Scalable algorithms for large-scale machine learning problems : Application to multiclass classification and asynchronous distributed optimization

Bikash Joshi

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITE GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Bikash JOSHI

Thèse dirigée par **Massih-Reza AMINI**, Professeur, Université Grenoble Alpes, et
codirigée par **Franck IUTZELER** Maitre de Conférences , Université Grenoble Alpes

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

Algorithmes d'apprentissage pour les grandes masses de données : Application à la classification multi-classes et à l'optimisation distribuée asynchrone

Scalable Algorithms for Large-scale Machine Learning Problems:Application to Multiclass Classification and Asynchronous Distributed Optimization

Thèse soutenue publiquement le **26 septembre 2017**,
devant le jury composé de :

Monsieur MASSIH-REZA AMINI
PROFESSEUR, UNIVERSITÉ GRENOBLE ALPES, Directeur de thèse
Monsieur FRANK IUTZELER
MAITRE DE CONFÉRENCES ,UNIVERSITÉ GRENOBLE ALPES, Co-directeur de thèse
Monsieur STEPHANE CANU
PROFESSEUR, INSA ROUEN, Rapporteur
Monsieur THIERRY ARTIERES
PROFESSEUR, ECOLE CENTRALE DE MARSEILLE, Rapporteur
Monsieur JERÔME MALICK
CHERCHEUR, CNRS DELEGATION ALPES, Président
Madame MARIANNE CLAUSEL
PROFESSEUR, UNIVERSITE DE LORRAINE, Examinateur

Abstract

This thesis focuses on developing scalable algorithms for large scale machine learning. In this work, we present two perspectives to handle large data. First, we consider the problem of large-scale multiclass classification. We introduce the task of multiclass classification and the challenge of classifying with a large number of classes. To alleviate these challenges, we propose an algorithm which reduces the original multiclass problem to an equivalent binary one. Based on this reduction technique, we introduce a scalable method to tackle the multiclass classification problem for very large number of classes and perform detailed theoretical and empirical analyses.

In the second part, we discuss the problem of distributed machine learning. In this domain, we introduce an asynchronous framework for performing distributed optimization. We present application of the proposed asynchronous framework on two popular domains: matrix factorization for large-scale recommender systems and large-scale binary classification. In the case of matrix factorization, we perform Stochastic Gradient Descent (SGD) in an asynchronous distributed manner. Whereas, in the case of large-scale binary classification we use a variant of SGD which uses variance reduction technique, SVRG as our optimization algorithm.

Résumé

L'objectif de cette thèse est de développer des algorithmes d'apprentissage adaptés aux grandes masses de données. Dans un premier temps, nous considérons le problème de la classification avec un grand nombre de classes. Afin d'obtenir un algorithme adapté à la grande dimension, nous proposons un algorithme qui transforme le problème multi-classes en un problème de classification binaire que nous sous-échantillonnons de manière drastique. Afin de valider cette méthode, nous fournissons une analyse théorique et expérimentale détaillée. Dans la seconde partie, nous approchons le problème de l'apprentissage sur données distribuées en introduisant un cadre asynchrone pour le traitement des données. Nous appliquons ce cadre à deux applications phares : la factorisation de matrice pour les systèmes de recommandation en grande dimension et la classification binaire.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my thesis supervisor, Massih-Reza Amini, for his continuous support and guidance for this thesis. I really appreciate his knowledge and kindness which made this work possible and always allowed me to work in a stress free environment. I also like to thank my co-supervisor Franck Iutzeler for his valuable guidance and motivation for this thesis. I was always impressed by his brilliance and energy for research.

My appreciation also extends to my jury members for their interesting questions and comments during my thesis defense. Moreover, I would like to thank my theis reviewers for carefully reviewing my thesis report and giving very insightful comments, which immensely helped me to improve the quality of my thesis report.

I am also thankful to the AMA laboratory for providing me the required resources and workspace for conducting my research. I also like to thank my lab colleagues, with whom I spent a wonderful time.

Last but not at all the least, I would like express my gratitude to all my family members for always motivating me and believing in my abilities. Especially, I am indebted to my mother, for her continuous love and support through my entire life.

CONTENTS

List of Figures

List of Tables

1 INTRODUCTION

1.1 BACKGROUND

Machine Learning (ML) algorithms help to extract useful information from data, so that it can be utilized in future for useful tasks such as prediction or clustering. ML algorithms work by learning a model or pattern from the data and using the model to discover useful information for the future unseen data. These algorithms have proved to have a huge social and economic impact for different stakeholders. This has fueled the popularity of ML in applications such as document analysis, computer vision, natural language processing, voice recognition, recommendation, ranking and many others.

In the last decade we have seen an exponential growth in the quantity of data, mainly due to the popularity of digital technologies. Some of the areas where such large scale data collections are prevalent are Computer Vision, Recommender Systems, Information Retrieval, Social Networks, etc. Efficiently handling and effectively exploiting such large magnitude of data has opened a new area of research. Also such large magnitude of data has posed a serious challenge for the traditional ML techniques. So, there is a need to adapt and improve the existing ML techniques to scale well and cope up with the new applications.

As the title suggests, in this thesis we intend to analyze the challenges of large-scale data in ML and devise effective algorithms to tackle it. In our study we consider two different areas involving large-scale data. Even though they are two separate perspectives, they both come under the umbrella of large-scale ML. First, we study and analyze the problem of large-scale multiclass classification. Second, we study distributed computing for handling large-scale data.

1.1.1 LARGE-SCALE MULTICLASS CLASSIFICATION

Multiclass classification refers to classification problems where we need to classify an example to one of finite set of categories. The goal of these algorithms is to learn a function which, given a new example will correctly assign a class label. Some of the popular applications of multiclass classification are text, image or video classification. Traditionally, multiclass classification problems involved at most hundreds of classes, however, in the past few years we have observed a spectacular increase in data thanks to the popularity of internet and social media websites such as Facebook [1], Wikipedia [2], Flickr [3], Youtube [4] etc. As for example there are around thousands of new articles added to Wikipedia [5] every day and each of them has to be categorized to one of millions of categories.

Similarly, recent applications involving text or image classification has to deal with very large number of classes (upto millions), hence this is also referred as extreme classification. Extreme classification poses several challenges to the existing approaches of multiclass classification. First set of challenges are related to the computational complexity of the algorithms. More precisely, learning a model using such large magnitude of data significantly increases the total runtime of the algorithm as well as the total memory used during the learning process. Another set of challenges are introduced from the underlying properties of such large scale data collections. First of which is known as class imbalance problem. In large-scale multiclass collections it is observed that most of the classes has very few representative examples as shown in Figure 1.1. Such distribution is referred to as long-tailed distribution. Presence of such class imbalance in data distribution affects the predictive performance of algorithms in large-scale collections, since it is difficult to properly predict the rare classes (classes with few examples). Similarly, large-scale datasets involve high dimensionality and large sample size, which also makes learning a challenging task.

[1] www.facebook.com
[2] http://www.wikipedia.org
[3] https://www.flickr.com
[4] https://www.youtube.com/
[5] https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

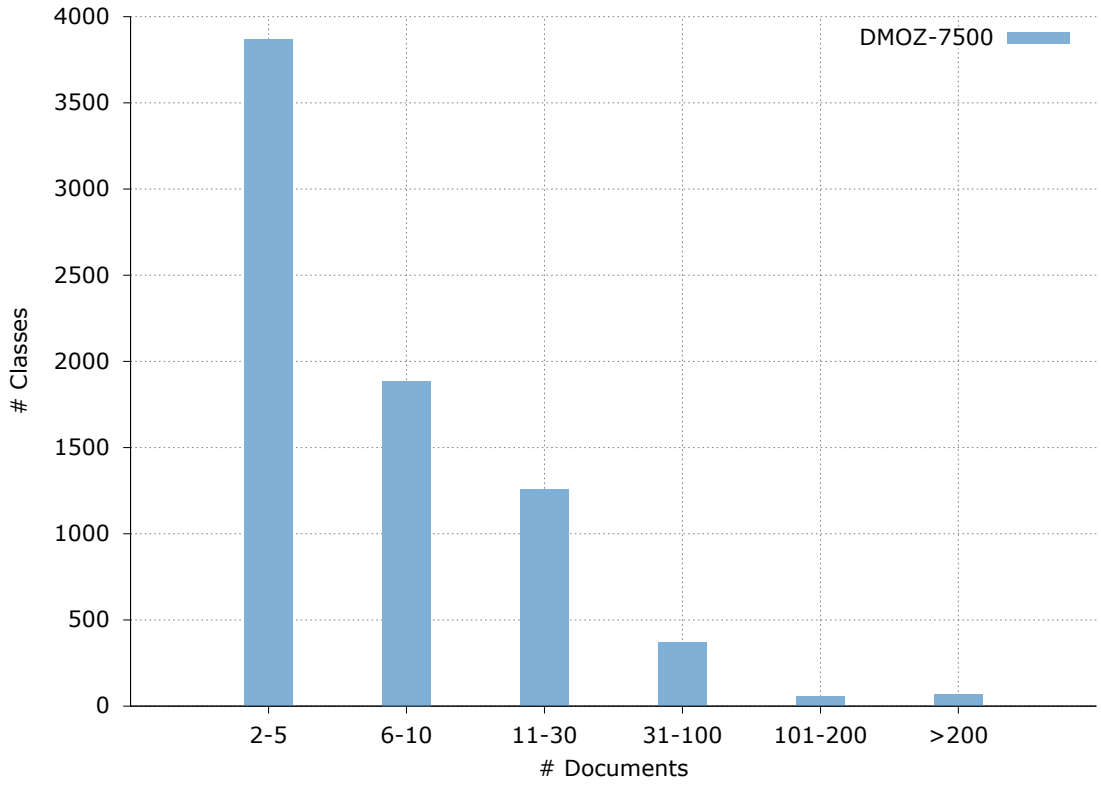


Figure 1.1: Distribution of classes in DMOZ dataset

1.1.2 Distributed Computing

As mentioned earlier, the main challenge we are faced in last few years is the rapidly increasing size of training data and model complexity. As for example we can see in 1.2, the growth of size of data over the years [6]. It is evident from the exponential growth of the size of data that, it is difficult to store such huge amount of data in one single machine. Even if we can keep such data in a single machine it will take tediously long time to perform learning in a sequential manner as in traditional machine learning approaches. So, with the rapid development and availability of computing resources, the obvious solution to tackle this problem is to adopt distributed techniques.

In distributed computing, the data is partitioned and dispatched across several machines and learning is performed simultaneously. In such a setting, each of the computing nodes have their own memory and processing units. The memory is not shared between the machines in distributed setting. This is different from commonly used technology known as shared-memory parallel computing, which is often confused with distributed computing. In parallel computing each of the computing units

[6] http://www.unece.org

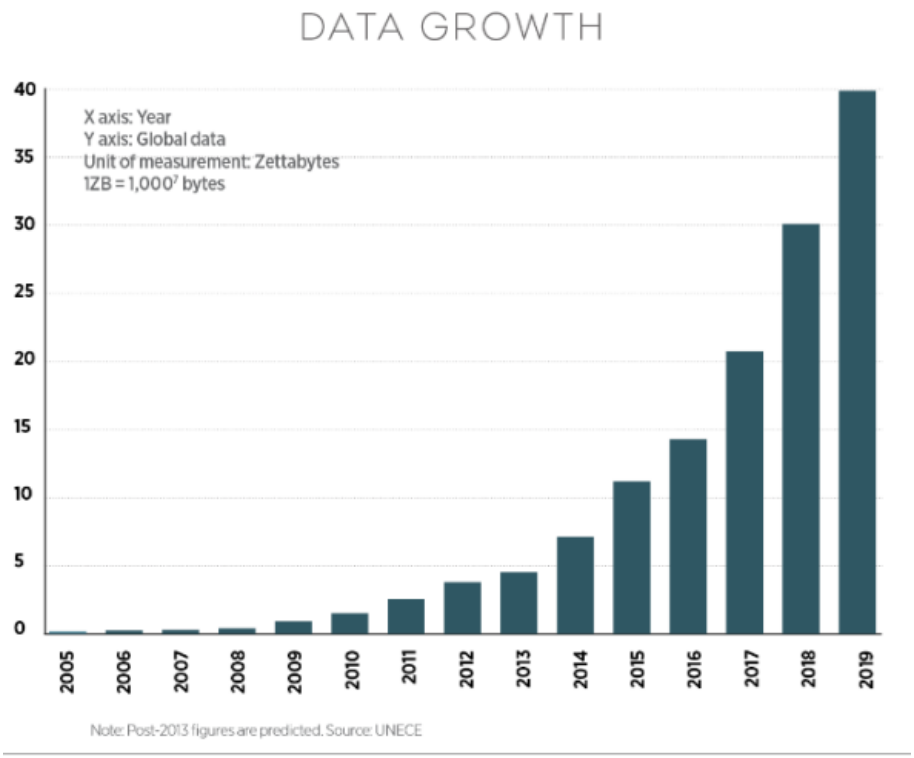


Figure 1.2: Evolution of size of big data over years

Source: http://www.unece.org

have access to the same shared memory. These two different settings are depicted in Figure 1.3.

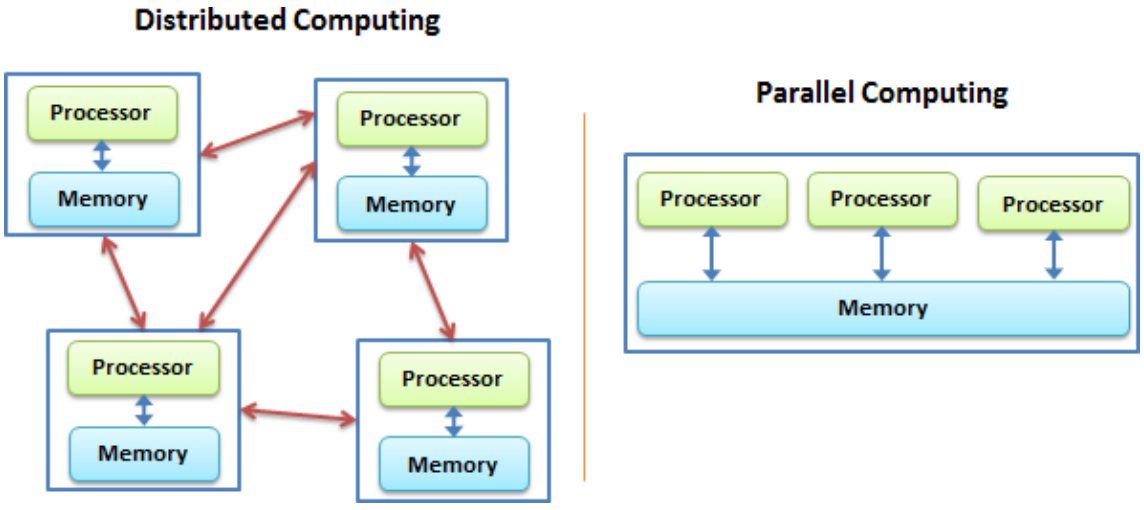


Figure 1.3: Distributed and Parallel Computing

Source: http://lycog.com/distributed-systems/parallel-and-distributed-computing/

In distributed computing, information is exchanged between the machines using the network bandwidth, which is often very limited. So, communication is one of the

scarcest resources in distributed computing [Li, 2017]. Most of the distributed methods rely on exchanging the gradients of loss function after every iteration [Zhang et al., 2015, Huo and Huang, 2016, Dean et al., 2012]. Hence, they tend to become expensive in terms of communication cost. Also, many of the distributed algorithms perform information exchange in a synchronized manner [Sra, 2012, Ho et al., 2013, Mairal, 2015]. For such methods, the slower machines become the bottleneck for the whole system, as faster machines has to wait for the slower machines to finish their computation. Hence, there is a need for totally distributed algorithms which can overcome the synchronization problem as well as minimize the total cost of communication between the machines.

1.2 Thesis Contributions

In this thesis, we take into account the above-mentioned problems of large-scale machine learning and propose algorithms in each of these domains which try to overcome the challenges addressed. We will discuss about the specific contributions corresponding to each of the problem in the following sections:

1.2.1 Large-scale Multiclass Classification

In the domain of large-scale multiclass classification we have following contributions:

1. First, we propose an algorithm to reduce multiclass classification problem to an equivalent binary classification problem. The reduced binary problem consists of similar number of positive and negative instances. Hence, it overcomes the class imbalance problem inherent in multiclass classification collection.

2. We further extend the algorithm, and introduce a double sampling strategy during training phase and an efficient candidate pre-selection approach during prediction phase. These modifications help to further improve the computational complexity of the model significantly. So, this makes the algorithm particularly attractive for extreme classification involving huge number of classes.

3. We derive generalization bounds for the proposed algorithm using local fractional Rademacher complexity, taking into account the inter-dependency between examples in the binary reduced dataset.

4. We empirically validate the effectiveness of the proposed algorithms in large text classification collections taking into account upto 100,000 categories.

1.2.2 Distributed Computing

In this front, we have following contributions:

1. We present a totally asynchronous distributed framework for distributed computing. In the proposed framework, data is partitioned and dispatched across several machines. Each machine performs gradient iterations in parallel over the local sub-part of data and updates local parameter simultaneously. Only after completing one complete pass over the local sub-part, the machines send the updated parameter to the master machine. The master machine integrates all received parameters and broadcasts it back to the worker machines, which use the updated parameter for a new pass over the data. In this way, we overcome the synchronization bottleneck as well as the communication cost is reduced since, we only exchange locally updated parameters as opposed to frequent exchange of gradients. Additionally, We present the convergence results for the proposed framework.

2. We empirically present the effectiveness of the proposed framework for binary classication and distributed matrix factorization.

1.3 Thesis Organization

We divide the whole thesis in two main parts:

I Reduction of Multi-class Classification to Binary Classification: Containing Chapters 2, 3 .

II Asynchronous Framework for Distributed Machine Learning: Containing Chapters 4, 5 and 6.

In Part I we discuss in detail the reduction of multiclass classification problems to binary classification. To begin with, we introduce Multiclass classification in chapter 2. We mainly discuss the challenges of performing large-scale classification and present

the state-or-art techniques. Then we discuss the ranking loss for multiclass classification and show the equivalence of this loss function to binary classification based loss. This gives main basis for our reduction algorithm. In Chapter 3, we formally present our two reduction algorithms. We also derive a generalization bound based on fractional Rademacher Complexity for the inter-dependent data. At the end, we present our empirical results in very large text classification collections of size up to 100000 classes.

In Part II, we explore the distributed techniques for handling large scale data for Machine Learning. In this domain, we propose an asynchronous framework for performing distributed machine learning. We show the application of the proposed framework in two popular domains: distributed matrix factorization in Recommender Systems and distributed binary classification. We begin this section with Chapter 4, where we introduce the distributed machine learning and discuss various research works in the two applications that we consider. Then we present our first application, distributed matrix factorization in Chapter 5. In this chapter, we present the algorithm AsyDM which is an asynchronous algorithm for performing matrix factorization for larges scale Recommender Systems. In Chapter 6, we present another application of the asynchronous algorithm for performing binary classification in large scale binary classification datasets. Theoretical and empirical analysis as well as the comparison with popular state-of-art approaches are discusses in each section.

I Reduction of Multiclass to Binary Classification

2 MULTICLASS CLASSIFICATION

2.1 INTRODUCTION

Supervised learning refers to problems which comprise examples of the input vectors along with their corresponding target vectors [Bishop, 2006]. For such problems, the strategy is to learn a prediction function from the training data. The learned function is later used for prediction on unseen data. Problems where we aim to assign each input vector to one of finite set of categories are referred to as classification problems. Some of the popular examples of classification problems are: digit recognition, spam detection, document classification, image classification etc. Another kind of problems where the output consists of one or more continuous variables, are known as regression problems. Some examples of regression problems are: credit score prediction, house price prediction, stock price prediction over time, user rating prediction in Recommender Systems etc.

Many supervised learning techniques aim at performing binary classification, where the number of possible classes is two. However, many real world classification problems involve the classification of an example to one or more classes from a predefined set of classes. Such problems are known as Multiclass classification problems. More precisely, problems where an example can belong to more than one classes, are referred as multilabel multiclass classification or often as multilabel classification. Whereas, the simplest case of problems where each example belongs to only one class from a finite set of classes, are known as monolabel multiclass classification or commonly as multiclass classification. Throughout this part, we will consider the case of monolabel multiclass classification and we will use the term multiclass classification to denote it.

