, and Jason Weston. Solving multiclass support vector machines with larank. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 89–96, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3.

Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc), 2008. URL http://leon.bottou.org/papers/bottou-bousquet-2008.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

Andrei Broder, Michael Mitzenmacher, and Andrei Broder I Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a ?siamese? time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 78–87. ACM, 2004.

Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 59–65, New York, NY, USA, 1978. ACM.

Rich Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, July 1997. ISSN 0885-6125. doi: 10.1023/A:1007379606734. URL http://dx.doi.org/10.1023/A:1007379606734.

Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Incremental algorithms for hierarchical classification. *J. Mach. Learn. Res.*, 7:31–54, December 2006. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1248547.1248549.

Yangchi Chen, Melba M. Crawford, and Joydeep Ghosh. Integrating support vector machines in a hierarchical output decomposition framework. In *In 2004 International Geosci. and Remote Sens. Symposium*, pages 949–953, 2004.

Yao-Nan Chen and Hsuan-Tien Lin. Feature-aware label space dimension reduction for multi-label classification. In *NIPS*, pages 1538–1546, 2012.

Ken Christensen, Allen Roginsky, and Miguel Jimeno. A new analysis of the false positive rate of a bloom filter. *Inf. Process. Lett.*, 110(21):944–949, October 2010. ISSN 0020-0190.

M. Cissé, T. Artières, and Patrick Gallinari. Learning compact class codes for fast inference in large multi class classification. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECML PKDD'12, pages 506–520, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33459-7. doi: 10.1007/978-3-642-33460-3_38. URL http://dx.doi.org/10.1007/978-3-642-33460-3_38.

Moustapha M Cisse, Nicolas Usunier, Thierry Artières, and Patrick Gallinari. Robust bloom filters for large multilabel classification tasks. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1851–1859. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5083-robust-bloom-filters-for-large-multilabel-classification-tasks.pdf.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, March 2002. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944790.944813.

Ofer Dekel and Ohad Shamir. Multiclass-multilabel classification with more classes than examples. volume 9, pages 137–144, 2010.

Ofer Dekel, Joseph Keshet, and Yoram Singer. Large margin hierarchical classification. In *Proceedings of the Twenty-first International Conference on Machine*

Learning, ICML '04, pages 27–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015374. URL http://doi.acm.org/10.1145/1015330.1015374.

Krzysztof Dembczynski, Weiwei Cheng, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, pages 279–286, 2010a.

Krzysztof Dembczynski, Weiwei Cheng, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, pages 279–286, 2010b.

Krzysztof Dembczynski, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88(1-2):5–45, 2012.

Jia Deng, Sanjeev Satheesh, Alexander C. Berg, and Fei Fei F. Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 567–575. 2011.

Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *J. Artif. Intell. Res. (JAIR)*, 2:263–286, 1995.

Susan Dumais and Hao Chen. Hierarchical classification of web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '00, pages 256–263, New York, NY, USA, 2000. ACM. ISBN 1-58113-226-3. doi: 10.1145/345508.345593. URL http://doi.acm.org/10.1145/345508.345593.

Sergio Escalera, Oriol Pujol, and Petia Radeva. Error-correcting ouput codes library. *The Journal of Machine Learning Research*, 11:661–664, 2010.

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 109–117, New York, NY, USA,

2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014067. URL http://doi.acm.org/10.1145/1014052.1014067.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9: 1871–1874, June 2008. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1390681.1442794.

T. Gao and D. Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011a.

Tianshi Gao and Daphne Koller. Multiclass boosting with hinge loss based on output coding. In *ICML*, pages 569–576, 2011b.

Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *In Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 22–30. Springer, 2004.

Shantanu Godbole, Sunita Sarawagi, and Soumen Chakrabarti. Scaling multi-class support vector machines using inter-class confusion. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 513–518, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: 10.1145/775047.775122. URL http://doi.acm.org/10.1145/775047.775122.

Siddharth Gopal and Yiming Yang. Distributed training of large-scale logistic models. In *ICML (2)*, JMLR Proceedings, pages 289–297. JMLR.org.

Siddharth Gopal and Yiming Yang. Recursive regularization for large-scale classification with hierarchical and graphical dependencies. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '13, pages 257–265, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2487644. URL http://doi.acm.org/10.1145/2487575.2487644.

Siddharth Gopal, Yiming Yang, Bing Bai, and Alexandru Niculescu-Mizil. Bayesian models for large-scale hierarchical classification. In P. Bartlett, F.C.N.

Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2420–2428. 2012.

Gregory Griffin and Pietro Perona. Learning and using taxonomies for fast visual categorization. In *CVPR*. IEEE Computer Society, 2008. URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2008.html#GriffinP08.

Bharath Hariharan, S. V. N. Vishwanathan, and Manik Varma. Large Scale Max-Margin Multi-Label Classification with Prior Knowledge about Densely Correlated Labels. In *Proceedings of International Conference on Machine Learning*, 2010.

Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In *NIPS*, pages 772–780, 2009.

Sung Ju J. Hwang, Kristen Grauman, and Fei Sha. Learning a tree of metrics with disjoint visual features. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 621–629. 2011.

Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intell. Data Anal.*, 6(5):429–449, October 2002. ISSN 1088-467X. URL http://dl.acm.org/citation.cfm?id=1293951.1293954.

Ashish Kapoor, Raajay Viswanathan, and Prateek Jain. Multilabel classification using bayesian compressed sensing. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Leon Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 2654–2662, 2012.

Adam R. Klivans and Alexander A. Sherstov. Improved lower bounds for learning intersections of halfspaces. In *Proceedings of the 19th annual conference on Learning Theory*, COLT'06, pages 335–349, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-35294-5, 978-3-540-35294-5. doi: 10.1007/11776420_26. URL http://dx.doi.org/10.1007/11776420_26.

Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on*

Machine Learning, ICML '97, pages 170–178, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-486-3. URL http://dl.acm.org/citation.cfm?id=645526.657130.

Aris Kosmopoulos, Ioannis Partalas, Éric Gaussier, Georgios Paliouras, and Ion Androutsopoulos. Evaluation measures for hierarchical classification: a unified view and novel approaches. *CoRR*, abs/1306.6802, 2013. URL http://arxiv.org/abs/1306.6802.

Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Learning and inference in probabilistic classifier chains with beam search. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECML PKDD'12, pages 665–680, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33459-7. doi: 10.1007/978-3-642-33460-3_48. URL http://dx.doi.org/10.1007/978-3-642-33460-3_48.

John Langford and Alina Beygelzimer. Sensitive error correcting output codes. In Peter Auer and Ron Meir, editors, *Learning Theory*, volume 3559 of *Lecture Notes in Computer Science*, pages 158–172. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26556-6. doi: 10.1007/11503415_11. URL http://dx.doi.org/10.1007/11503415_11.

Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. Zero-data learning of new tasks. In *AAAI*, volume 8, pages 646–651, 2008.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. volume 86, pages 2278–2324. IEEE, 1998.

Song Liu, Haoran Yi, Liang-Tien Chia, and Deepu Rajan. Adaptive hierarchical multi-class svm classifier for texture-based image classification. In *ICME*, pages 1190–1193. IEEE, 2005a. URL http://dblp.uni-trier.de/db/conf/icmcs/icme2005.html#LiuYCR05.

Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explor. Newsl.*, 7(1):36–43, June 2005b. ISSN 1931-0145. doi: 10.1145/1089815.1089821. URL http://doi.acm.org/10.1145/1089815.1089821.

Zhigang Liu, Wenzhong Shi, Qianqing Qin, Xiaowen Li, and Donghui Xie. Hierarchical support vector machines. In *IGARSS*, page 4. IEEE, 2005c. ISBN 0-7803-9050-4. URL http://dblp.uni-trier.de/db/conf/igarss/igarss2005.html#LiuSQLX05.

Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17 (4):395–416, December 2007. ISSN 0960-3174. doi: 10.1007/s11222-007-9033-z. URL http://dx.doi.org/10.1007/s11222-007-9033-z.

Marcin Marszalek and Cordelia Schmid. Semantic Hierarchies for Visual Object Recognition. In *IEEE Conference on Computer Vision & Pattern Recognition (CVPR '07)*, page 1, Minneapolis, United States, 2007. IEEE Computer Society. doi: 10.1109/CVPR.2007.383272. URL http://hal.inria.fr/inria-00548680.

Marcin Marszalek and Cordelia Schmid. Constructing category hierarchies for visual recognition. In *In ECCV08 pages IV: 47991*, 2008.

Andrew McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 359–367, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8. URL http://dl.acm.org/citation.cfm?id=645527.657461.

Luis Carlos Molina, Lluís Belanche, and Àngela Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 306–313. IEEE, 2002.

Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Math. Program.*, 120(1):221–259, April 2009. ISSN 0025-5610. doi: 10.1007/s10107-007-0149-x. URL http://dx.doi.org/10.1007/s10107-007-0149-x.

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.

Mark Palatucci, Dean Pomerleau, Geoffrey E Hinton, and Tom M Mitchell. Zero-shot learning with semantic output codes. In *NIPS*, volume 3, pages 5–2, 2009.

Francesco Pappalardo, Cristiano Calonaci, Marzio Pennisi, Emilio Mastriani, and Santo Motta. Hamfast: Fast hamming distance computation. In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 01*, CSIE '09, pages 569–572, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3507-4.

Y. C. Pati, R. Rezaiifar, Y. C. Pati R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27 th Annual Asilomar Conference on Signals, Systems, and Computers*, pages 40–44, 1993.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, ECML PKDD '09, pages 254–269, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-04173-0. doi: 10.1007/978-3-642-04174-7_17. URL http://dx.doi.org/10.1007/978-3-642-04174-7_17.