Multiclass classification is a popular area of research in ML. Some of the popular examples of multiclass classification are text classification [Joshi et al., 2017, Joshi et al., 2015b, Yen et al., 2016], image categorization [Deng et al., 2010, Perronnin et al.,

2010, Cinbis et al., 2012], face recognition [Guo et al., 2016, Parkhi et al., 2015] and video annotation [Abu-El-Haija et al., 2016, Vondrick et al., 2013]. With the explosion in generation of data from different sources, modern multiclass classification problems involve very large size of datasets as well as very large number of classes and size of feature vectors. For example: in the case of text classification (as in LSHTC [1] and BioASQ [2] challenges) and image classification [Deng et al., 2010] the number of classes and size of feature space can be several thousands or even upto the order of millions. This increasing size of multiclass classification problems causes serious problems to the traditional multiclass classification approaches. We will explore those challenges in more detail in upcoming sections.

2.2 Related Work

Large-scale multiclass classification, which has evolved as a popular branch of machine learning, considers problems involving extremely large number of classes. Hence it is also referred as extreme classification. A number of prior works have addressed different aspects of this problem, which we are going to review in this section. Different approaches of multiclass classification can be categorized in the following categories.

2.2.1 Extensible Algorithms

Multiclass classification problem can be solved by extending the popular binary classification algorithms such as SVM, neural networks, decision trees, k-nearest neighbors, Naive Bayes etc [Aly, 2005].

- Neural Networks: Multilayer Feedforward Neural Networks can be naturally extended to handle multiclass scenario by using K output binary neurons for each of the class in multiclass setting. Different ways of choosing output codewords are proposed [Dietterich and Bakiri, 1995] such as: One-per-class coding, Distributed output coding etc.

- Decision Trees: Decision trees are widely used and powerful algorithm mainly for binary classification. The algorithm makes a tree shaped structure where the nodes are the features and the leafs represent the class labels. A new examples

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

is classified to one of the classes by following the route from root node to the leaf node. At each node a test is performed for the features of the example. This algorithm can be naturally extended to multiclass scenario by considering K leafs corresponding to each class label.

- Support Vector Machines (SVM): SVM's are considered as one of the most popular and robust algorithms in ML. The main idea of algorithm lies in maximizing the margin of the separating hyperplane. In normal setting, they are devised to handle binary classification. However, many extensions [Allwein et al., 2000, Dietterich and Bakiri, 1995, Friedman, 1996, Hsu and Lin, 2002] have been proposed to adapt it to handle multiclass scenario. In these extensions, the optimization problem is modified with additional parameters or constraints.

The extensible algorithms do not scale well to handle large-scale problems. In most of the cases, the complexity of algorithm becomes intractable. Because of these limitations, they are rarely used for extreme classification.

2.2.2 Transformation to Binary (Binarization)

Many research work have been proposed in last years for binary clasification, such as margin based classifiers, decision trees and ensembles [Bishop, 2006]. Some of the techniques can be naturally extended to handle multiclass problems (e.g. decision trees). Whereas other powerful and popular techniques, for e.g. Support Vector Machines (SVM) [Cortes and Vapnik, 1995] cannot be easily adapted to multiclass scenario. So, it is a common practice to decompose the multiclass problem to simple binary classification problems, the process is commonly referred to as binarization [Rocha and Goldenstein, 2014]. Binarization involves mapping one multiclass problem into several binary problems (divide and conquer), solve the individual problems using traditional binary learners (base learners) and finally combine their individual outcomes to derive multiclass prediction [Garcia-Pedrajas and Ortiz-Boyer, 2006]. Broadly these approaches can be classified to three main categories: Ove-Vs-One(OVO), One-Vs-All (OVA) and Error Correcting Output Codes (ECOC).

In OVO, a binary problem is created for each pair of classes of the initial problem. So, this leads to $K(K-1)/2$ binary problems, and as many binary classifiers. Figure 2.1 depicts OVO approach with separating hyperplanes of individual SVM classifiers for three classes. The predicted class for a new instance is the one that receives majority

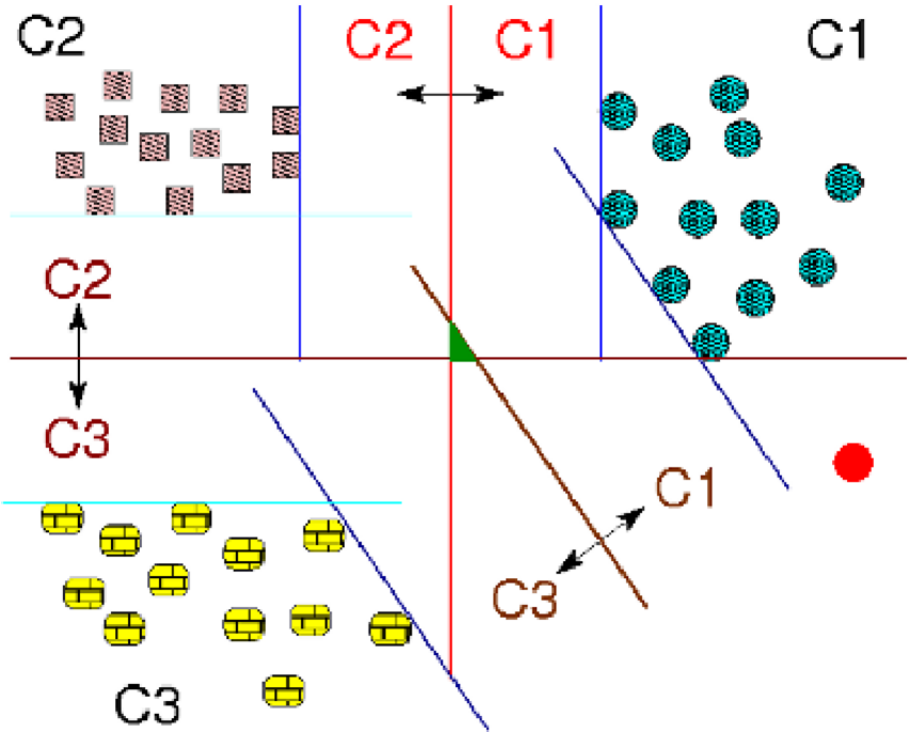


Figure 2.1: One-Vs-One Multiclass Classification [Fleury et al., 2013]

of the votes. This approach has a quadratic complexity in the number of classes, hence they cannot be used for extreme classification as they do not pass the scale for very large number of classes. Hence the approaches better suited for extreme classification setting are OVA and ECOC.

In OVA [Lorena et al., 2008] K binary problems are created, in each of them one class is seen as the positive class and the others as negative class. Given real-valued predictors $g_1, \ldots, g_K$, the predicted class for an instance x is given by $\arg\max_y g_y(x)$ i.e. the class with highest score by the predictor. This approach is shown in Figure 2.2. This approach has a long history, where the early work dates back to [Schiolkopf et al., 1995]. However, in the popular work of [Rifkin and Klautau, 2004], this approach was further revived and their empirical results showed its superiority over OVO, ECOC and M-SVM approaches. Later [Fan et al., 2008a] provided an easy and scalable implementation of the algorithm in their Liblinear package, which is widely used for experiments. The complexity of OVA is O(K X d), where K and d are number of classes and feature dimension respectively. Even though this complexity is better as compared to OVO, it is still high for extreme classification setting. Additionally, this approach is highly affected by class imbalance problem inherent in extreme classification.

In the ECOC-based approach [Dietterich and Bakiri, 1995], each of the L classes are represented with a binary code c_k, also known as the codeword. This gives rise

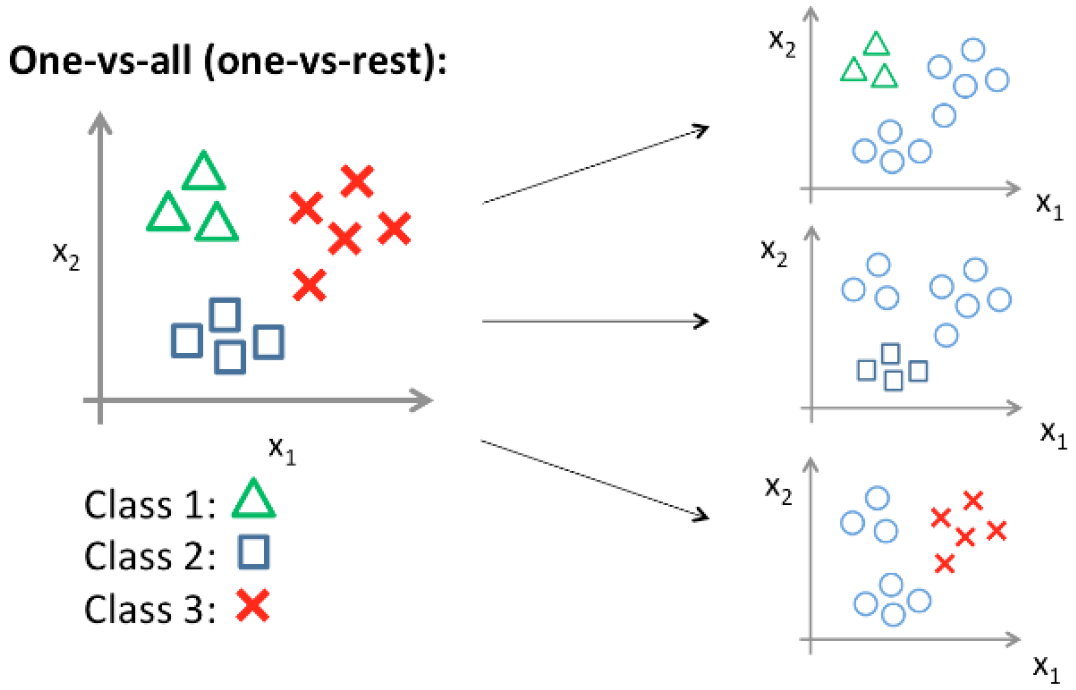


Figure 2.2: One-Vs-All Multiclass Classification

Source: https://houxianxu.github.io/2015/04/23/logistic-softmax-regression/

to a coding matrix M of size L × c_k. During training, one binary classifier is learned for each column to separate the positive and negative codes. Hence, there are as many binary classifiers learned as the number of columns in the code matrix. For a new test example, x, a codeword is assigned by evaluating with each of the learned classifiers (h). At prediction time, inference is performed by selecting the class that minimizes the distance between its code and the predicted code. The most popular distance measures are Hamming or Euclidean distance. This procedure is depicted in Figure 2.3. As shown in the figure, the correct predicted class for the new example x is C_3, which is the closest to the input test example in terms of both Hamming and Euclidean distance measures. Methods to speed up prediction or training with ECOC have recently been proposed: for example, only a subset of the classifiers may be used at inference time without loss of accuracy [Park and Fürnkranz, 2012]; in another direction, a Naive Bayes approach that only requires a single pass over the data for training has proved effective [Park and Fürnkranz, 2014]. However, the key challenge for these methods is to accurately choose the coding matrix.

Usually, a class binarization task involves creating, learning and combining several binary base learners. So, in large-scale setting standard binarization approaches such

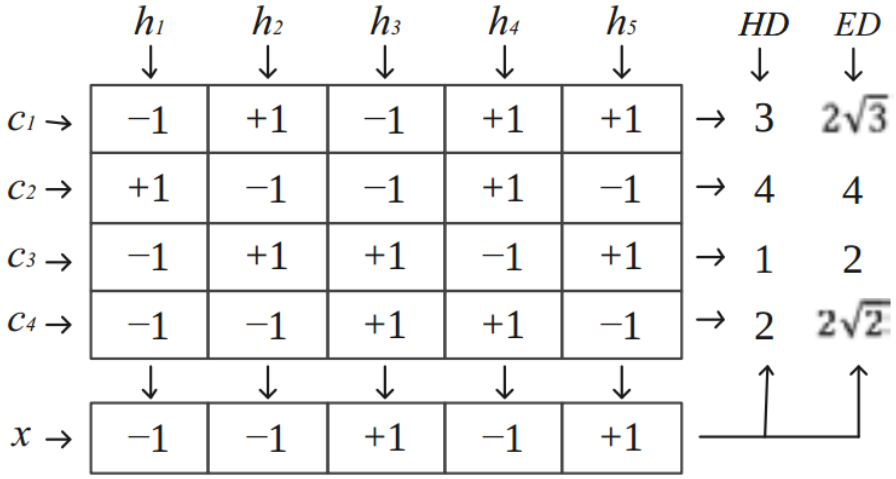


Figure 2.3: ECOC Multiclass Classification

Source: http://www.islab.ece.ntua.gr/

as OVA and OVO become computationally intractable for training and prediction both because they require large number of models. Also, they suffer from the popular problem of class imbalance inherent in large-scale multiclass problems.

2.2.3 Embedding Based approaches

Extreme multiclass/multilabel problems often involve learning with training examples, features and labels upto the order of millions. Hence, a natural way to handle such large number of class labels is to project the label vectors into a low dimensional space, with the assumption that the label matrix is low-rank. Different embedding based approaches mostly differ in the way the label matrix is compressed to low-rank and then decompressed back to original space. Different compression and decompression techniques are employed such as compressed sensing [Hsu et al., 2009, Kapoor et al., 2012], Bloom Filters [Cisse et al., 2013], SVD [Tai and Lin, 2012], output codes [Zhang and Schneider, 2011] etc. One state-of-art method LEML [Yu et al., 2014a] uses the Empirical Risk Minimization (ERM) framework while using a regularized least-square objective. In another recently proposed method SLEEC (Sparse Local Embedding for Extreme Classification) [Bhatia et al., 2015], the data is first clustered into smaller regions. It then performs local embeddings of label vectors using K-nearest neighbour classifier. The main advantages of embedding based methods include their simplicity, ease of implementation, strong theoretical foundations, ability

to handle label correlations (for multilabel scenario) and their ability to be extended in online or incremental scenarios [Bhatia et al., 2015]. Moreover, their use has been extended beyond classification, and been successfully used in ranking problems as demonstrated in WSABIE system [Weston et al., 2011a]. Hence, they are quite popular approach especially for extreme multilabel classification [Hsu et al., 2009, Tai and Lin, 2012, Balasubramanian and Lebanon, 2012, Bi and Kwok, 2013, Chen and Lin, 2012, Ferng and Lin, 2011]. However, the main downside of these approaches include their slow training and prediction times even with the use of considerably low embedding dimension. Also, the critical assumption that the training matrix is low rank, is violated in almost all real world applications. Hence, these approaches often suffer from lower accuracy in such applications.

2.2.4 TREE-BASED APPROACHES

Methods using tree-based classifiers have gained popularity in recent times. These methods rely on binary tree structures where each leaf corresponds to a class and inference is performed by traversing the tree from top to bottom, a binary classifier being used at each node to determine the child node to develop. These methods have logarithmic time complexity. In an earlier work, FilterTree [Beygelzimer et al., 2009b] presents a robust tree-based method for multiclass classification. However, the problem with this approach was the choice of partition. In most cases, the success of this method was related with choice of partition. Partition finding problem was also addressed in conditional probability tree [Beygelzimer et al., 2009a], however the use of conditional probability violates the logarithmic time operation. Later, [Bengio et al., 2010] used recursive spectral clustering on a confusion graph to address the partitioning problem. However, this makes the problem O(k) or even worse during training, making it intractable in extreme classification scenario. In another work, [Weston et al., 2013] used k-means hierarchial clustering to recover partition of the label sets, focusing their work primarily for multilabel rather than multiclass problems. [Choromanska et al., 2013] proposed an efficient method for extreme multiclass classification, using decision trees with an online learning algorithm. FastXML [Prabhu and Varma, 2014] is another popular method useful for both multiclass and multilabel scenario which optimizes an nDCG based ranking loss function. This method employs the partitioning of feature space instead of the label space using the observation that only small number of labels are active in each region of feature space. An extension

of this method is proposed as PfastReXML [Jain et al., 2016], which uses propensity scores to improve on tail label prediction. Also, another algorithm Log-time Log-space (LTLS) [Jasinska and Karampatziakis, 2016] claims to perform extreme classification in logarithmic time and space by embedding large classification problems into simple prediction problems and using dynamic programming for inference. Their empirical results show significant improvement in time and space usage. However their poor classification accuracy, especially for large-class cases is not justified. Another notable work using decision trees is Recall Tree [Daume III et al., 2016], which uses a binary tree to map an example to an small subset of candidate labels and uses a more tractable one-against-all classifier for prediction.

The main advantage of using the tree-based methods is to make the training, and/or prediction time logarithmic in number of classes, hence making it more attactive for extreme classification. However, its still a challenging task to find a balanced tree structure which can partition the class labels. Even though the above-mentioned methods have proposed several heuristics to address this problem and are able to reduce the complexity of the model to logarithmic time, empirically they do not show good predictive performance especially in large-class scenario. This is mainly caused by the fact that the prediction error made at the top the tree structure cannot be corrected at lower levels, also known as the cascading effect.

2.2.5 MISCELLANEOUS

In the proposed method also, we design joint features between classes and examples allowing to learn a single parameter vector for the whole problem. Another similar piece of work in [Weston et al., 2011b] learns representation for each classes. This approach learns a projection of examples and classes into a low dimensional space, hence reducing both training and inference time. However, in contrast to our approach, this method learns one parameter vector per class, while we use joint features of classes and examples allowing to reduce the number of vector parameters to one. In another line of work, [Titsias, 2016] proposed an approximation for the softmax layer for calculating probabilities of multiclass classification. Specifically, the author proposes a new bound on the softmax which factorizes the calculations in a product and thus avoids to evaluate the normalization constant which can become intractable for very large number of classes. In the case of extreme multiclass classification a doubly stochastic approximation scheme is used, without providing any theoretical guarantees, where

one randomly selects a number of candidate classes while performing gradient descent. In our proposed method we also introduce a double stochastic procedure as an unbiased empirical risk minimization of the original expected loss.

Another recent algorithm to address extreme classification is PD-sparse method [Yen et al., 2016], where authors use sparsity for high-dimensional datasets. However, sparsity is not guaranteed to generate small size models without hurting model accuracy.

2.3 CHALLENGES IN MULTICLASS CLASSIFICATION

Large-scale multiclass classification or extreme classification has emerged as a popular research problem in machine learning research community. Hence, it has been the central topic in many top conferences and workshops in last few years such as Large Scale Hierarchical Text Classification Challenge [3] whose results were presented in ECML, ECIR, WSDM and ICML, the Extreme classification workshop in NIPS 2013, 2015, 2016 and 2017 [4]. These popular workshops/conferences alongwith many others helped to bring together researchers from all over the globe to discuss the challenges in extreme classification. Additionally, this has given rise to many useful benchmark datasets for the task and popular algorithms to tackle the problem. Here, we will summarize the main challenges in extreme classification.

1. Class imbalance/ Data Scarcity problem: In large-scale classification collections, it is observed that the average category size is related to the number of categories taken into consideration. This pattern is evident from the Figure 2.4. This figure shows that the average category size decreases as the number of categories increases. This pattern is more prominent in large-scale collections. It has also been observed in large-scale collections that majority of classes have very few representative examples, whereas a few classes contain majority of examples. This behavior is termed as power law distribution or long-tailed distribution [Babbar et al., 2014a]. Figure 2.5 shows such long tailed distribution for one of the popular text classification datasets (DMOZ). Such power law distribution of category size implies a sever class imbalnce in such collections which makes learning a difficult task. Hence, class imbalance problem has to be taken

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

[4] http://manikvarma.org/events/XC13/index.html

into account when learning a model for extreme classification.

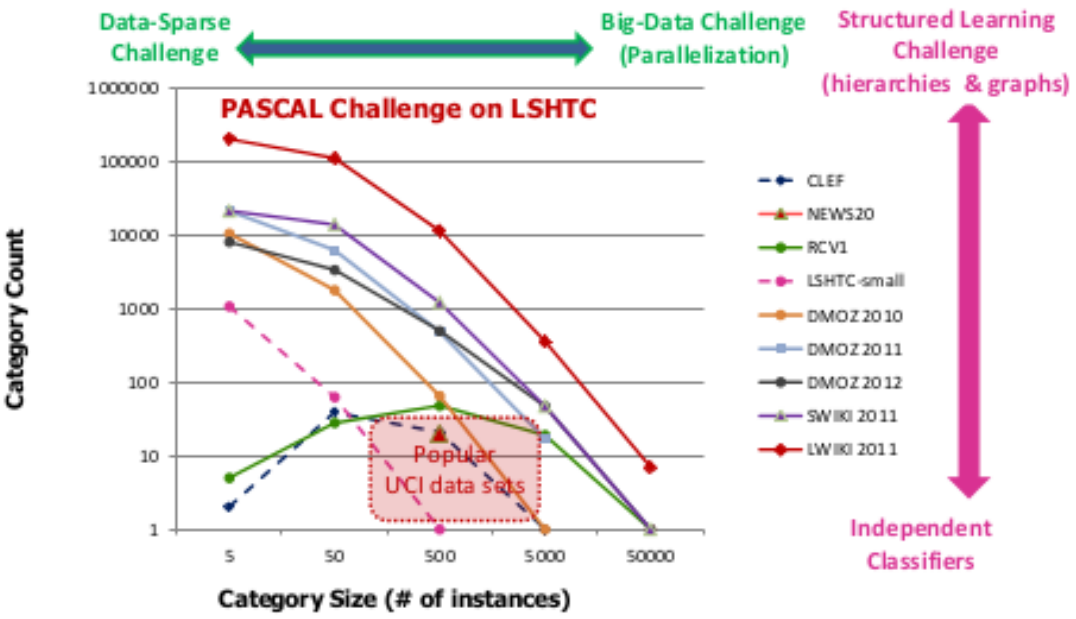


Figure 2.4: Research Challenges in Extreme Classification

Source: Yimming Yang Talk at WSC Workshop WSDM 2014

2. Dimensionality: In extreme classification scenarios, one has to deal with datasets of very large size and dimensionality. For example: in the case of Wikipedia large dataset with 325K categories, the feature dimension is more than a million. Similarly, for DMOZ dataset the feature size is more than half a million. Feature dimension directly relates with the size of the vocabulary to be used as well as the complexity of the learning model. There have been many approaches which tend to encode the problem in lower dimension using different feature embeddings. However, it is not a trivial task as it might directly affect the prediction performance of the model. Also, if the number of categories is huge, the number of examples in the dataset is also very large. Hence, the algorithms have to be adapted to handle such extreme classification setting.

3. Model complexity: As we already discussed, extreme classification involves problems of huge dimensionality and training data size. This significantly increases the memory and computational burden when learning a model. For example, we will show in experimental section that for a subset of WIKI-325K dataset containing 100K categories, One-Vs-All algorithm requires nearly 1 TB to store the parameters. Similarly it takes several days for learning the model. This a com-

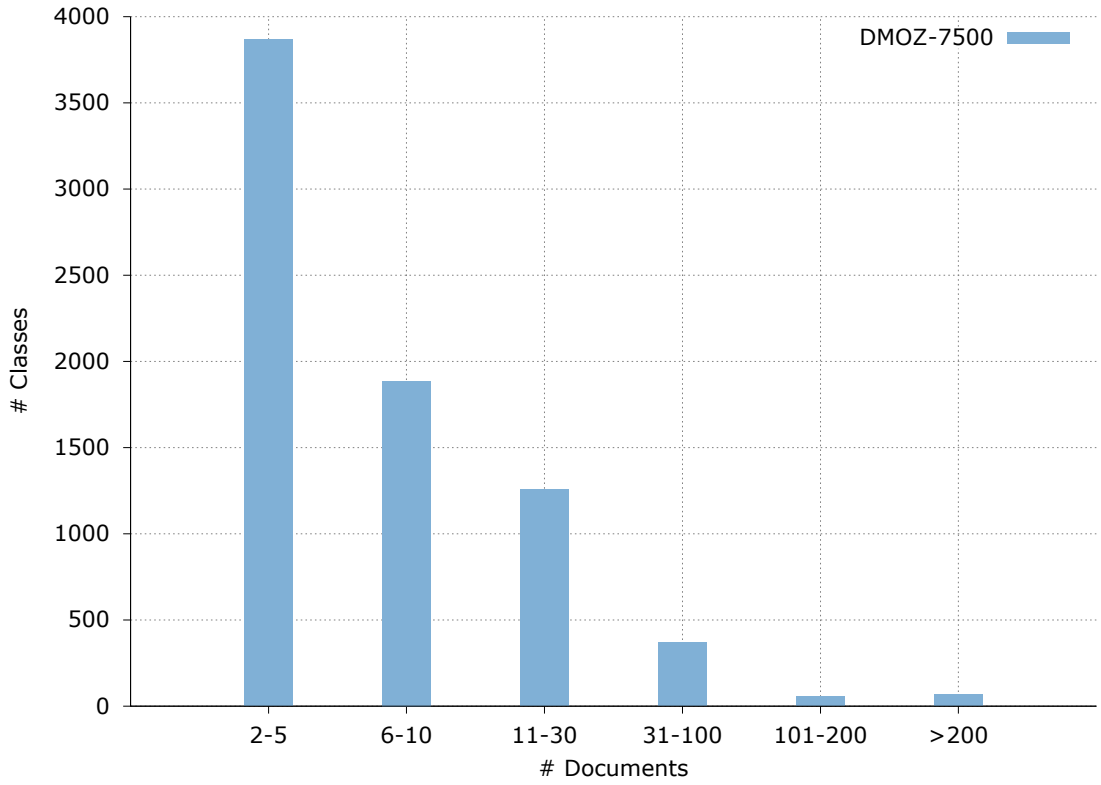


Figure 2.5: Long-tailed distribution on DMOZ dataset

mon problem for most of the approaches listed in the previous section. Hence, in extreme classification setting, it is critical to devise algorithms which have low computational complexity in terms of the time to learn a model as well as the total memory used to store the learned parameters.

2.4 Text Classification

In this section, we introduce text classification, which is a popular multiclass classification problem. This is also our application of choice to validate the proposed algorithms. Text classification is a supervised learning problem, where we are given the description $\mathbb{X} \in \mathbb{R}^d$ of text documents and a fixed set of classes $\mathbb{C} = \{c_1, c_2, ..., c_k\}$ [Manning et al., 2008]. In the learning phase, we are given a training set $\mathbb{D}$ with labels, and our task is to learn a predictor γ that maps documents to classes:

$$\gamma : \mathbb{X} \rightarrow \mathbb{C}$$

In most of the recent text classification problems, the document space $\mathbb{X}$ is high dimensional as well as the number of class labels. For example, in two popular doc-

ument collections DMOZ [5] and Wikipedia [6] they are up to the order of hundreds of thousands to millions.

2.4.1 TEXT PREPROCESSING

Before applying learning algorithms, the text collections are preprocessed. This helps to weed out unnecessary information from the text while keeping the information useful for better learning. Many of the popular tools in Natural Language Processing (NLP) such as NLTK [7], Scikit-learn [8] provide in-built functionality for preprocessing. Some of the commonly used preprocessing steps are listed below [Manning et al., 2008]:

- Stop-word removal: Stop words refer to commonly occurring words in a language. Such words do not contribute to distinguish one text document from the other, hence needs to be removed. Popular text processing toolkits have predefined list of stop words.
- Stemming and Leammatization: Documents use different form of words because of grammatical reasons. For e.g. organize, organizes, organized or organizing. Similarly many words might be derivationally related words with similar meaning such as democracy, democratic and democratization. Both stemming and lemmatization are used to reduce such multiple derivationally related words to their base form. Stemming is a crude method that chops off the derivational affixes from words. Whereas lemmatization performs it in a more proper way by using the vocabulary and morphological analysis of words.

2.4.2 FEATURE REPRESENTATION:

After preprocessing, next step is to represent each text document with a set of features. Individual components of text document are the words. Hence, many feature representations are proposed to represent the words in the document. In this section, we will discuss two representations: Bag of words, which is the most popularly used

[5] http://www.dmoz.org/
[6] www.wikipedia.org
[7] http://www.nltk.org
[8] http://scikit-learn.org

representation in text applications and dyadic representation, which we will use in our proposed algorithms.

- Bag of Words (BOW) Representation: This model is also known as Vector Space Model (VSM) [Raghavan and Wong, 1986, Van Rijsbergen, 1979, Hu, 2011]. In this model, each document is represented as a vector of length same as the total distinct terms in the corpus, also known as the dictionary. Each distinct term present in a document is given a weight and represented as the document vector. Hence, for each term in the dictionary not present in the document as given a weight as zero. There are many ways to assign the weights for each terms in the document. One simple way is to use the frequency of occurrence of each distinct term in the document as the feature value. It is also commonly known as Term Frequency (TF). Such representation can be used to assess the similarity between two document vectors. However, using raw term frequency suffers from a critical issue [Manning et al., 2008] that it assumes each term to be equally important. However, certain terms may be used too frequently in all the documents and hence are very less useful to discriminate between two documents. Hence, to mitigate the effect of these popularly occurring words, another representation is proposed which tries to scale down the frequency of the terms with document frequency (number of documents in the collection containing that term). This representation is known as Inverse Document Frequency (IDF) and is given as [Manning et al., 2008]:

$$idf_t = \log \frac{N}{df_t}$$

Where N is the total number of documents in the collection and df_t represents the document frequency for the term t. IDF of a rare term is high and frequently appearing terms is low. The most popular bag of words representation, known as *tf-idf* weighting, which is calculated as the multiplication of *tf* and *idf* values. The tf-idf representation of a term t in a document d is given as:

$$\textit{tf-idf} = tf_{t,d} \times idf_t$$

So, the $\textit{tf-idf}_{t,d}$ term assigned for each term in document d is [Manning et al., 2008]:

1. highest when t occurs many times in a few documents.
2. lower when term occurs fewer times in a document; or occurs in many documents.
3. lowest when the term occurs in almost all the documents.

- Dyadic (or Joint) Representation: Another popular feature representation in information retrieval is the one commonly used in ranking of documents according to their relevance to a query. This field of research is known as learning to rank [Liu et al., 2009, Qin et al., 2010, Liu, 2011]. Here, the query-document pair is represented by a multi-dimensional feature vector. The small set of features try to encode the relevance of the document with respect to the query. For example one simple example of a feature can be the total number of terms present in both the query feature vector and the document vector. Similarly, a number of classical information retrieval features can be manufactures by considering this relevance. Moreover, this joint representation can be used to extract specialized similarity features such as BM25, LMIR [Qin et al., 2010]. All the documents belonging to one class can be considered as a single large document. Hence, the query-document joint feature representation can be extended to the joint representation of a document and the collection of documents belonging to one class. In our work, we use this joint feature representation to classify documents to one of the classes. We have exchangeably used the terms joint or dyadic or similarity-based feature representation throughout this thesis to denote it. In the experiments section we will present the features we have used in the experiments.

2.4.3 Evaluation Measures

The correctness of a classification task can be evaluated using four aspects: examples which are predicted as positive and the true class is also positive (true positives), examples which are predicted as negative and the true class is also negative (true negatives), examples predicted as positive however the true class is negative (false positives) and examples which are predicted to belong to negative class however the true class is positive (false negatives). All these attributes are summed up as a confusion matrix shown in Table 2.1.

We begin with presenting evaluation measure for a binary classification task. These can be extended for multiclass classification case as well. Some of the commonly used evaluation measures are summarized below:

1. **Accuracy** is measured as the fraction of predictions that are correct. Mathematically, it is represented as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Precision**: measures the fraction of correctly identified positive examples out of the total positive predicted as positive. Mathematically, it is represented as:

$$Precision = \frac{TP}{TP + FP}$$

3. **Recall**: measures the total fraction of positive examples identified out of the total positive examples. Mathematically, it is represented as:

$$Recall = \frac{TP}{TP + FN}$$

4. **F-Measure**: (also known as F_1 measure)is the measure that combines both precision and recall as their harmonic mean. Mathematically, it is represented as:

$$F - Measure = 2.\frac{Precision * Recall}{Precision + Recall}$$

True class	Predicted as positive	Predicted as negative
Positive	true positive (tp)	false negative (fn)
Negative	false positive (fp)	true negative (tn)

Table 2.1: Confusion matrix

Table 2.2 presents the popularly used evaluation measures for multiclass classification [Sokolova and Lapalme, 2009]. For each class C_i, the assessment is denoted as tp_i, fn_i, tn_i and fp_i. Here, one thing to note is that most of the measures are assessed in two ways: average of the same measures calculated individually for each class (macro-averaging denoted with index M), or the computing the cumulative sum

of tp, fp, tn, fn and then calculating the measures (micro averaging denoted as μ subscripts). Macro averaging treats each class equally whereas micro averaging favours the bigger classes. Hence, macro averaging is considered as a superior evaluation measure as compared to average accuracy and micro-averaging in multiclass classification where the class imbalance problem is inherent.

Measure	Formula	Description
Average Accuracy	$\frac{\sum_{i=1}^{l} \frac{tp_i+tn_i}{tp_i+fn_i+fp_i+tn_i}}{l}$	Measures per-class effectiveness of a classifier
Error Rate	$\frac{\sum_{i=1}^{l} \frac{fp_i+fn_i}{tp_i+fn_i+fp_i+tn_i}}{l}$	Measures per-class classification error
$Precision_\mu$	$\frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l}(tp_i+fp_i)}$	Agreement of the data class labels with those of a classifier if calculated from sums of per-text decisions
$Recall_\mu$	$\frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l}(tp_i+fn_i)}$	Effectiveness of a classifier to identify class labels if calculated from sums of per-text decisions
$Fscore_\mu$	$\frac{(\beta^2+1)Precision_\mu Recall_\mu}{\beta^2 Precision_\mu+Recall_\mu}$	Harmonic mean of micro averaged precision and micro averaged recall
$Precision_M$	$\frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i+fp_i}}{l}$	An average per-class agreement of the data class labels with those of the classifiers
$Recall_M$	$\frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i+fn_i}}{l}$	An average per-class effectiveness of a classifier to identify class labels
$Fscore_M$	$\frac{(\beta^2+1)Precision_M Recall_M}{\beta^2 Precision_M+Recall_M}$	Harmonic mean of macro averaged precision and macro averaged recall

Table 2.2: Popular evaluation measures for multiclass classification

2.5 CLOSING REMARKS

In this chapter, we discussed multiclass classification in detail. We begin with the introduction to multiclass classification. In Section 2.2, we presented various state-of-the-art algorithms to solve multiclass classification problems. Then in section 2.3, we discussed various challenges associated with it. Later in section 2.4, we introduced a popular application involving multiclass classification, which is also the application of choice to validate our proposed methods. In that section, we discussed various steps of text classification such as text preprocessing, feature representations and the popular evaluation measures. Now, with enough background in multiclass classification and the text classification application, we will present our proposed multiclass to binary reduction technique in next chapter.

3 Reduction to Binary Classification

3.1 Ranking Loss for Multiclass Classification

We consider monolabel multiclass classification problems defined on a joint space $\mathbb{X}\times\mathbb{Y}$ where $\mathbb{X} \subseteq \mathbb{R}^d$ is the *input space* and $\mathbb{Y} = \{1, \ldots, K\}$ the *output space*, made of K class labels. Elements of $\mathbb{X} \times \mathbb{Y}$ are denoted as $\mathbf{x}^y = (x, y)$. Furthermore, we assume the training set $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m$ is made of i.i.d pairs distributed according to a fixed but unknown probability distribution $\mathbb{D}$, and we consider a class of functions $\mathbb{G} = \{g : \mathbb{X} \times \mathbb{Y} \to \mathbb{R}\}$ as our predictors. We define the instantaneous loss of $g \in \mathbb{G}$ on an example $\mathbf{x}^y$ as:

$$e(g, \mathbf{x}^y) = \frac{1}{K-1} \sum_{y' \in \mathbb{Y} \setminus \{y\}} 1_{g(\mathbf{x}^y) \leq g(\mathbf{x}^{y'})}, \tag{3.1}$$

where 1_π is the indicator function that is equal to 1 if the predicate π is true and 0 otherwise. Compared to the classical multiclass error:

$$e'(g, \mathbf{x}^y) = 1_{y \neq \text{argmax}_{y' \in \mathbb{Y}}\, g(\mathbf{x}^{y'})}, \tag{3.2}$$

the loss of (3.1) estimates the average number of classes, given any input data, that get a greater scoring by g than the correct class. The loss (3.1) is hence a *ranking* criterion, and the multiclass SVM of [Weston and Watkins, 1998] and AdaBoost.MR [Schapire and Singer, 1999] optimize convex surrogate functions of this loss. The multiclass classification problem we are going to study is that of finding a function $g \in \mathbb{G}$ using the labeled training set $\mathbb{S}$ with small generalization error $L(g)$:

$$L(g) = \mathbb{E}_{\mathbf{x}^y \sim \mathbb{D}}[e(g, \mathbf{x}^y)]. \tag{3.3}$$

Accordingly, the empirical error of $g \in \mathbb{G}$ over $\mathbb{S}$ is

$$\hat{L}_m(g, \mathbb{S}) = \frac{1}{m} \sum_{i=1}^{m} e(g, \mathbf{x}_i^{y_i}) \tag{3.4a}$$

$$= \frac{1}{m(K-1)} \sum_{i=1}^{m} \sum_{y' \in \mathbb{Y} \setminus \{y_i\}} 1_{g(\mathbf{x}_i^{y}) \leq g(\mathbf{x}_i^{y'})} \tag{3.4b}$$

We further work out the empirical loss of Equation (3.4) in order to *i*) have it ressemble a more usual binary classification loss with, in particular, a single sum running over only one index, *ii*) make apparent the need of dealing with non-i.i.d. random variables and *iii*) after a theoretical introduction, set the stage for our practical binary reduction approach.

A first step to reshape the empirical loss is to see that the instantaneous loss (3.1) can be rewritten as

$$e(g, \mathbf{x}^y) = \frac{1}{K-1} \sum_{y' \in \mathbb{Y} \setminus \{y\}} 1_{\tilde{y} h(\mathbf{x}^y, \mathbf{x}^{y'}) \leq 0},$$

where h is defined as $h(\mathbf{x}^y, \mathbf{x}^{y'}) = g(\mathbf{x}^y) - g(\mathbf{x}^{y'})$. This bears strong resemblance with a binary-classification-loss-based risk, a resemblance that can be strengthened by introducing the transformed set $T(\mathbb{S})$ of size $n = m(K-1)$ defined as

$$T(\mathbb{S}) = \left\{ \left(\boldsymbol{Z}_j, \tilde{y}_j \right) : j = 1, \ldots, n \right\}, \tag{3.5}$$

where each $\boldsymbol{Z}_j$ is one of the pairs $(\mathbf{x}_i^y, \mathbf{x}_i^{y'})$, and $\tilde{y}_j = 1$ if the first observation in $\boldsymbol{Z}_j$ is constituted by an example $\mathbf{x}_i$ and its true class in $\mathbb{S}$ (i.e. $y = y_i$) and the second observation is constituted by the same example and any other of the $K-1$ classes; and $\tilde{y}_j = -1$ otherwise (i.e. if the order is reverse). This allows us to rewrite the empirical loss of (3.4b) as:

$$L_n^T(h, T(\mathbb{S})) = \frac{1}{n} \sum_{j=1}^{n} 1_{\tilde{y}_j h(\boldsymbol{Z}_j) \leq 0}. \tag{3.6}$$

With these definitions at hand, it is clear that the selection of a hypothesis in $\mathbb{G}$ minimizing the empirical risk of (3.4) over the training set $\mathbb{S}$, is equivalent to the search of a hypothesis in $\mathbb{H} = \{h : h(\mathbf{x}^y, \mathbf{x}^{y'}) = g(\mathbf{x}^y) - g(\mathbf{x}^{y'}), g \in \mathbb{G}\}$ minimizing the empirical risk of (3.6) over $T(\mathbb{S})$. However, even if the examples in $\mathbb{S}$ are i.i.d., the examples in $T(\mathbb{S})$ are no longer independent since the same observations $\mathbf{x}^y \in \mathbb{S}$ are involved in different pairs of $T(\mathbb{S})$. Thus, in order to obtain generalization error bounds $L_n^T(h, T(\mathbb{S}))$ we need to address the issue of learning with interdependent data. We will discuss this issue in detail in next sections.