Jesse Read, Luca Martino, and David Luengo. Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, 47(3): 1535–1546, 2014.

Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141, December 2004a. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1005332.1005336.

Ryan M. Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004b.

Robert E. Schapire. Using output codes to boost multiclass learning problems. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 313–321, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-486-3. URL http://dl.acm.org/citation.cfm?id=645526.657134.

Bernhard Schölkopf, Chris Burges, and Vladimir Vapnik. Extracting support data for a given task. In *KDD*, pages 252–257, 1995.

Carlos N. Silla, Jr. and Alex A. Freitas. A survey of hierarchical classification across different application domains. *Data Min. Knowl. Discov.*, 22(1-2):31–72, January 2011. ISSN 1384-5810. doi: 10.1007/s10618-010-0175-9. URL http://dx.doi.org/10.1007/s10618-010-0175-9.

Aixin Sun and Ee-Peng Lim. Hierarchical text classification and evaluation. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 521–528, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1119-8. URL http://dl.acm.org/citation.cfm?id=645496.657884.

Farbound Tai and Hsuan-Tien Lin. Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542, 2012.

Lorenzo Torresani and Kuang chih Lee. Large margin component analysis. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1385–1392. MIT Press, Cambridge, MA, 2007.

Konstantin Tretyakov. Machine learning techniques in spam filtering. 2004.

Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *IJDWM*, 3(3):1–13, 2007.

Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Random k-labelsets for multi-label classification. *IEEE Transactions on Knowledge and Data Engineering*, 2010. ISSN 1041-4347. doi: 10.1109/TKDE.2010.164.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94559-8.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

Volkan Vural and Jennifer G. Dy. A hierarchical method for multi-class support vector machines. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 105–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015427. URL http://doi.acm.org/10.1145/1015330.1015427.

Andreas S. Weigend, Erik D. Wiener, and Jan O. Pedersen. Exploiting hierarchy in text categorization. *Inf. Retr.*, 1(3):193–216, October 1999. ISSN 1386-4564. doi: 10.1023/A:1009983522080. URL http://dx.doi.org/10.1023/A:1009983522080.

Kilian Q. Weinberger and Olivier Chapelle. Large margin taxonomy embedding for document categorization. In *NIPS*, pages 1737–1744, 2008.

Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. volume 10, pages 207–244. JMLR.org, 2009.

Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, volume 9, page 6, 2008.

J. Weston and C. Watkins. Multi-class support vector machines, 1998. URL citeseer.nj.nec.com/8884.html.

J. Weston and C. Watkins. Support vector machines for multiclass pattern recognition. In *Proceedings of the Seventh European Symposium On Artificial Neural Networks*, 1999.

Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Proceedings of the Twenty-Second International*

Joint Conference on Artificial Intelligence - Volume Volume Three, IJCAI'11, pages 2764–2770. AAAI Press, 2011. ISBN 978-1-57735-515-1.

Christian Widmer, Jose Leiva, Yasemin Altun, and Gunnar Rätsch. Leveraging sequence classification by taxonomy-based multitask learning. In *Proceedings of the 14th Annual international conference on Research in Computational Molecular Biology*, RECOMB'10, pages 522–534, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-12682-0, 978-3-642-12682-6. doi: 10.1007/978-3-642-12683-3_34. URL http://dx.doi.org/10.1007/978-3-642-12683-3_34.

Lin Xiao, Dengyong Zhou, and Mingrui Wu. Hierarchical classification via orthogonal transfer. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 801–808, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

Yiming Yang, Jian Zhang, and Bryan Kisiel. A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 96–103, New York, NY, USA, 2003. ACM. ISBN 1-58113-646-3. doi: 10.1145/860435.860455. URL http://doi.acm.org/10.1145/860435.860455.

Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Mach. Learn.*, 85(1-2): 41–75, October 2011. ISSN 0885-6125.

Lihi Zelnik-manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*, pages 1601–1608. MIT Press, 2004.

Ning Zhang, Ling-Yu Duan, Qingming Huang, Lingfang Li, Wen Gao, and Ling Guan. Automatic video genre categorization and event detection techniques on large-scale sports data. In Joanna W. Ng, Christian Couturier, Hausi A. Muller, and Arthur G. Ryman, editors, *CASCON*, pages 283–297. ACM, 2010. URL http://dblp.uni-trier.de/db/conf/cascon/cascon2010.html#ZhangDHLGG10.

Yi Zhang and Jeff Schneider. Multi-label output codes using canonical correlation analysis. In *AISTATS 2011*, 2011.

Bin Zhao and Eric P. Xing. Sparse output coding for large-scale visual recognition. In *CVPR*, pages 3350–3357. IEEE, 2013. URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2013.html#ZhaoX13.