3.2 Multiclass to Binary Reduction

3.2.1 Reduction Strategy

In section 3.1 , we derived an equivalence of multiclass classification problem to a binary problem. Hence, if we can represent each example of our dataset as dyadic pairs of ($\mathbf{x}$, y_i) for all i ∈ K, then based on that equivalence shown in Equation 3.6 we can transform the multiclass dataset to binary. Figure 3.1, depicts this transformation over a toy problem. More precisely, we consider the following transformation:

$$T(\mathbb{S}) = \left(\begin{cases} \left(\boldsymbol{Z}_j = \left(\mathbf{x}_i^k, \mathbf{x}_i^{y_i}\right) \quad , \tilde{y}_j = -1\right) & \text{if } k < y_i \\ \left(\boldsymbol{Z}_j = \left(\mathbf{x}_i^{y_i}, \mathbf{x}_i^k\right), \tilde{y}_j = +1\right) & \text{elsewhere} \end{cases} \right)_{j \doteq (i-1)(K-1)+k}, \tag{3.7}$$

for $j = (i-1)(K-1)+k$ with $i \in [m], k \in [K-1]$, thus T transforms a monolabel K class classification set $\mathbb{S}$ of m feature/label pairs into a set $T(\mathbb{S})$ of size $N = m(K-1)$. We consider the following class of functions

$$\begin{aligned} \mathbb{H} &= \{h : (\mathbb{X} \times \mathbb{Y})^2 \to \mathbb{R}; \\ h(\mathbf{x}^y, \mathbf{x}^{y'}) &= g(\mathbf{x}^y) - g(\mathbf{x}^{y'}), g \in \mathbb{G}, \end{aligned} \tag{3.8}$$

Here, one thing to note is that the label assignment in Equation 3.7 is done based on the new hypothesis function learned over the subtraction of dyadic representations as represented in 3.8. Hence, if the first representation in the subtraction pair corresponds to the true label, then the subtraction should be positive making the binary label as +1 and vice versa.

3.2.2 Reduction Example

Figure 3.1, presents a toy example of reduction of multiclass classification to binary classification. The example consists of an original multiclass dataset denoted as $\mathbb{S} = \{x_1, x_2, x_3, x_4\}$, which consists of 4 examples each of them belong to four classes; $\mathbb{Y} = \{y_1, y_2, y_3, y_4\}$ respectively. We apply our binary transformation function denoted as T to this multiclass dataset and obtain the binary reduced dataset denoted as $T(S)$. As can be seen in the figure, new examples in the transformed set are created by the subtraction of dyadic representations of each example with its true class and all other class labels in the output space following the transformation function in 3.7. We will

discuss about the feature representation in the next section. As we said before for m examples in the original set we create $m \times (K-1)$ examples in the transformed set. Hence, in the example with 4 training examples and 4 class labels we have 4 × (4 -1) = 12 examples in the transformed set. One notable property of this transformation is that the number of positive and negative examples in the transformed set are equivalently similar. This helps to solve the class imbalance problem of multiclass classification.

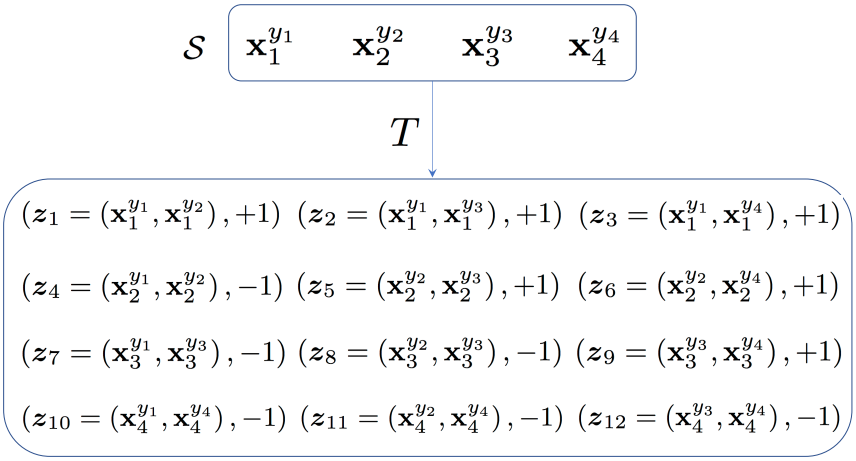


Figure 3.1: A toy example depicting the transformation T (Eq. 3.7) applied to a training set $\mathbb{S}$ of size $m = 4$ and $K = 4$.

3.2.3 Low-dimensional Feature Map

In multiclass classification, the output space is unstructured and the algorithms using the "trivial" feature map need a single parameter vector for each class. So, the parameter for such problems is in fact the concatenation of one parameter vector per class. However, our reduction technique relies on the dyadic (joint) representation of $\mathbf{x}^y$ of example and classes. This allows us to make use of a non-trivial feature representation $\phi(\mathbf{x}^y)$ by using a small number of adequately chosen similarity features between examples and classes. Typical low-dimensional features for text classification can be common terms between example and all examples in a class, similarity features etc (see Section 3.3.3.2). This joint feature space is independent of the number of classes and hence remains same for any number of classes. So, learning can be achieved by combining these features, using same parameter vector for all the classes. The use of such low-dimensional feature representation has a huge benefit in terms of memory usage. We will demonstrate this fact further in the Experiments section later.

3.3 Naive Reduction Algorithm (mRb)

3.3.1 Algorithm Description

Now, using the reduction strategy introduced in previous section, we present our first classification algorithm based on the reduction of multiclass to binary.

3.3.1.1 Reduction Phase

The first step of the classification algorithm is the binary reduction phase introduced in 3.2.1. Algorithm 1 outlines the reduction phase of the proposed algorithm. We use a low-dimensional dyadic feature representation rather than using the original feature space as suggested in 3.2.3. The dyadic representation consists of a small number of adequately chosen features. The output of the reduction phase is a binary transformed dataset.

Input: Labeled training set $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m$
Initialize
$T(\mathbb{S}) \leftarrow \emptyset$
for $i = 1..m$ **do**
  for $k = 1..K$ **do**
    if $k < y_i$ **then**
      $T(\mathbb{S}) \leftarrow T(\mathbb{S}) \cup \left(\boldsymbol{Z} = \left(\phi(\mathbf{x}^k), \phi(\mathbf{x}^{y_i})\right) \quad , \tilde{y} = -1\right)$
    end
    else
      $T(\mathbb{S}) \leftarrow T(\mathbb{S}) \cup \left(\boldsymbol{Z} = \left(\phi(\mathbf{x}^{y_i}), \phi(\mathbf{x}^k)\right) \quad , \tilde{y} = +1\right)$
    end
  end
end
return $T(\mathbb{S})$

Algorithm 1: Multi-class to Binary Reduction Phase

3.3.1.2 Learning Phase

Since the transformation gives us a binary dataset, we can now train a binary learner. Some of the popularly used binary learners [Bishop, 2006] are Logistic Regression,

SVM etc. The binary learner learns a weight vector, $\mathtt{W}$. These learned weights can be used to classify the future unseen instances.

3.3.1.3 Prediction Phase

Now, after training the binary learner, we use the learned model to classify the test instances. However, its not very trivial in our case, since the model is learned over binary dataset and the test instances are multiclass. So, the procedure we follow during prediction phase is depicted in Algorithm 2. For each test example, x', we make a dyadic representation with respect to all the class labels in $\mathbb{Y}$, and the one with the highest dot product with the weight vector is assigned as the predicted class for the test example.

Input: Unlabeled test set $\mathbb{T} = (\mathbf{x}_i)_{i=1}^{T}$
Learned feature weight vector $\mathtt{W}$
Initialize:
$P \leftarrow \emptyset$
forall $\mathbf{x} \in \mathbb{T}$ **do**
  $P \leftarrow P \cup \text{argmax}_{k \in \mathbb{K}} \langle \mathtt{W}, \phi(\mathbf{x}^k) \rangle$
end
return predicted classes P

Algorithm 2: Prediction with Binary Learned Model

3.3.2 Generalization Bound Analysis Using Fractional Rademacher Complexity

With these definitions at hand, it is clear that the selection of a hypothesis in $\mathbb{G}$ minimizing the empirical risk of (3.4) over the training set $\mathbb{S}$, is equivalent to the search of a hypothesis in $\mathbb{H} = \{h : h(\mathbf{x}^y, \mathbf{x}^{y'}) = g(\mathbf{x}^y) - g(\mathbf{x}^{y'}), g \in \mathbb{G}\}$ minimizing the empirical risk of (3.6) over $T(\mathbb{S})$. However, even if the examples in $\mathbb{S}$ are i.i.d., the examples in $T(\mathbb{S})$ are no longer independent since the same observations $\mathbf{x}^y \in \mathbb{S}$ are involved in different pairs of $T(\mathbb{S})$. Thus, in order to obtain generalization error bounds $L_n^T(h, T(\mathbb{S}))$ we need to address the issue of learning with interdependent data.

There exist several ways to tackle this problem among which two settings received particular attention in the literature. The first one deals with learning from mixing pro-

cesses, where the dependency between random variables decreases over time [Mohri and Rostamizadeh, 2009, Steinwart and Christmann, 2010]. The second direction, on which the present work is based on, is developed around the idea of graph coloring that divides a graph, representing the relations between random variables, into sets of independent random variables called proper cover of the graph [Janson, 2004].

A proper cover of $T(\mathbb{S})$ is constituted of $K-1$ disjoint sets $(C_k)_{k=1}^{K-1}$ each containing m independent examples. For all $k \in \{1,\dots,K-1\}$ it is defined as

$$C_k = \{(\boldsymbol{Z}_{k+j(K-1)}, \tilde{y}_{k+j(K-1)}); j \in \{0,\dots,m-1\}\}$$

Moreover, $(C_k, \alpha_k)_{k=1}^{K-1}$ is said to be a proper exact fractional cover of $T(\mathbb{S})$, if $(C_k)_{k=1}^{K-1}$ is a proper cover of $T(\mathbb{S})$ and if $\forall k, \alpha_k > 0$ and

$$\forall i \in \{1,\dots,n\}, \sum_{k=1} \alpha_k 1_{(\boldsymbol{Z}_i, \tilde{y}_i) \in C_k} = 1.$$

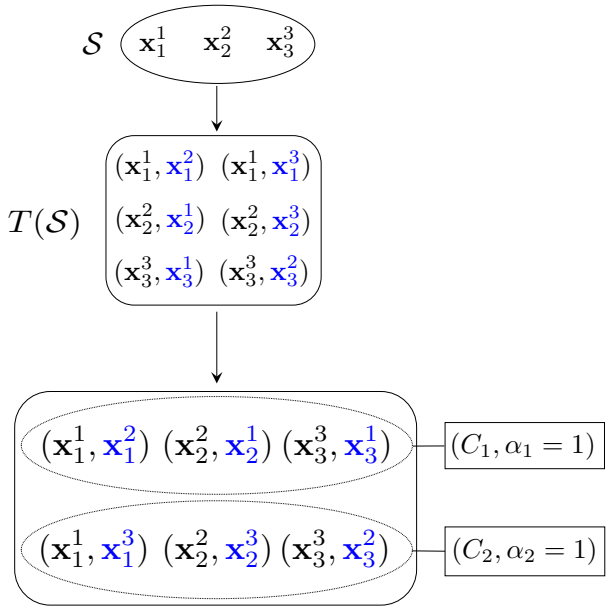


Figure 3.2: The proper exact fractional cover of the set $T(\mathbb{S})$ obtained after transformation of the training set $\mathbb{S} = \{\mathbf{x}_1^1, \mathbf{x}_2^2, \mathbf{x}_3^3\}$. For the sake of clarity, the class labels of pairs of examples are omitted. The fractional chromatic number of T is in this case $\chi_T^* = 2$.

The fractional chromatic number of T, denoted as χ_T^* is then the minimum sum of weights, or the minimum number of sets containing each independent random variables, which for the proposed transformation is equal to $K-1$. Figure 3.2 depicts the transformation and its associated proper exact fractional on a toy problem.

Using graph coloring arguments, [Janson, 2004] extended Hoeffding's inequality to sums of interdependent random variables and based on that result, different studies proposed new generalization error bounds for learning with interdependent data, thus proving the consistency of the ERM principle for this case [Usunier et al., 2006, Ralaivola et al., 2010]. Here we build on [Usunier et al., 2006] who proposed a generalization of [McDiarmid, 1989] concentration inequality to the case of interdependent random variables.

Our theoretical result is the following theorem which provides data-dependent bound on the generalization error of the multiclass classifier (Eq. 3.3). This result is at the basis of the algorithm for the binary classification of pairs of examples that we expose in the next section. We consider here kernel-based hypotheses with $\kappa : \mathbb{Z} \to \mathbb{R}$ a *positive semidefinite* (PSD) kernel and $\phi : \mathbb{X} \times \mathbb{Y} \to \mathbb{H}$ its associated feature mapping function, defined as:

$$\mathbb{G}_B = \{\mathbf{x}^y \in \mathbb{X} \times \mathbb{Y} \mapsto \langle \boldsymbol{w}, \phi(\mathbf{x}^y) \rangle \mid ||\boldsymbol{w}|| \leq B\} \tag{3.9}$$

where $\boldsymbol{w}$ is the weight vector defining the kernel-based hypotheses and $\langle \cdot, \cdot \rangle$ denotes the dot product. We further define the following associated function class:

$$\mathbb{H}_B = \{(\mathbf{x}^y, \mathbf{x}'^{y'}) \in \mathbb{Z} \mapsto g_w(\mathbf{x}^y) - g_w(\mathbf{x}'^{y'}) \mid g_w \in \mathbb{G}_B\}.$$

Theorem 3.1. *Let $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m \in (\mathbb{X} \times \mathbb{Y})^m$ be a dataset of m examples drawn i.i.d. according to a probability distribution $\mathbb{D}$ over $\mathbb{X} \times \mathbb{Y}$ and $T(\mathbb{S}) = ((\boldsymbol{Z}_i, \tilde{y}_i))_{i=1}^n \in (\mathbb{Z} \times \{-1, 1\})^n$ the transformed set obtained with the transformation function T defined above. Further let $\kappa : \mathbb{Z} \to \mathbb{R}$ be a PDS kernel, and let $\phi : \mathbb{X} \times \mathbb{Y} \to \mathbb{H}$ be the associated feature mapping function. Then for all $1 > \delta > 0$ with probability at least $(1-\delta)$ over $T(\mathbb{S})$ the following generalization bound holds for all $h_w \in \mathbb{H}_B$:*

$$L^T(h_w) \leq \epsilon_n^T(h_w, T(\mathbb{S})) + \frac{2B\mathfrak{G}(T(\mathbb{S}))}{m\sqrt{K-1}} + 3\sqrt{\frac{\ln(\frac{2}{\delta})}{2m}} \tag{3.10}$$

where $\epsilon_n^T(h, T(\mathbb{S})) = \frac{1}{n}\sum_{i=1}^n \mathscr{L}(\tilde{y}_i h_w(\boldsymbol{Z}_i))$ with the surrogate Hinge loss $\mathscr{L} : t \mapsto \min(1, \max(1-t, 0))$, $L^T(h_w) = \mathbb{E}_{T(\mathbb{S})}[L_n^T(h_w, T(\mathbb{S}))]$ and $\mathfrak{G}(T(\mathbb{S})) = \sqrt{\sum_{i=1}^n d_\kappa(\boldsymbol{Z}_i)}$ with

$$d_\kappa(\mathbf{x}^y, \mathbf{x}^{y'}) = \kappa(\mathbf{x}^y, \mathbf{x}^y) + \kappa(\mathbf{x}^{y'}, \mathbf{x}^{y'}) - 2\kappa(\mathbf{x}^y, \mathbf{x}^{y'})$$

Proof. Exploiting the fact that $\mathscr{L}$ dominates the 0/1 loss and using the fractional Rademacher data-dependent generalization bound proposed for interdependent data

in Theorem 4 of [Usunier et al., 2006] one has

$$L^T(h_w) \leq \epsilon^T(h_w) \leq \hat{\epsilon}_n^T(h_w, T(\mathbb{S})) + \hat{\mathscr{R}}_n^T(\mathscr{L} \circ \mathbb{H}_B, \mathbb{S}) + 3\sqrt{\frac{\chi_T^* \ln(\frac{2}{\delta})}{2n}}$$

Where $\epsilon^T(h_w) = \mathbb{E}_{T(\mathbb{S})}[\hat{\epsilon}_n^T(h_w, T(\mathbb{S}))]$ and $\hat{\mathscr{R}}_n^T(\mathscr{L} \circ \mathbb{H}_B, \mathbb{S})$ is the empirical fractional Rademacher complexity of $\mathscr{L} \circ \mathbb{H}_B$ on $T(\mathbb{S})$. Further, as $\mathscr{L}$ is 1-Lipschitz, so

$$\hat{\mathscr{R}}_n^T(\mathscr{L} \circ \mathbb{H}_B, \mathbb{S}) \leq \hat{\mathscr{R}}_n^T(\mathbb{H}_B, \mathbb{S})$$

where

$$\hat{\mathscr{R}}_n^T(\mathbb{H}_B, \mathbb{S}) = \sum_{k=1}^{K-1} \frac{2\alpha_k}{M} \mathbb{E}_\sigma \sup_{h \in \mathbb{H}_B} \sum_{j=0}^{m-1} \sigma_j h_w(\boldsymbol{Z}_{k+j(K-1)})$$

Now, for all $k \in \{1, .., K-1\}$ and $j \in \{0, .., m-1\}$, let z_{kj} and z'_{kj} be the first and the second pair of $\boldsymbol{Z}_{k+j(K-1)}$, then from the bilinearity of dot product and the Cauchy-Schwartz inequality, $\hat{\mathscr{R}}_n^T(\mathbb{H}_B, \mathbb{S})$ is upper-bounded by

$$\begin{aligned} &\sum_{k=1}^{K-1} \frac{2\alpha_k}{n} \mathbb{E}_\sigma \sup_{h_w \in \mathbb{H}_B} \left\langle \boldsymbol{w}, \sum_{j=0}^{m-1} \sigma_j (\phi(z_{kj}) - \phi(z'_{kj})) \right\rangle \\ &\qquad \leq \sum_{k=1}^{K-1} \frac{2B\alpha_k}{n} \mathbb{E}_\sigma \left\| \sum_{j=0}^{m-1} \sigma_j (\phi(z_{kj}) - \phi(z'_{kj})) \right\| \end{aligned}$$

Further, for all $i, j \in \{0, \ldots, m-1\}^2, i \neq j$, we have $\mathbb{E}_\sigma[\sigma_i \sigma_j] = 0$ so

$$\begin{aligned} \hat{\mathscr{R}}_n^T(\mathbb{H}_B, \mathbb{S}) &\leq \sum_{k=1}^{K-1} \frac{2B\alpha_k}{n} \sqrt{\sum_{j=0}^{m-1} d_\kappa(z_{kj}, z'_{kj})} \\ &= \frac{2B\chi_T^*}{n} \sum_{k=1}^{K-1} \frac{\alpha_k}{\chi_T^*} \sqrt{\sum_{j=0}^{m-1} d_\kappa(z_{kj}, z'_{kj})} \end{aligned}$$

Now as $\sum_{k=1}^{K-1} \frac{\alpha_k}{\chi_T^*} = 1$ and that $t \mapsto \sqrt{t}$ is concave, from Jensen inequality we have

$$\hat{\mathscr{R}}_m^T(\mathbb{H}_B, \mathbb{S}) \leq \frac{2B\chi_T^*}{n} \sqrt{\sum_{k=1}^{K-1} \frac{\alpha_k}{\chi_T^*} \sum_{j=0}^{m-1} d_\kappa(z_{kj}, z'_{kj})}$$

The result follows from rearranging the examples and the equalities $\chi_T^* = K - 1$, and $n = (K-1)m$. □

3.3.3 Preliminary Experiments with mRb

To validate the proposed classification algorithm based on the reduction of multiclass problem to binary, we conducted a series of experiments on text classification.

3.3.3.1 Dataset

We evaluate the proposed method for multi-class classification in a large-scale scenario using DMOZ and Wikipedia datasets of the Large Scale Hierarchical Text Classification challenge (LSHTC 2011) [Partalas et al., 2015]. These datasets contain 27875 and 36504 categories respectively for DMOZ and Wikipedia and they are provided in a pre-processed format using stop-word removal and stemming. The dimension of the vectorial space (d), the size of the training set (m) and the test set are respectively 594158 , 394756 and 104263 for DMOZ and 346299 , 456886 and 81262 for Wikipedia. For each of these datasets we randomly draw several samples with increasing number of classes: 100, 500, 1000, 2000, 3000, 4000, 5000, 7500 and by keeping the same proportion of examples in the training and the test sets than in the initial collections. Various characteristics of these subsets of the two original datesets are listed in Table 3.1.

# of classes	DMOZ			WIKI		
	Train size	Test size	Feature dimension	Train size	Test size	Feature dimension
100	985	258	23382	1481	326	11841
500	4874	1279	66541	7995	1623	32736
1000	9479	2478	102745	15615	3288	47520
2000	18378	4830	177108	30447	6509	74912
3000	27729	7287	202775	45340	9569	85585
4000	37634	9886	264216	63375	13422	113074
5000	47281	12426	271205	76904	16268	114049
7500	103794	26886	371634	91283	20025	122847

Table 3.1: Characteristics of the datasets used in our experiments

3.3.3.2 Feature Representation:

For the feature mapping, we used the following features in the vector representation of $\phi(\mathbf{x}^y)$ (Table 3.2) by considering a class y as a mega-document, constituted by the concatenation of all of the documents in the training set belonging to it. The first 8 features are classical ones employed in learning to rank [Liu et al., 2007] by resembling class and a document to respectively a document and a query. The last

two features represent the distance of an example x to its two nearest neighbors in class y.

Features in the vector representation of $\phi(\mathbf{x}^y)$.		
1. $\sum_{t \in y \cap x} \ln(1 + y_t)$	**2.** $\sum_{t \in y \cap x} \ln(1 + \frac{l_{\mathbb{S}}}{\mathbb{S}_t})$	**3.** $\sum_{t \in y \cap x} I_t$
4. $\sum_{t \in y \cap x} \frac{y_t}{\lvert y \rvert}.I_t$	**5.** $\sum_{t \in y \cap x} \ln(1 + \frac{y_t}{\lvert y \rvert})$	**6.** $\sum_{t \in y \cap x} \ln(1 + \frac{y_t}{\lvert y \rvert}.I_t)$
7. $\sum_{t \in y \cap x} \ln(1 + \frac{y_t}{\lvert y \rvert}.\frac{l_{\mathbb{S}}}{\mathbb{S}_t})$	**8.** $\sum_{t \in y \cap x} 1$	**9.** $d_1(\mathbf{x}^y)$
10. $d_2(\mathbf{x}^y)$		

Table 3.2: Let x_t represent the term frequency of term t in document x, and $\mathbb{V}$ the set of distinct terms within $\mathbb{S}$, then $y_t = \sum_{x \in y} x_t$, $|y| = \sum_{t \in \mathbb{V}} y_t$, $\mathbb{S}_t = \sum_{x \in \mathbb{S}} x_t$, $l_{\mathbb{S}} = \sum_{t \in \mathbb{V}} \mathbb{S}_t$. I_t is the inverse document frequency of term t, $d_1(\mathbf{x}^y)$ and $d_2(\mathbf{x}^y)$ are the distances of x to its two nearest neighbours in class y.

3.3.3.3 Baselines:

To assess the performance of the proposed algorithm, we perform a comparison of the popular state-of-the-art approaches for multiclass classification. We specifically compared the following methods:

- `mRb`: The proposed multiclass to binary reduction approach.
- `OVA`: The Liblinear [Fan et al., 2008b] implementaion of One-Vs-All SVM.
- `OVO`: The Liblinear implementation of One-Vs-One.
- `M-SVM`: The Liblinear implementation of Multiclass SVM (Crammer-Singer algorithm [Crammer and Singer, 2002]).
- $LogT$: Vowpal-Wabbit (a public fast learning system proposed by [Choromanska and Langford, 2015] for extreme multiclass classification). We use their logtree solver for our comparison.

3.3.3.4 Experimental Settings:

- **Platform:** In all of our experiments, we used a server with an intel Xenon 1.8HGz processor and 16GB of RAM.

- **Parameters:** For OVA and M-SVM, we need to choose appropriate value of C parameter. In our experiments, we perform a grid search in the following range of values $\{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$, and use the one that leads to the best performance on the validation set.

- **Evaluation Measures:** Results are evaluated over the test set first using the accuracy. As we have discussed in previous section, one of the prominent challenges for large-scale multiclass classification is class imbalance problem. When the dataset exhibits such behaviour accuracy cannot be considered as a good measure for evaluation. Hence, we also use macro F_1-Measure (we will denote it as $\mathtt{MaF}_1$) as another measure for evaluation , which is the harmonic average of macro precision and macro recall (see section 2.4.3).

3.3.3.5 Evaluation on the largest data part

	DMOZ-7500			Wikipedia-7500		
	Accuracy	$\mathtt{MaF}_1$	N_c	Accuracy	$\mathtt{MaF}_1$	N_c
mRb	$0.499^{\downarrow}{}_{\pm.011}$	$\mathbf{0.352}_{\pm.009}$	0.495	$0.467^{\downarrow}{}_{\pm.023}$	$\mathbf{0.378}_{\pm.012}$	0.551
OVA	$\mathbf{0.549}_{\pm.036}$	$0.282^{\downarrow}{}_{\pm.018}$	0.379	$\mathbf{0.484}_{\pm.029}$	$0.348^{\downarrow}{}_{\pm.017}$	0.489
LogT	$0.311^{\downarrow}{}_{\pm.034}$	$0.096^{\downarrow}{}_{\pm.029}$	0.194	$0.231^{\downarrow}{}_{\pm.035}$	$0.151^{\downarrow}{}_{\pm.021}$	0.287

Table 3.3: Accuracy, $\mathtt{MaF}_1$ of methods that could be trained with 7500 classes of DMOZ and Wikipedia collections. N_c is the proportion of classes that are covered or in other words the fraction of classes that are identified in test set. Statistics are given over 50 random samples of training/test sets.

We start our evaluation by analyzing the performance measures of different approaches on the setting with the largest number of classes we considered in our experiments ($K = 7500$). Table 3.3 summarizes results obtained by mRb, OVA and LogT, as the corresponding training processes of M-SVM and OVO were killed by the system and did not pass the scale. Results are averaged over 50 random splits of tests sets.

We use bold face to indicate the highest performance rates, and the symbol ↓ indicates that performance is significantly worse than the best result, according to a Wilcoxon rank sum test used at a p-value threshold of 0.01 [Lehmann, 1975]. The competitive methods are OVA and mRb with a discrepancy over their accuracy and $\mathtt{MaF}_1$ measures on both collections. To analyze this divergence we estimated the proportion of classes that have been covered, or for which at least one true positive document was found. It comes out that mRb covers 6% to 12% more classes than OVA (that is 465 to 900 more classes on both datasets). The reason here is that OVA is affected by the class imbalance problem especially in the extreme case where classes contain very few documents. For the large scale scenario this problem is accentuated as the class distribution is long-tailed, as for example in DMOZ-7500, more than half of the classes contain less than 5 documents (Figure 2.5).

3.3.3.6 Evaluation on all subsets

We also show the comparison of various baselines on data subsets with increasing number of classes. We analyzed their performance in terms of $\mathtt{MaF}_1$values.

As expected all performance curves decrease monotonically with respect to an increasing number of classes. The breaking points beyond which OVO and M-SVM cannot be trained, happen at the same time on both collections for respectively $K = 500$ and $K = 3000$ classes. The performance of mRb are in between of those of OVA and M-SVM before the breaking point, with a slight advantage for M-SVM, while mRb uniformly outperforms OVA with a larger gap on Wikipedia. We notice that on this collection, mRb achieves for 7500 classes $\mathtt{MaF}_1$ score comparable to the OVA's one for 5000 classes. Comparatively, for $K = 3000$, the numbers of parameters of these two models are roughly 5.4×10^8 to 6.5×10^8 on respectively Wikipedia and DMOZ collections which are $O(10^7)$. However, since we adopt a low-dimensional feature representation, we have a very small parameter size $O(10)$. This low-dimensional representation significantly reduces the complexity of the model, especially for cases with higher number of classes.

3.3.3.7 Evaluation of training time

Another performance comparison we performed was the total training time for all the algorithms. Figure 3.4 summarizes the training time of all methods for an increasing number of classes on DMOZ dataset. In the large-class scenario (such as 7500 classes)

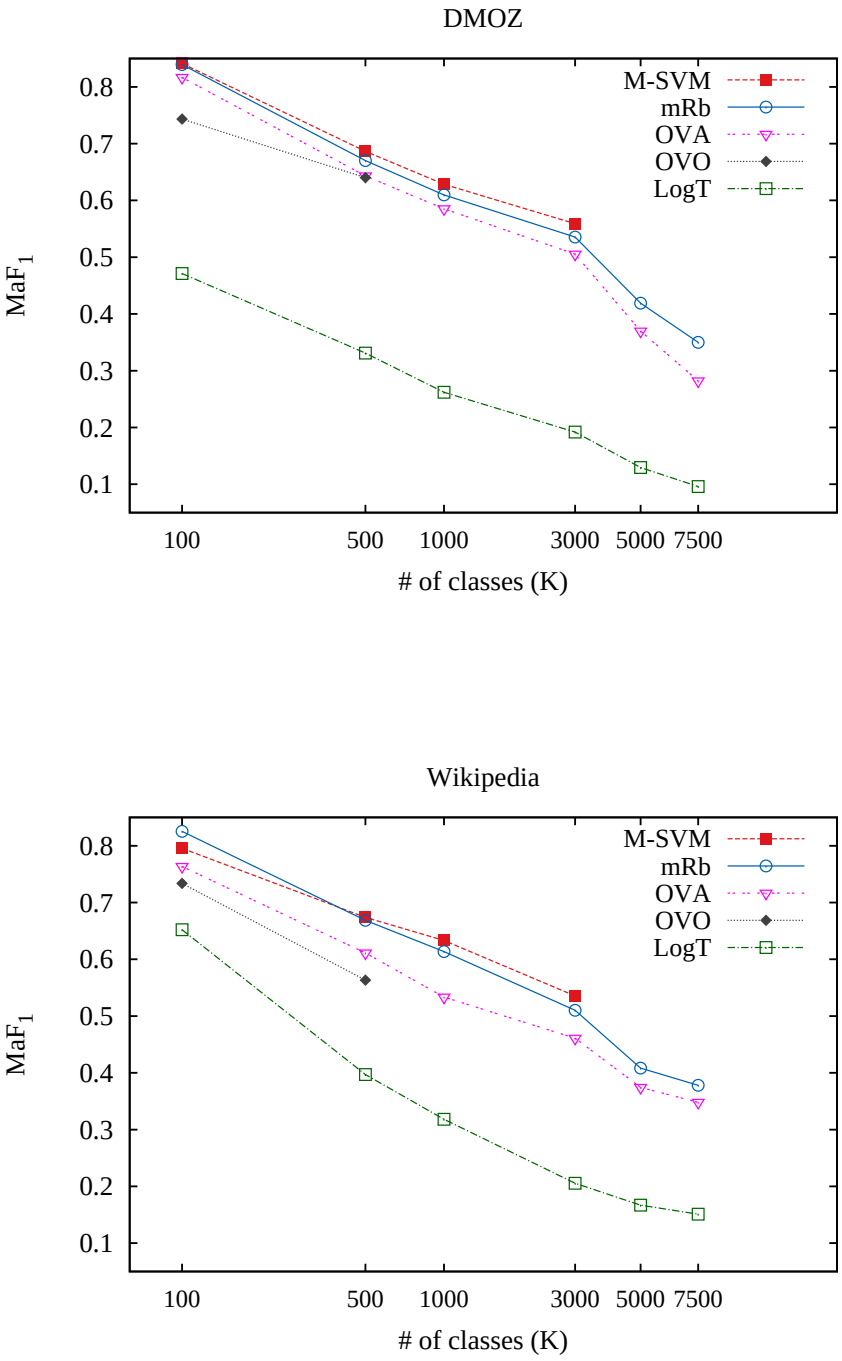


Figure 3.3: $\texttt{MaF}_1$ of all methods with respect to the number of classes for `DMOZ` (top) and `Wikipedia` (down).

we can observe that the only algorithms which are able to pass the scale are `OVA`, `mRb` and `LogT`. Out of which `mRb` performs significantly faster as compared to `OVA`. `LogT` performs quite well in terms of training time, since it obeys a tree structure for training,

which makes its training time logarithmic in the number of classes. However, as we noticed in the previous result, the prediction performance of logarithmic algorithms are not competitive as compared to other state-of-the-art methods.

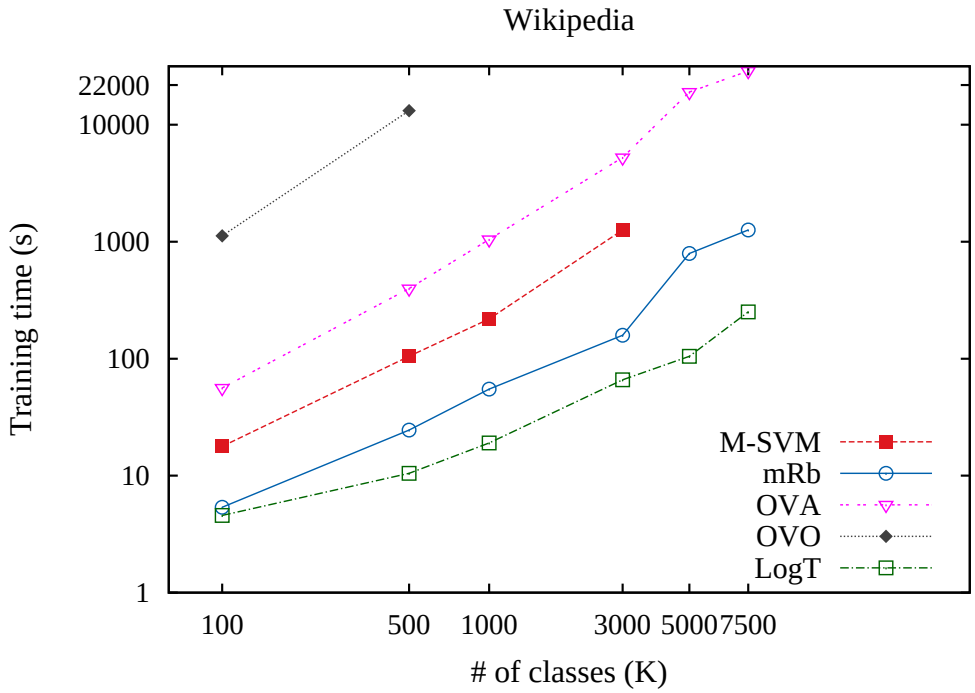


Figure 3.4: Training time in seconds of all methods with respect to the number of classes for `Wikipedia`

3.3.4 New Challenges

In the preliminary results with `mRb`, we noticed that the proposed classification approach was able to tackle the main challenges of multiclass classification. First of all the reduced binary problem has equivalent number of positive and negative examples, hence it does not suffer from class imbalance problem. This fact contributes to the comparable or better predictive performance of the proposed method as compared to the baselines. Additionally, learning a classifier over the binary dataset is significantly faster. We can notice this in the training time comparison plot in Figure 3.4. The use of low-dimensional joint feature representation helped to reduce the feature

dimension significantly. As we already mentioned, the feature dimension in text classification problem can be very huge (up to the order of millions). Hence, the use of dyadic features helped us to restrict this huge feature dimension to as small as just 10 features. This contributes to significantly lower memory requirements as compared to most of baseline approaches such as `OVA`, `OVO` and `M-SVM`.

However, there are a some new challenges associated with the proposed approach. Even though the proposed method scales well enough for large-scale cases, it still has fairly large computational cost and memory usage. Let us discuss this with respect to both phases of the algorithm:

- Reduction Phase: If we denote the number of examples in the training set as m and the total number of classes is K, one complete reduction process refers to $m \times K$ transformations of all (example, class) pairs from original feature space to low-dimensional feature set and $m \times (K-1)$ subtractions of low-dimensional feature representations as can be seen from Algorithm 1. But since large-scale applications involve both higher number of training examples and class size, the computational cost for these operations can become huge. Also, the number of examples in the binary reduced dataset is $m \times K$. Similarly the memory required to store such huge amount of reduced examples becomes quite high.

- Testing Phase: During the testing phase, for each test example x' we first perform K transformations and then calculate the dot product of the learned weight W with each joint representations (x', y_i) for i $\in K$. This again causes large computational cost.

Hence in order to overcome the above-mentioned challenges of `mRb` algorithm, we proposed a modified version of the algorithm denoted as `DS-mRb`, which improves over each of the challenges mentioned above. In the next section, we will discuss the proposed algorithm in detail and in the later sections we will discuss its theoretical and empirical properties.

3.4 Double Sampled Multi to Binary Reduction Algorithm (DS-mRb)

3.4.1 Algorithm Description

First, we will introduce our proposed DS-mRb algorithm by detailing its two main characteristics: (i) an aggressive, doubly sampled, multi-class to binary reduction; and (ii) an efficient prediction method with candidate pre-selection.

3.4.1.1 Aggressive Double Sampling

Earlier we discussed that the transformation of Multi-class to binary, T introduced in Section 3.3 can lead to a large computational overhead. In order to improve the memory/computational complexity, we practice a μ, κ-*double subsampling* on $T(\mathbb{S})$ by:

1. For each class $k \in \{1, \ldots, K\}$, draw randomly a set $\mathbb{S}_{\pi_k}$ of examples from $\mathbb{S}$ of that class with probability π_k, and let $\mathbb{S}_\pi = \bigcup_{k=1}^{K} \mathbb{S}_{\pi_k}$;
2. For each example $\mathbf{x}^y$ in $\mathbb{S}_\pi$, draw uniformly κ adversarial classes in $\mathbb{Y}\backslash\{y\}$.

After this double sampling, we construct our transformed problem as in Eq. (3.7), leading to a set $T_\kappa(S_\mu)$ of size $\mu\kappa K$. Algorithm 3 presents in pseudocode the μ, κ-*double subsampled* reduction of the multiclass problem.

This *aggressive double sampling* practically leads to dramatic improvements in terms of memory consumption, computational complexity, and a higher multiclass prediction accurracy. Majority of large-scale multiclass classification datasets exhibit a long-tailed distribution [Babbar et al., 2014b], which implies that most of the classes contain very few examples, especially when the number of classes is large. In order not to miss out those rare classes during sampling, we first sample randomly a few training examples from each class. This also avoids having a large number of very similar examples in one class leading to minimal performance improvement.

Still, in cases where the number of classes K becomes large, the initialization of (subsampled) set $T(\mathbb{S}_\mu)$ can be a computational bottleneck. However, thanks to our loss formulation (Eq. 3.1), its size can be sensibly reduced. Indeed, $e(g, \mathbf{x}^y)$ can be seen as the expectation of $1_{g(\mathbf{x}^y) \leq g(\mathbf{x}^{y'})}$ on y' uniformly over $\mathbb{Y}\backslash\{y\}$:

Input: Labeled training set $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m$

initialization: $\mathbb{S}_\pi \leftarrow \emptyset$;

$T_\kappa(\mathbb{S}_\pi) \leftarrow \emptyset$;

for $k = 1..K$ **do**

  Draw randomly a set $\mathbb{S}_{\pi_k}$ of examples of class k from $\mathbb{S}$ with probability π_k;

  $\mathbb{S}_\pi \leftarrow \mathbb{S}_\pi \cup \mathbb{S}_{\pi_k}$;

end

forall $\mathbf{x}^y \in \mathbb{S}_\pi$ **do**

  Draw uniformly a set $\mathbb{Y}_{\mathbf{x}^y}$ of κ classes from $\mathbb{Y}\backslash\{y\}$ ▷ $\kappa \ll K$;

  forall $k \in \mathbb{Y}_{\mathbf{x}^y}$ **do**

    if $k < y$ **then**

      $T_\kappa(\mathbb{S}_\pi) \leftarrow T_\kappa(\mathbb{S}_\pi) \cup \big(\boldsymbol{Z} = \big(\phi(\mathbf{x}^k), \phi(\mathbf{x}^y)\big), \quad \tilde{y} = -1\big)$;

    end

    else

      $T_\kappa(\mathbb{S}_\pi) \leftarrow T_\kappa(\mathbb{S}_\pi) \cup \big(\boldsymbol{Z} = \big(\phi(\mathbf{x}^y), \phi(\mathbf{x}^k)\big), \quad \tilde{y} = +1\big)$;

    end

  end

end

return $T_\kappa(\mathbb{S}_\pi)$

Algorithm 3: `DS-mRb`

$$e(g, \mathbf{x}^y) = \frac{1}{K-1} \sum_{y' \in \mathbb{Y}\backslash\{y\}} 1_{g(\mathbf{x}^y) \leq g(\mathbf{x}^{y'})} \approx \mathbb{E}_{y'}[1_{g(\mathbf{x}^y) \leq g(\mathbf{x}^{y'})}].$$

It means that one can define a new empirical loss by sampling over the classes. Let $\kappa \leq K-1$ be the number of classes to investigate per example. For each example $\mathbf{x}^y$, draw uniformly a κ-tuple $\mathbb{K}$ of distinct elements of $\mathbb{Y}\backslash\{y\}$. The new subsampled loss over $\mathbb{S}_\mu = (\mathbf{x}_i^{y_i})_{i=1,..,\mu K}$ is:

$$\hat{\hat{L}}_{\mu,\kappa}(g, \mathbb{S}) = \frac{1}{\mu K} \sum_{i=1}^{\mu K} \left(\frac{1}{\kappa} \sum_{y' \in \mathbb{Y}_{\mathbf{x}^y}} 1_{g(\mathbf{x}_i^{y_i}) \leq g(\mathbf{x}_i^{y'})} \right). \tag{3.11}$$

This loss is an unbiased estimator of the normalized loss $\overline{L}(g) = \mathbb{E}_{\mathbf{x}^y \sim \overline{\mathbb{D}}}[e(g, \mathbf{x}^y)] = \mathbb{E}_{\mathbf{x}^y \sim \overline{\mathbb{D}}}\big[\mathbb{E}_{y'}[1_{g(\mathbf{x}^y) \leq g(\mathbf{x}^{y'})} | y = y]\big]$ where $\overline{\mathbb{D}}$ is the distribution of the examples after class normalization. Thus, this new loss enables to approximate the statistical loss L at a much lower computational cost than the classical emprical loss. Finally, one can notice

that such reasoning does not hold for the classical loss of Eq. 3.2 due to its non-linear formulation.

3.4.1.2 Prediction with Candidate Selection

After learning over our reduced problem using the $T_\kappa(\mathbb{S}_\mu)$ dataset obtained after aggressive sampling, we obtain a vector $\mathbf{w}$ such that, for an observation $\mathbf{x}$, the larger $\langle \mathbf{w}, \phi(\mathbf{x}^y) \rangle$ over y is, the more likely $\mathbf{x}$ belongs to class y.

However in the large class scenario, computing the feature representation for all classes may require a huge amount of time. So, in order to improve the prediction time, we apply the trick of selecting a small subset of candidate classes beforehand. For a new observation $\mathbf{x}$, the candidate set denoted as $\mathbb{K}_\sigma$ contains the σ nearest classes for the test example, based on the centroid distance of test example vector with the class centroids. Class centroids are computed by taking mean of all the examples of that particular class.

Candidate set is selected by computing the cosine distance between a test example vector and each class centroid vectors and selecting the σ nearest ones. Note that class centroid may already have been computed in the preliminary feature representation and thus represent no additional computation. Algorithm 4 presents the pseudocode of prediction with candidate selection.

Input: Unlabeled test set $\mathbb{T} = (\mathbf{x}_i)_{i=1}^T$

Learned feature weight vector $\mathbf{w}$

Initialize:

$P \leftarrow \emptyset$

forall $\mathbf{x} \in \mathbb{T}$ **do**

  Select $\mathbb{K}_\sigma$ candidate set of σ nearest-centroid classes for $\mathbf{x}$

  $P \leftarrow P \cup \text{argmax}_{k \in \mathbb{K}_\sigma} \langle \mathbf{w}, \phi(\mathbf{x}^k) \rangle$

end

return predicted classes P

Algorithm 4: Prediction with Candidate Selection Algorithm

3.4.2 Generalization Bound Analysis using Local Fractional Rademacher Complexity

In this work, we derive a new generalization bounds based on Local Rademacher Complexities introduced in [Ralaivola and Amini, 2015] that implies second-order (i.e. variance) information inducing faster convergence rates (Theorem 3.2). Our analysis relies on the notion of graph covering introduced in [Janson, 2004] and defined as :

Definition 3.1 (Exact proper fractional cover of $\mathcal{G}$, [Janson, 2004]). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph. $\mathbb{C} = \{(\mathbb{C}_k, \omega_k)\}_{k\in[J]}$, for some positive integer J, with $\mathbb{C}_k \subseteq \mathcal{V}$ and $\omega_k \in [0,1]$ is an exact proper fractional cover of $\mathcal{G}$, if:*

1. *it is* proper: *$\forall k$, $\mathbb{C}_k$ is an* independent set*, i.e., there is no connections between vertices in $\mathbb{C}_k$;*
2. *it is an* exact fractional cover *of G: $\forall v \in \mathcal{V}$, $\sum_{k:v\in\mathbb{C}_k} \omega_k = 1$.*

The weight $W(\mathbb{C})$ of $\mathbb{C}$ is given by: $W(\mathbb{C}) \doteq \sum_{k\in[J]} \omega_k$ and the minimum weight $\chi^*(\mathcal{G}) = \min_{\mathbb{C}\in\mathscr{K}(\mathcal{G})} W(\mathbb{C})$ over the set $\mathscr{K}(\mathcal{G})$ of all exact proper fractional covers of $\mathcal{G}$ is the *fractional chromatic number* of $\mathcal{G}$.

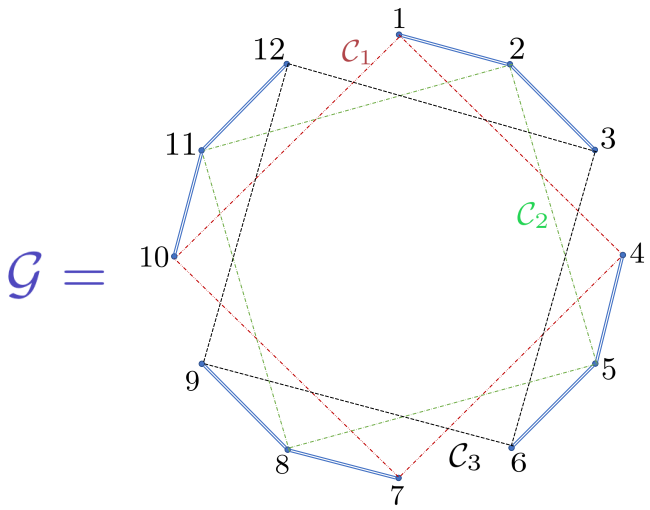


Figure 3.5: The dependency graph $\mathcal{G} = \{1, \ldots, 12\}$ corresponding to the toy problem of Figure 3.1, where dependent nodes are connected with vertices in blue double-line. The exact proper fractional cover $\mathbb{C}_1$, $\mathbb{C}_2$ and $\mathbb{C}_3$ is shown in dashed. The fractional chromatic number is in this case $\chi^*(\mathcal{G}) = K - 1 = 3$.

From this statement, [Janson, 2004] extended Hoeffding's inequality and proposed large deviation bounds for sums of dependent random variables which was the precursor of new generalisation bounds, including a Talagrand's type inequality for empirical processes in the dependent case presented in [Ralaivola and Amini, 2015].

With the classes of functions $\mathbb{G}$ and $\mathbb{H}$ introduced previously, consider the parameterized family $\mathbb{H}_r$ which, for $r > 0$, is defined as:

$$\mathbb{H}_r = \{h : h \in \mathbb{H}, \mathbb{V}[h] \doteq \mathbb{V}_{\mathbf{Z},\tilde{y}}[1_{\tilde{y}h(\mathbf{Z})}] \leq r\},$$

where $\mathbb{V}$ denotes the variance. The fractional Rademacher complexity introduced in [Usunier et al., 2005] entails our analysis :

$$\mathfrak{R}_{T(\mathbb{S})}(\mathbb{H}) \doteq \frac{2}{N} \mathbb{E}_{\xi} \sum_{k \in [K-1]} \omega_k \mathbb{E}_{\mathbb{C}_k} \sup_{h \in \mathbb{H}} \sum_{\substack{\alpha \in \mathbb{C}_k \\ \mathbf{z}_\alpha \in T(\mathbb{S})}} \xi_\alpha h(\mathbf{Z}_\alpha),$$

with $(\xi_i)_{i=1}^N$ a sequence of independent Rademacher variables verifying $\mathbb{P}(\xi_n = 1) = \mathbb{P}(\xi_n = -1) = \frac{1}{2}$. If other is not specified explicitly we assume below all $\omega_k = 1$.

Our first result that bounds the generalization error of a function $h \in \mathbb{H}$; $R(h) = \mathbb{E}_{T(\mathbb{S})}[\tilde{R}_{T(\mathbb{S})}(h)]$, with respect to its empirical error $\tilde{R}_{T(\mathbb{S})}(h)$ over a transformed training set, $T(\mathbb{S})$, and the fractional Rademacher complexity, $\mathfrak{R}_{T(\mathbb{S})}(\mathbb{H})$, states as :

Theorem 3.2. *Let $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m \in (\mathbb{X} \times \mathbb{Y})^m$ be a dataset of m examples drawn i.i.d. according to a probability distribution $\mathbb{D}$ over $\mathbb{X} \times \mathbb{Y}$ and $T(\mathbb{S}) = ((\mathbf{Z}_i, \tilde{y}_i))_{i=1}^N$ the transformed set obtained as in Eq.* (3.7)*. Then for any $1 > \delta > 0$ and $0/1$ loss $\ell : \{-1, +1\} \times \mathbb{R} \to [0, 1]$, with probability at least $(1 - \delta)$ the following generalization bound holds for all $h \in \mathbb{H}_r$:*

$$R(h) \leq \tilde{R}_{T(\mathbb{S})}(h) + \mathfrak{R}_{T(\mathbb{S})}(\ell \circ \mathbb{H}_r) + \frac{5}{2}\left(\sqrt{\mathfrak{R}_{T(\mathbb{S})}(\ell \circ \mathbb{H}_r)} + \sqrt{\frac{r}{2}}\right)\sqrt{\frac{\log \frac{1}{\delta}}{m}} + \frac{25}{48}\frac{\log \frac{1}{\delta}}{m}.$$

Lemma 1. *Fractional chromatic number is monotone in graph inclusion: if $\mathscr{G} = \langle \mathscr{V}_{\mathscr{G}}, \mathscr{E}_{\mathscr{G}} \rangle \subseteq \mathscr{H} = \langle \mathscr{V}_{\mathscr{H}}, \mathscr{E}_{\mathscr{H}} \rangle$ implies $\mathscr{V}_{\mathscr{G}} \subseteq \mathscr{V}_{\mathscr{H}}$ and $\mathscr{E}_{\mathscr{G}} \subseteq \mathscr{E}_{\mathscr{H}}$ we have $\chi^*(\mathscr{G}) \leq \chi^*(\mathscr{H})$.*

Proof. Consider any exact proper fractional cover [Janson, 2004] of graph $\mathscr{H}$, $\mathbb{C}_{\mathscr{H}} = \{(\mathbb{C}_k, \omega_k)\}_{k \in J}$ for some index set J. By removing from each $\mathbb{C}_k$ vertices that belong to $\mathscr{V}_{\mathscr{H}} \setminus \mathscr{V}_{\mathscr{G}}$ and incident edges we get a cover $\mathbb{C}_{\mathscr{G}} = \{(\mathbb{C}'_k, \omega_k)\}_{k \in J}$ of graph $\mathscr{G}$. Once for a certain k holds $\mathbb{C}'_k = \varnothing$ we remove it from $\mathbb{C}_{\mathscr{H}}$ which is essentially the same as assignment $\omega_k \doteq 0$.

The cover $\mathbb{C}_{\mathscr{G}}$ is a proper fractional cover of $\mathscr{G}$ since the number of connections between vertices in $\mathbb{C}'_k$ is a subset of those in $\mathbb{C}_k$ for any $k \in J$. The cover $\mathbb{C}_{\mathscr{G}}$ is also exact (modulo empty sets in $\mathbb{C}_{\mathscr{H}}$) since for any v :

$$v \in \mathscr{V}_{\mathscr{H}} \cap \mathscr{V}_{\mathscr{G}} : \quad \sum_{k:v\in\mathbb{C}'_k} \omega_k = \sum_{k:v\in\mathbb{C}_k} \omega_k = 1,$$

where $\mathbb{C}_{\mathscr{H}} = \{(\mathbb{C}_k, \omega_k)\}_{k\in J}$ is an exact proper fractional cover of graph $\mathscr{H}$. That implies that each exact proper fractional cover $\mathbb{C}_{\mathscr{H}}$ of graph $\mathscr{H}$ can be converted to an exact proper fractional cover $\mathbb{C}_{\mathscr{G}}$ of graph $\mathscr{G}$ without increasing the covering cost $W(\mathbb{C}_{\mathscr{G}}) \doteq \sum_{\mathbb{C}_k\in\mathbb{C}_{\mathscr{G}}} \omega_k \leq W(\mathbb{C}_{\mathscr{H}})$. Denote the set of all exact proper fractional coverings of graph $\mathscr{G}$ as $\mathscr{K}(\mathscr{G})$ and coverings obtained by pruning $\mathscr{K}(\mathscr{H})$ as above through $\mathscr{K}_{\mathscr{H}}(\mathscr{G})$.

By the definition of fractional chromatic number we have

$$\chi^*(\mathscr{G}) = \min_{\mathbb{C}\in\mathscr{K}(\mathscr{G})} W(\mathbb{C}) \overset{(1)}{\leq} \min_{\mathbb{C}\in\mathscr{K}_{\mathscr{H}}(\mathscr{G})} W(\mathbb{C}) \leq \chi^*(\mathscr{H}),$$

where (1) is implied by inclusion $\mathscr{K}_{\mathscr{H}}(\mathscr{G}) \subseteq \mathscr{K}(\mathscr{G})$. □

Lemma 2 (Empirical Bennet inequality, theorem 4 of [Maurer and Pontil, 2009]). *Let $Z_1, Z_2, \ldots, Z_n$ be i.i.d. variables with values in $[0,1]$ and let $\delta > 0$. Then with probability at least $1-\delta$ in $\mathbf{Z} = (Z_1, Z_2, \ldots, Z_n)$ we have*

$$\mathbb{E}[\mathbf{Z}] - \frac{1}{n}\sum_{i=1}^{n} Z_i \leq \sqrt{\frac{2\mathbb{V}_n(\mathbf{Z})\log 2/\delta}{n}} + \frac{7\log 2/\delta}{3(n-1)},$$

where $\mathbb{V}_n(\mathbf{Z})$ is the sample variance

$$V_n(\mathbf{Z}) = \frac{1}{n(n-1)} \sum_{1\leq i<j\leq n} (Z_i - Z_j)^2.$$

Lemma 3 (Concentration of Fractionally Sub-Additive Functions, theorem 3 of [Ralaivola and Amini, 2015]). *Let $\mathbb{H}$ be a set of functions from $\mathbb{X}$ to $\mathbb{R}$ and assume all functions in $\mathbb{H}$ are measurable, square-integrable and satisfy $\mathbb{E}[f(X_n)] = 0, \forall n \in [N]$ and $\sup_{f\in\mathbb{H}} \|f\|_\infty \leq 1$. Assume that $\mathbb{C} = \{(\mathbb{C}_k, \omega_k)\}_{k\in J}$ is a cover of the dependency graph of $X_{[N]}$ and let $\chi_f \doteq \sum_k \omega_k$.*

Let us define:

$$Z \doteq \sum_{k\in[J]} \omega_k \sup_{f\in\mathbb{H}} \sum_{n\in\mathbb{C}_k} f(X_n)$$

Let σ_k *be so that* $\sigma_k^2 = \sum_{n\in\mathbb{C}_k} \sup_{f\in\mathbb{H}} \mathbb{E}\left[f^2(X_n)\right]$, $v \doteq \sum_k \omega_k\sigma_k^2 + 2\mathbb{E}[Z]$, *and* $c \doteq 25\chi_f/16$. *Then, for any* $t \geq 0$.

$$\mathbb{P}\left(Z \geq \mathbb{E}[Z] + \sqrt{2cvt} + \frac{ct}{3}\right) \leq e^{-t} \tag{3.12}$$

Let below $\mathbb{D}$ be a probability measure over $\mathbb{X}\times\mathbb{Y}$. It can be decomposed into a direct product of $\mathbb{D} = \mathbb{D}_{\mathbb{Y}} \times \mathbb{D}_{\mathbb{X}|\mathbb{Y}}$ with marginal distribution $\mathbb{D}_{\mathbb{Y}}$ over $\mathbb{Y}$ and conditional $\mathbb{D}_{\mathbb{X}|\mathbb{Y}}$ over $\mathbb{X}$. Let $\bar{\mathbb{D}} = \bar{\mathbb{D}}_{\mathbb{Y}} \times \mathbb{D}_{\mathbb{X}|\mathbb{Y}}$ be a measure properly renormalized in accordance with the algorithm, e.g. $\mathbb{P}_{y\sim\bar{\mathbb{D}}}[y(\mathbf{x}) = y] = \pi_y/\pi$, where $\pi = \sum_{i=y}^{K} \pi_y$.

Lemma 4. *Let* $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m \in (\mathbb{X}\times\mathbb{Y})^m$ *be a dataset of* m *examples drawn i.i.d. according to a probability measure* $\mathbb{D} = \mathbb{D}_{\mathbb{Y}} \times \mathbb{D}_{\mathbb{X}|\mathbb{Y}}$ *over* $\mathbb{X}\times\mathbb{Y}$ *and* $T(\mathbb{S}) = ((\boldsymbol{Z}_i, \tilde{y}_i))_{i=1}^N$ *the transformed set obtained with the transformation function* T *defined in Eq.* (3.7). *Let* $\bar{\mathbb{D}} = \bar{\mathbb{D}}_{\mathbb{Y}} \times \mathbb{D}_{\mathbb{X}|\mathbb{Y}}$, $\mathbb{P}_{\bar{\mathbb{D}}_{\mathbb{Y}}}[y(\mathbf{x}) = k] = \pi_k/\pi$, $1 \leq i \leq K$, *be a measure over* $\mathbb{X}\times\mathbb{Y}$ *used in the* (π,κ)-mRb *algorithm. With the class of functions* $\mathbb{G} = \{g : \mathbb{X}\times\mathbb{Y} \to \mathbb{R}\}$ *and* $\mathbb{H} = \{h : h(\phi(\mathbf{x}^y), \phi(\mathbf{x}^{y'})) = g(\mathbf{x}^y) - g(\mathbf{x}^{y'}), g \in \mathbb{G}\}$ *for any* $\delta > 0$ *for all* $h \in \mathbb{H}$ *with probability at least* $1-\delta$ *we have :*

$$R(h) \leq \alpha\mathbb{E}_{\mathbf{x}^y\sim\bar{\mathbb{D}}}\tilde{R}_{T(\mathbf{x}^y)}(h) + \sqrt{\frac{2\alpha\log 2K/\delta}{\beta(m-1)}} + \frac{7\beta\log 2K/\delta}{3(m-1)}.$$

holds the for all $h \in \mathbb{H}$, *where* $\ell : \{-1,+1\}\times\mathbb{R} \to [0,1]$ *is the* $0/1$ *loss, and* $\alpha = \max_{y:1\leq y\leq K} \pi\eta_y/\pi_y$, *and* $\beta = \max_{y:1\leq y\leq K} \pi/\pi_y$, *and* $\eta_y > 0$ *is the proportion the class* y *in the training set* $\mathbb{S}$.

Proof. First, decompose the expected risk $R(h)$ as a sum of the conditional risks over the classes

$$\begin{aligned} R(h) = \mathbb{E}_{\mathbf{x}^y\sim\mathbb{D}}[e_h(\mathbf{x}^y)] &\overset{(1)}{=} \mathbb{E}_{y\sim\mathbb{D}_{\mathbb{Y}}}\mathbb{E}_{\mathbf{x}^y\sim\mathbb{D}|y(\mathbf{x})=y}[e_h(\mathbf{x}^y)|y(\mathbf{x}) = y] \\ &\overset{(2)}{=} \sum_{y=1}^{K} \mathbb{P}_{\mathbf{x}^y\sim\mathbb{D}}[y(\mathbf{x}) = y]\cdot\mathbb{E}_{\mathbf{x}^y\sim\mathbb{D}|y(\mathbf{x})=y}[e_h(\mathbf{x}^y)|y(\mathbf{x}) = y], \end{aligned} \tag{3.13}$$

where (1) and (2) are due to the law of total expectation.

Similarly consider the expected loss $\mathbb{E}_{\mathbf{x}^y\sim\bar{\mathbb{D}}}\tilde{R}_{T(\mathbf{x}^y)}(h)$:

$$\begin{aligned} \mathbb{E}_{\mathbf{x}^y\sim\bar{\mathbb{D}}}\tilde{R}_{T(\mathbf{x}^y)}(h) = \mathbb{E}_{\mathbf{x}^y\sim\bar{\mathbb{D}}}[e_h(\mathbf{x}^y)] &\overset{(1)}{=} \mathbb{E}_{y\sim\bar{\mathbb{D}}_{\mathbb{Y}}}\mathbb{E}_{\mathbf{x}^y\sim\mathbb{D}_{\mathbb{X}|\mathbb{Y}},y(\mathbf{x})=y}[e_h(\mathbf{x}^y)|y(\mathbf{x}) = y] \\ &\overset{(2)}{=} \sum_{y=1}^{K} \mathbb{P}_{\mathbf{x}^y\sim\bar{\mathbb{D}}}[y(\mathbf{x}) = y]\cdot\mathbb{E}_{\mathbf{x}^y\sim\mathbb{D}|y(\mathbf{x})=y}[e_h(\mathbf{x}^y)|y(\mathbf{x}) = y] \\ &= \sum_{y=1}^{K} \frac{\pi_y}{\pi}\cdot\mathbb{E}_{\mathbf{x}^y\sim\mathbb{D}|y(\mathbf{x})=y}[e_h(\mathbf{x}^y)|y(\mathbf{x}) = y], \end{aligned} \tag{3.14}$$

where (1) and (2) are also due to the law of total expectation.

From (3.13) and (3.14) we conclude

$$R(h) \leq \max_{y:1\leq y\leq K} \frac{\mathbb{P}_{\mathbf{x}^y\sim\mathbb{D}}[y(\mathbf{x})=y]}{\pi_y/\pi} \cdot \mathbb{E}_{\mathbf{x}^y\sim\bar{\mathbb{D}}}\tilde{R}_{T(\mathbf{x}^y)}(h) \tag{3.15}$$

Finally, we need to bound the multiplier in front of $\mathbb{E}_{\mathbf{x}^y\sim\bar{\mathbb{D}}}\tilde{R}_{T_\kappa(\mathbb{S}_\pi)}(h)$ in Eq. (3.15). Denote through η_y an empirical probability of the class $y \in \mathbb{Y}$:

$$\eta_y = \frac{1}{m}\sum_{\mathbf{x}\in\mathbb{S}} 1_{y(\mathbf{x})=y}.$$

Note, that empirical variance $\mathbb{V}_n(\eta_y)$ in accordance with lemma 2 is

$$\mathbb{V}_m(\eta_y) = \frac{\eta_y(1-\eta_y)m}{(m-1)}$$

For any $y \in \mathbb{Y}$ we have with probability at least $1-\delta/K$ by lemma 2 :

$$\begin{aligned}\mathbb{P}_{\mathbf{x}^y\sim\mathbb{D}}[y(\mathbf{x})=y] \leq \eta_y + \sqrt{\frac{2\mathbb{V}_m(\eta_y)\log 2K/\delta}{m}} + \frac{7\log 2K/\delta}{3(m-1)} \overset{(1)}{=} \\ \eta_y + \sqrt{\frac{2\eta_y(1-\eta_y)\log 2K/\delta}{m-1}} + \frac{7\log 2K/\delta}{3(m-1)} \overset{(2)}{\leq} \\ \eta_y + \sqrt{\frac{2\eta_y\log 2K/\delta}{m-1}} + \frac{7\log 2K/\delta}{3(m-1)},\end{aligned}$$

where (1) is a substitution of $\mathbb{V}_m(\eta_y)$ by its explicit value; (2) is due to the fact that $0 < \eta_y \leq 1$.

Then simultaneously for all $y \in \mathbb{Y}$ we have with probability at least $1-\delta$:

$$\mathbb{P}_{\mathbf{x}^y\sim\mathbb{D}}[y(\mathbf{x})=y] \leq \eta_y + \sqrt{\frac{2\eta_y\log 2K/\delta}{m-1}} + \frac{7\log 2K/\delta}{3(m-1)}$$

Thus with probability at least $1-\delta$:

$$\max_{y:1\leq y\leq K} \frac{\mathbb{P}_{\mathbf{x}^y\sim\mathbb{D}}[y(\mathbf{x})=y]}{\pi_y/\pi} \leq \alpha + \sqrt{\frac{2\alpha\log 2K/\delta}{\beta(m-1)}} + \frac{7\beta\log 2K/\delta}{3(m-1)}, \tag{3.16}$$

with

$$\alpha = \max_{y:1\leq y\leq K}\frac{\pi\eta_y}{\pi_y}, \quad \beta = \max_{y:1\leq y\leq K}\frac{\pi}{\pi_y}$$

From equations (3.15) and (3.16) and the fact that $\mathbb{E}_{\mathbf{x}^y\sim\bar{\mathbb{D}}^N}\tilde{R}_{T(\mathbf{x}^y)}(h) \leq 1$, we have with probability at least $1-\delta$:

$$R(h) \leq \alpha\mathbb{E}_{\mathbf{x}^y\sim\bar{\mathbb{D}}}\tilde{R}_{T(\mathbf{x}^y)}(h) + \sqrt{\frac{2\alpha\log 2K/\delta}{\beta(m-1)}} + \frac{7\beta\log 2K/\delta}{3(m-1)}.$$

□

The results of the previous lemmas, hence entail the following lemma.

Lemma 5. *Let $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m \in (\mathbb{X}\times\mathbb{Y})^m$ be a dataset of m examples drawn i.i.d. according to a probability distribution $\mathbb{D}$ over $\mathbb{X}\times\mathbb{Y}$ and $T_\kappa(\mathbb{S}) = ((\boldsymbol{Z}_i, \tilde{y}_i))_{i=1}^{m\kappa}$ the transformed set obtained as in Eq.* (3.7) *and draw κ adversarial samples by algorithm* DS-mRb. *With the class of functions $\mathbb{G} = \{g : \mathbb{X}\times\mathbb{Y} \to \mathbb{R}\}$ and $\mathbb{H} = \{h : h(\phi(\mathbf{x}^y), \phi(\mathbf{x}^{y'})) = g(\mathbf{x}^y) - g(\mathbf{x}^{y'}), g \in \mathbb{G}\}$, consider the parameterized family $\mathbb{H}_r$ which, for $r > 0$, is defined as :*

$$\mathbb{H}_r = \{h : h \in \mathbb{H}, \mathbb{V}[h] \doteq \mathbb{V}_{\boldsymbol{Z},\tilde{y}}[1_{\tilde{y}h(\boldsymbol{Z})}] \leq r\},$$

where $\mathbb{V}$ denotes the variance. Then for any $\delta > 0$ and 0/1 loss $\ell : \{-1,+1\}\times\mathbb{R} \to [0,1]$, with probability at least $(1-\delta)$ the following generalization bound holds for all $h \in \mathbb{H}_r$:

$$R(h) \leq \tilde{R}_{T_\kappa(\mathbb{S})}(h) + \mathfrak{R}_{T_\kappa(\mathbb{S})}(\ell\circ\mathbb{H}_r) + \frac{5}{2}\left(\sqrt{\mathfrak{R}_{T_\kappa(\mathbb{S})}(\ell\circ\mathbb{H}_r)} + \sqrt{\frac{r}{2}}\right)\sqrt{\frac{(K-1)\log 1/\delta}{m\kappa}} + \frac{25}{48}\frac{\log 1/\delta}{m}.$$

Proof. Consider the function Φ defined as:

$$\Phi(\underline{X}, r) \doteq N \sup_{h\in\mathbb{H}_r}\left[\mathbb{E}_{\underline{X}'}[\tilde{R}_{T(\underline{X}')}(h)] - \tilde{R}_{T(\underline{X}')}(h)\right],$$

where $\underline{X}'$ is an i.i.d. copy of $\underline{X}$ and where we have used the notation $\mathbb{E}_{\underline{X}'}[\tilde{R}_{T(\underline{X}')}(h)]$ for $\mathbb{E}_{T(\mathbb{S})}\hat{R}_N(h, T(\mathbb{S}))$ to make explicit the dependence on the sequence of dependent variables $\underline{X}'$. It is easy to see that:

$$\begin{aligned}\Phi(\underline{X}, r) &\leq \sum_{k\in[K-1]} \omega_k \sup_{h\in\mathbb{H}_r}\sum_{\alpha\in\mathbb{C}_k}\left[\mathbb{E}_{(\tilde{y}',\boldsymbol{Z}')}[1_{\tilde{y}'h(\boldsymbol{Z}')}] - 1_{\tilde{y}_\alpha h(\boldsymbol{Z}_\alpha)}\right] \\ &= Z. \end{aligned} \tag{3.17}$$

Lemma 3 readily applies to upper bound the right hand side of (3.17). Therefore, for $t > 0$, the following holds with probability at least $1 - e^{-t}$:

$$\Phi(\underline{X}, r) \leq \mathbb{E}[Z] + \sqrt{2cvt} + \frac{ct}{3},$$

where $c = 25\chi_f/16 = 25(K-1)/16$ and $v \leq Nr + 2\mathbb{E}[Z]$. Using $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ and $2\sqrt{ab} \leq ua + b/u$ for all $u > 0$, we get,

$$\forall u > 0, \Phi(\underline{X}, r) \leq (1+u)\mathbb{E}[Z] + \sqrt{2cNrt} + \left(\frac{1}{3} + \frac{1}{u}\right)ct.$$

Furthermore, with a simple symmetrization argument, we have,

$$\mathbb{E}[Z] = \mathbb{E}\left[\sum_{k\in[K-1]} \omega_k \sup_{h\in\mathbb{H}_r} \sum_{\alpha\in\mathbb{C}_k} \left[\mathbb{E}_{\tilde{y}',Z'}[1_{\tilde{y}'h(Z')}] - 1_{\tilde{y}_\alpha h(Z_\alpha)}\right]\right] \leq N\mathfrak{R}(\ell\circ\mathbb{H}_r),$$

with $\omega_k = 1$ for all k since the fractional chromatic number of the dependency graph corresponds to the sample $T(\mathbb{S})$ equals to $K-1$ and stands for the covering determined by Eq. (4) with unit weights ω_k.

Further, as $N = m\kappa$, and fractional chromatic number of $T_\kappa(\mathbb{S}) \leq T(\mathbb{S}) = K-1$ (theorem 1 of [Joshi et al., 2015a]), with probability at least $1-e^{-t}$, we have for all $h \in \mathbb{H}_r$

$$R(h) - \tilde{R}_{T_\kappa(\mathbb{S})}(h) \leq \inf_{u>0}\left((1+u)\mathfrak{R}_{T_\kappa(\mathbb{S})}(\ell\circ\mathbb{H}_r) + \frac{5}{4}\sqrt{\frac{2(K-1)rt}{m\kappa}} + \frac{25}{16}\left(\frac{1}{3}+\frac{1}{u}\right)\frac{(K-1)t}{\kappa m}\right). \quad (3.18)$$

The minimum of the right hand side of the inequality (3.18) is reached for $u^* = \frac{5}{4}\sqrt{\frac{(K-1)t}{\kappa m\mathfrak{R}_{T_\kappa(\mathbb{S})}(\ell\circ\mathbb{H}_r)}}$, plugging back the minimizer and solving for $\delta = e^{-t}$ gives the result. □

Proof of the theorem 1. Theorem 1 of [Joshi et al., 2015a] states that fractional chromatic number of $T(\mathbb{S})$ is bounded from above by $K-1$. Then by the lemma 5 with have with probability at least $1-\delta$:

$$R(h) \leq \tilde{R}_{T(\mathbb{S})}(h) + \mathfrak{R}_{T(\mathbb{S})}(\ell\circ\mathbb{H}_r) + \frac{5}{2}\left(\sqrt{\mathfrak{R}_{T(\mathbb{S})}(\ell\circ\mathbb{H}_r)} + \sqrt{\frac{r}{2}}\right)\sqrt{\frac{\log 1/\delta}{m}} + \frac{25}{48}\frac{\log 1/\delta}{m},$$

entails the statement of the theorem 1. □

Our main result is the following theorems which bounds the generalization error of a function $h \in \mathbb{H}$ learned by minimizing $\tilde{R}_{T_\kappa(\mathbb{S}_\pi)}$.

Theorem 2 (a). *Let $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m \in (\mathbb{X}\times\mathbb{Y})^m$ be a dataset of m examples drawn i.i.d. according to a probability measure $\mathbb{D} = \mathbb{D}_\mathbb{Y} \times \mathbb{D}_{\mathbb{X}|\mathbb{Y}}$ over $\mathbb{X}\times\mathbb{Y}$ and $T(\mathbb{S})$ the transformed set obtained with the transformation function T defined in Eq.* (3.7). *Let $\mathbb{S}_\pi \in (\mathbb{X}\times\mathbb{Y})^n$ and $T_\kappa(\mathbb{S}_\pi)$, $|T_\kappa(\mathbb{S}_\pi)| = M$ be a training sets derived from $\mathbb{S}$ and $T(\mathbb{S})$ respectively using the algorithm* DS-mRb *with parameters $\pi_1,\ldots,\pi_K$ and κ. With the class of functions*

$\mathbb{G} = \{g : \mathbb{X} \times \mathbb{Y} \to \mathbb{R}\}$ *and* $\mathscr{H} = \{h : h(\phi(\mathbf{x}^y), \phi(\mathbf{x}^{y'})) = g(\mathbf{x}^y) - g(\mathbf{x}^{y'}), g \in \mathbb{G}\}$ *we have the following bound on the expected risk of the classifier :*

$$R(h) \leq \alpha \tilde{R}_{T_\kappa(\mathbb{S}_\pi)}(h) + \alpha \mathfrak{R}_{T_\kappa(\mathbb{S}_\pi)}(\ell \circ \mathbb{H}) + \alpha \sqrt{\frac{(K-1)\log 2/\delta}{2M\kappa}} + \sqrt{\frac{2\alpha \log 4K/\delta}{\beta(m-1)}} + \frac{7\beta \log 4K/\delta}{3(m-1)}.$$

holds with probability at least $1-\delta$*, for any* $\delta > 0$*, the for all* $h \in \mathbb{H}$*,* $\ell : \{-1, +1\} \times \mathbb{R} \to [0, 1]$ *is the* $0/1$ *loss, and*

$$\alpha = \max_{y : 1 \leq y \leq K} \eta_y / \pi_y, \quad \beta = \max_{y : 1 \leq y \leq K} 1/\pi_y,$$

and η_y *is strictly positive empirical probability of the class* y *over* $\mathbb{S}$*.*

Proof. By lemma 4 we have for $\bar{\mathbb{D}} = \bar{\mathbb{D}}_{\mathbb{Y}} \times \mathbb{D}_{\mathbb{X}|\mathbb{Y}}$, $\mathbb{P}_{\bar{\mathbb{D}}_{\mathbb{Y}}}[y(\mathbf{x}) = i] \propto \pi_i$, $1 \leq i \leq K$ with probability at least $1 - \delta/2$:

$$R(h) \leq \alpha \mathbb{E}_{\mathbf{x}^y \sim \bar{\mathbb{D}}} \tilde{R}_{T_\kappa(\mathbb{S}_\pi)}(h) + \sqrt{\frac{2\alpha \log 4K/\delta}{\beta(m-1)}} + \frac{7\beta \log 4K/\delta}{3(m-1)}.$$

By theorem 4 of [Usunier et al., 2005] we have with probability at least $1 - \delta/2$:

$$\mathbb{E}_{\mathbf{x}^y \sim \bar{\mathbb{D}}} \tilde{R}_{T_\kappa(\mathbb{S}_\pi)}(h) \leq \tilde{R}_{T_\kappa(\mathbb{S}_\pi)}(h) + \mathfrak{R}(\mathbb{H}) + \sqrt{\frac{\chi^*_{T_\kappa(\mathbb{S}_\pi)} \log 2/\delta}{2M\kappa}},$$

where a dependency graph for subsample $T_\kappa(\mathbb{S}_\pi)$ is a subgraph of the dependency graph for the whole sample $T(\mathscr{S})$.

Then by lemma 1 we have $\chi^*_{T_\kappa(\mathbb{S}_\pi)} \leq \chi^*_{T(\mathbb{S})} = K - 1$, the last is due to theorem 1 of [Joshi et al., 2015a], where $\chi^*_{T(\mathbb{S})}$ and $\chi^*_{T(\mathbb{S})}$ stand for the fractional chromatic number of the dependency graph for $T_\kappa(\mathbb{S}_\pi)$ and $T(\mathbb{S})$ resp. Gather together the last two equations we prove the theorem. □

Theorem 2 (b). *Let* $\mathbb{S} = (\mathbf{x}_i^{y_i})_{i=1}^m \in (\mathbb{X} \times \mathbb{Y})^m$ *be a dataset of* m *examples drawn i.i.d. according to a probability distribution* $\mathbb{D}$ *over* $\mathbb{X} \times \mathbb{Y}$ *and* $T(\mathbb{S}) = ((\boldsymbol{Z}_i, \tilde{y}_i))_{i=1}^N$ *the transformed set obtained with the transformation function* T *defined in Eq.* (3.7)*. Let* $\mathbb{S}_\pi \in (\mathbb{X} \times \mathbb{Y})^M$ *and* $T_\kappa(\mathbb{S}_\pi)$ *be a training set derived from* $T(\mathbb{S})$ *using the algorithm* `DS-mRb` *with parameters* $\pi_1, \ldots, \pi_K$ *and* κ*. With the class of functions* $\mathbb{G} = \{g : \mathbb{X} \times \mathbb{Y} \to \mathbb{R}\}$ *and* $\mathscr{H} = \{h : h(\phi(\mathbf{x}^y), \phi(\mathbf{x}^{y'})) = g(\mathbf{x}^y) - g(\mathbf{x}^{y'}), g \in \mathbb{G}\}$*, consider the parameterized family* $\mathbb{H}_r$ *which, for* $r > 0$*, is defined as :*

$$\mathbb{H}_r = \{h : h \in \mathbb{H}, \mathbb{V}[h] \doteq \mathbb{V}_{\boldsymbol{Z}, \tilde{y}}[1_{\tilde{y} h(\boldsymbol{Z})}] \leq r\},$$

where $\mathbb{V}$ *denotes the variance. Then for any* $\delta > 0$ *with probability at least* $(1-\delta)$ *the following generalization bound holds for all* $h \in \mathbb{H}_r$ *:*

$$R(h) \leq \alpha \tilde{R}_{T_\kappa(\mathbb{S}_\pi)}(h) + \alpha \mathfrak{R}_{T_\kappa(\mathbb{S}_\pi)}(\ell \circ \mathbb{H}_r) + \alpha \sqrt{\frac{\log 4/\delta}{2m}} + \sqrt{\frac{2\alpha \log 4K/\delta}{\beta(m-1)}} + \frac{7\beta \log 4K/\delta}{3(m-1)} + \frac{5\alpha}{2}\left(\sqrt{\mathfrak{R}_{T_\kappa(\mathbb{S}_\pi)}(\ell \circ \mathbb{H}_r)} + \sqrt{\frac{r}{2}}\right)\sqrt{\frac{(K-1)\log 2/\delta}{\kappa M}} + \frac{25\alpha}{48}\frac{\log 2/\delta}{M},$$

where $\ell : \{-1,+1\} \times \mathbb{R} \to [0,1]$ *is the* $0/1$ *loss and* $\mathfrak{R}(\mathbb{H}_r)$ *is the* Local Fractional Rademacher Complexity *defined as:*

$$\mathfrak{R}(\mathbb{H}_r) \doteq \frac{2}{N}\mathbb{E}_\xi\left[\sum_{k\in[K-1]} \omega_k \mathbb{E}_{\boldsymbol{Z}_{\mathbb{C}_k}}\left[\sup_{h\in\mathbb{H}_r} \sum_{\alpha\in\mathbb{C}_k(T(\underline{\mathbb{S}}))} \xi_\alpha h(\boldsymbol{Z}_\alpha)\right]\right]$$

with $\xi = (\xi_1, \ldots, \xi_N)$ *a sequence of* N *independent Rademacher variables such that* $\mathbb{P}(\xi_n = 1) = \mathbb{P}(\xi_n = -1) = 1/2$*, and* $\alpha = \max_{y:1\leq y\leq K} \eta_y/\pi_y$*,* $\beta = \max_{y:1\leq y\leq K} 1/\pi_y$*, and* $\eta_y > 0$ *is the empirical probability of the class* y *over* $\mathbb{S}$*.*

Proof. The proof of the theorem essentially combines the results of theorem 1 and lemma 4.

By lemma 4 we have with probability at least $1-\delta/2$:

$$R(h) \leq \alpha \mathbb{E}_{\mathbf{x}^y\sim\tilde{\mathbb{D}}} \tilde{R}_{T(\mathbf{x}^y)}(h) + \sqrt{\frac{2\alpha \log 4K/\delta}{\beta(m-1)}} + \frac{7\beta \log 4K/\delta}{3(m-1)}. \tag{3.19}$$

Lemma 2 (a) applied to $T_\kappa(\mathbb{S}_\pi)$, $T_\kappa(\mathbb{S}_\pi) = M\kappa$ gives with probability at least $1-\delta/2$:

$$\mathbb{E}_{\mathbf{x}^y\sim\tilde{\mathbb{D}}} \tilde{R}_{T(\mathbf{x}^y)}(h) \leq \tilde{R}_{T_\kappa(\mathbb{S}_\pi)}(h) + \mathfrak{R}_{T_\kappa(\mathbb{S}_\pi)}(\ell \circ \mathbb{H}_r) + \frac{5}{2}\left(\sqrt{\mathfrak{R}_{T_\kappa(\mathbb{S}_\pi)}(\ell \circ \mathbb{H}_r)} + \sqrt{\frac{r}{2}}\right)\sqrt{\frac{(K-1)\log 2/\delta}{M\kappa}} + \frac{25}{48}\frac{\log 2/\delta}{M} \tag{3.20}$$

Substitution (3.19) in (3.20) gives :

$$R(h) \leq \alpha \tilde{R}_{T_\kappa(\mathbb{S}_\mu)}(h) + \alpha \mathfrak{R}_{T_\kappa(\mathbb{S}_\mu)}(\ell \circ \mathbb{H}_r) + \sqrt{\frac{2\alpha \log 4K/\delta}{\beta(m-1)}} + \frac{7\beta \log 4K/\delta}{3(m-1)} + \frac{5\alpha}{2}\left(\sqrt{\mathfrak{R}_{T_\kappa(\mathbb{S}_\mu)}(\ell \circ \mathbb{H}_r)} + \sqrt{\frac{r}{2}}\right)\sqrt{\frac{(K-1)\log 2/\delta}{M\kappa}} + \frac{25\alpha}{48}\frac{\log 2/\delta}{M}$$

□

Proof of the theorem 2. The statement of theorem 2 in the paper is essentially a union of the statements of theorem 2 (a) and theorem 2 (b) proved above. □

The essence of the last theorem is improvement conditional error within classes with low prior probability, which in its turn improves macro $\mathtt{MaF}_1$-measure of the classifier.

3.4.3 LARGE-CLASS (EXTREME) CLASSIFICATION EXPERIMENTS USING DS-mRb

In this section, we provide an empirical evaluation of the proposed reduction approach with the DS-mRb sampling strategy for large-scale multi-class classification of document collections. First, we present the mapping $\phi : \mathbb{X} \times \mathbb{Y} \rightarrow \mathbb{R}^p$. Then, we provide a statistical and computational comparison of our method with state-of-the-art large-scale approaches on popular datasets.

3.4.3.1 Datasets

We evaluate the proposed method using popular datasets from the Large Scale Hierarchical Text Classification challenge (LSHTC) 1 and 2 [Partalas et al., 2015]. These datasets are provided in a pre-processed format using stop-word removal and stemming. Various characteristics of these datesets including the statistics of train, test and heldout are listed in Table 3.4. Since, the datasets used in LSHTC2 challenge were in multi-label format, we converted them to multi-class format by replicating the instances belonging to different class labels. Also, for the largest dataset (WIKI-large) used in LSHTC2 challenge, we used samples with 50,000 and 100,000 classes. The smaller dataset of LSHTC2 challenge is named as WIKI-Small, whereas the two 50K and 100K samples of large dataset are named as WIKI-50K and WIKI-100K in our result section.

Datasets	# of classes, K	Train Size	Test Size	Heldout Size	Dimension, d
LSHTC1	12294	126871	31718	5000	409774
DMOZ	27875	381149	95288	34506	594158
WIKI-Small	36504	796617	199155	5000	380078
WIKI-50K	50000	1102754	276939	5000	951558
WIKI-100K	100000	2195530	550133	5000	1271710

Table 3.4: Characteristics of the datasets used in our experiments

3.4.3.2 Baselines

We compare the proposed approach, denoted as the sampling strategy by DS-mRb, with popular baselines listed below:

- `OVA`: LibLinear [Fan et al., 2008a] implementation of one-vs-all SVM.
- `M-SVM`: LibLinear implementation of multi-class SVM proposed in [Crammer and Singer, 2002].
- `RecallTree` [Daume III et al., 2016]: A recent tree based multi-class classifier implemented in Vowpal Wabbit.
- `FastXML` [Prabhu and Varma, 2014]: An extreme multi-class classification method which performs partitioning in the feature space for faster prediction.
- `PfastReXML` [Jain et al., 2016]: Tree ensemble based extreme classifier for multi-class and multilabel problems.
- `PD-Sparse` [Yen et al., 2016]: A recent approach which uses multi-class loss with ℓ_1-regularization.

For methods `FastXML`, `PfastReXML` and `PD-Sparse` we used the solvers provided by the authors. Also referring to the work [Yen et al., 2016], we did not consider other recent methods `SLEEC` [Bhatia et al., 2015] and `LEML` [Yu et al., 2014a] in our experiments, since they have been shown to be consistently outperformed by the above mentioned state-of-the-art approaches.

3.4.3.3 Feature Representation

For our newly proposed method `DS-mRb`, we used mostly the same features as used for `mRb`. The only difference is the last two features. Instead of using the to nearest neighbors, we use centroid distance and BM25 as the last two features. The two additional features gave good promise for better result. The feature are summarized in Table 3.5.

3.4.3.4 Parameter Tuning

Each of these methods require tuning of various hyper-parameters that influence their performance. For each methods, we tuned the hyper-parameters over a heldout validation set and used the combination which gave best predictive performance. In the following section we will discuss the important hyper-parameters that we needed to tune.

Features in the joint example/class representation representation $\phi(\mathbf{x}^y)$.		
1. $\sum_{t \in y \cap \mathbf{x}} \log(1 + y_t)$	**2.** $\sum_{t \in y \cap \mathbf{x}} \log\left(1 + \frac{l_{\mathbb{S}}}{F_t}\right)$	**3.** $\sum_{t \in y \cap \mathbf{x}} I_t$
4. $\sum_{t \in y \cap \mathbf{x}} \frac{y_t}{\lvert y \rvert}.I_t$	**5.** $\sum_{t \in y \cap \mathbf{x}} \log\left(1 + \frac{y_t}{\lvert y \rvert}\right)$	**6.** $\sum_{t \in y \cap \mathbf{x}} \log\left(1 + \frac{y_t}{\lvert y \rvert}.I_t\right)$
7. $\sum_{t \in y \cap \mathbf{x}} \log\left(1 + \frac{y_t}{\lvert y \rvert}.\frac{l_{\mathbb{S}}}{F_t}\right)$	**8.** $\sum_{t \in y \cap \mathbf{x}} 1$	**9.** $d(\mathbf{x}^y, \texttt{centroid}(y))$
10. BM25 $= \sum_{t \in y \cap x} I_t \cdot \frac{2 \times y_t}{y_t + (0.25 + 0.75 \cdot \texttt{len}(y)/\texttt{avg}(\texttt{len}(y))}$		

Table 3.5: Joint example/class representation for text classification, where $t \in y \cap \mathbf{x}$ are terms that are present in both the class y's mega-document and document $\mathbf{x}$. Denote by $\mathcal{V}$ the set of distinct terms within $\mathbb{S}$ then $\mathbf{x}_t$ is the frequency of term t in $\mathbf{x}$, $y_t = \sum_{\mathbf{x} \in y} \mathbf{x}_t$, $|y| = \sum_{t \in \mathcal{V}} y_t$, $F_t = \sum_{\mathbf{x} \in \mathbb{S}} \mathbf{x}_t$, $l_{\mathbb{S}} = \sum_{t \in \mathcal{V}} \mathbb{S}_t$. Finally, I_t is the inverse document frequency of term t, $len(y)$ is the length (number of terms) of documents in class y, and $avg(len(y))$ is the average of document lengths for all the classes

- `OVA`, `M-SVM`: For both these methods chose SVM with linear kernel as the base classifier, since it was performing the best in our experiments as well as reported in another work [Yen et al., 2016]. Here, the parameter to be tuned for both these methods is the penalty term denoted as 'C'.

- `RecallTree`: For `RecallTree` method, we tuned four hyper-parameters: bit precision ("b"), learning rate (l), loss function type (loss_function) and number of passes over the training data (passes).

- `FastXML` and `PfastReXML`: For `FastXML` and `PfastReXML` methods the important hyper-parameters are: number of trees to be grown (t) and SVM weight co-efficient (c).

- `PD-Sparse`: For `PD-Sparse` method the hyper-parameters to be tuned are: L1-regularization weight weight (l) and training with or without hashing (multiTrain or multiTrainHash respectively). Here, it is important to note that the use of hashing causes lower memory usage but increases the training time significantly. However, for larger datasets it was impossible to train without hashing because of huge memory required by the model. Hence, in all our methods we used hashing while training.

- DS-mRb: For the proposed method, we first choose the average number of examples to be taken per class in the first sub-sampling. Then based on the probability distribution of each class we randomly pick examples from each class. Also we tune the number of adversarial classes (κ) and Candidate classes (q).

The list of used hyper-parameters for our final results are reported in the table 3.6.

Algorithm	Parameters	LSHTC1	DMOZ	WIKI-Small	WIKI-50K	WIKI-100K
OVA	C	10	10	1	NA	NA
M-SVM	C	1	1	NA	NA	NA
RecallTree	b	30	30	30	30	28
	l	1	0.7	0.7	0.5	0.5
	loss_function	Hinge	Hinge	Logistic	Hinge	Hinge
	passes	5	5	5	5	5
FastXML	t	100	50	50	100	50
	c	100	100	10	10	10
PfastReXML	t	50	50	100	200	100
	c	100	100	10	10	10
PD-Sparse	l	0.01	0.01	0.001	0.0001	0.01
	Hashing	multiTrainHash	multiTrainHash	multiTrainHash	multiTrainHash	multiTrainHash
DS-mRb	Examples per class in average*	5	5	2	2	2
	Adversarial Classes (κ)	122	27	36	5	10
	Candidate Classes (q)	10	10	10	10	10

* Here examples per class for proposed mRb method represents the average number of examples sampled per class. The examples are chosen at random from each class with probability π_k based on the distribution.

Table 3.6: Hyper-parameters used in the final experiments

3.4.3.5 Comparison Result:

The parameters of the datasets along with the results for compared methods are shown in Table 3.7. The results are provided in terms of train and predict times, total memory usage, and predictive performance measured with accuracy and macro F1-measure ($\mathtt{MaF}_1$). For better visualization and comparison, we plot the same results as bar plots in Fig. 3.6 keeping only the best five methods while comparing the total runtime and memory usage.

First, we observe that the tree based approaches (FastXML, PfastReXML and RecallTree) have worse predictive performance compared to the other methods. This is due to the fact that the prediction error made at the top-level of the tree cannot be corrected at lower levels, also known as cascading effect. Even though they have lower runtime and memory usage, they suffer from this side effect.

For large scale collections (**WIKI-Small**, **WIKI-50K** and **WIKI-100K**), the solvers with competitive predictive performance are OVA, M-SVM, PD-Sparse and DS-mRb. How-

Data		OVA	M-SVM	RecallTree	FastXML	PfastReXML	PD-Sparse	DS-mRb
LSHTC1	train time	23056s	48313s	701s	8564s	3912s	5105s	321s
m = 163589	predict time	328s	314s	21s	339s	164s	67s	544s
d = 409774	total memory	40.3G	40.3G	122M	470M	471M	10.5G	2G
K = 12294	Accuracy	44.1%	36.4%	18.1%	39.3%	39.8%	45.7%	37.4%
	$\texttt{MaF}_1$	27.4%	18.8%	3.8%	21.3%	22.4%	27.7%	26.5%
DMOZ	train time	180361s	212356s	2212s	14334s	15492s	63286s	1060s
m = 510943	predict time	2797s	3981s	47s	424s	505s	482s	2122s
d = 594158	total memory	131.9G	131.9G	256M	1339M	1242M	28.1G	5.3G
K = 27875	Accuracy	37.7%	32.2%	16.9%	33.4%	33.7%	40.8%	27.8%
	$\texttt{MaF}_1$	22.2%	14.3%	1.75%	15.1%	15.9%	22.7%	20.5%
WIKI-Small	train time	212438s	>4d	1610s	10646s	21702s	16309s	1290s
m = 1000772	predict time	2270s	NA	24s	453s	871s	382s	2577s
d = 380078	total memory	109.1G	109.1G	178M	949M	947M	12.4G	3.6G
K = 36504	Accuracy	15.6%	NA	7.9%	11.1%	12.1%	15.6%	21.5%
	$\texttt{MaF}_1$	8.8 %	NA	<1%	4.6%	5.63%	9.91%	13.3%
WIKI-50K	train time	NA	NA	4188s	30459s	48739s	41091s	3723s
m = 1384693	predict time	NA	NA	45s	1110s	2461s	790s	4083s
d = 951558	total memory	330G	330G	226M	1327M	1781M	35G	5G
K = 50000	Accuracy	NA	NA	17.9%	25.8%	27.3%	33.8%	33.4%
	$\texttt{MaF}_1$	NA	NA	5.5%	14.6%	16.3%	23.4%	24.5%
WIKI-100K	train time	NA	NA	8593s	42359s	73371s	155633s	9264s
m = 2750663	predict time	NA	NA	90s	1687s	3210s	3121s	20324s
d = 1271710	total memory	1017G	1017G	370M	2622M	2834M	40.3G	9.8G
K = 100000	Accuracy	NA	NA	8.4%	15%	16.1%	22.2%	25%
	$\texttt{MaF}_1$	NA	NA	1.4%	8%	9%	15.1%	17.8%

Table 3.7: Comparison of the result of various baselines in terms of time, memory, accuracy, and macro F1-measure

ever, standard OVA and M-SVM have a complexity that grows linearly with K thus the total runtime and memory usage explodes on those datasets, making them impossible to learn. For instance, on Wiki large dataset sample of 100K classes, the memory consumption of both approaches exceeds the Terabyte and they take several days to complete the training. Furthermore, on this data set and the second largest Wikipedia collection (**WIKI-50K** and **WIKI-100K**) the proposed approach is highly competitive in terms of Time, Total Memory and both performance measures comparatively to all the other approaches. These results suggest that the method least affected by long-tailed class distributions is DS-mRb, mainly because of two reasons: first, the sampling tends to make the training set balanced and second, the reduced binary dataset contains similar number of positive and negative examples. Hence, for the proposed approach, there is an improvement in both accuracy and $\texttt{MaF}_1$ measures.

The recent PD-Sparse method also enjoys a competitive predictive performance but it requires to store intermediary weight vectors during optimization which pre-

vents it from scaling well. The PD-Sparse solver provides an option for hashing leading to fewer memory usage during training which we used in the experiments; however, the memory usage is still significantly high for large datasets and at the same time this option slows down the training process considerably.

In overall, among the methods with competitive predictive performance, DS-mRb seems to present the best runtime and memory usage; its runtime is even competitive with most of tree-based methods, leading it to provide the best compromise among the compared methods over the three measures: time, memory and predictive performance.

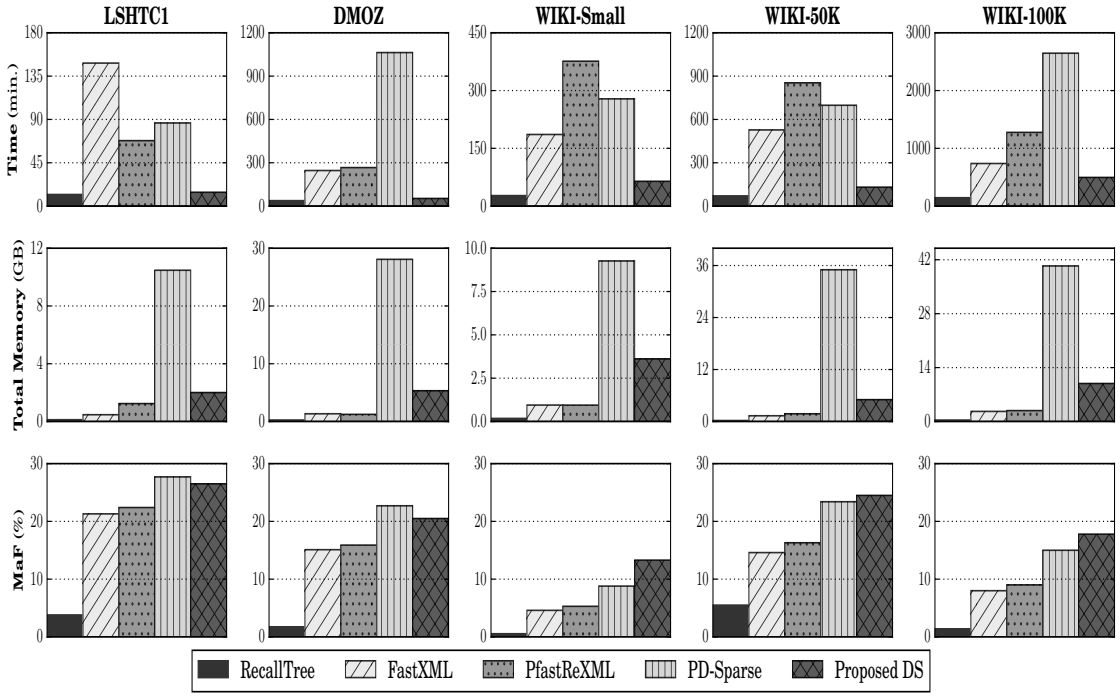


Figure 3.6: Comparisons in Total (Train and Test) Time (min.), Total Memory usage (GB), and $\mathtt{MaF}_1$ of the five best performing methods on **LSHTC1**, **DMOZ**, **WIKI-Small**, **WIKI-50K** and **WIKI-100K** datasets.

3.5 Closing Remarks

In this part of the thesis we presented a new approach to reduce a large-scale multi-class classification problem to equivalent binary classification problem by subtraction of pairwise joint representation of example, class pairs. First, we proposed a basic algorithm, which helps to overcome the challenges of class imbalance and large-scale classification. However, the reduction process introduces new challenges corresponding to the size of the new binary dataset and the time taken for the reduction process as well as the memory usage. To overcome these new challenges, we introduced an extended version of the algorithm referred to as `DS-mRb`. This extension of naive approach incorporates a double sampling approach during reduction and candidate selection during prediction phase. This helps to reduce the total runtime and memory usage of the algorithm, whereas the predictive performance remains the same as before. We also experimentally validated the effectiveness of the propsoed approach on popular datasets of text classification considering large class cases (up to 100000). The experiments were performed on 5 different datasets of different characteristics and sizes. We also present the comparison of result with respect to 6 other recent state-of-the-art approaches for large-scale multiclass classification. The results suggest that the propsoed approach, `DS-mRb` is the best performance compromise among all compared methods.

II Asynchronous Framework for Distributed Machine Learning

4 Distributed Machine Learning

4.1 Introduction

As the popularity of internet has increased over the last decade, the amount of available data has grown rapidly. So, machine learning algorithms need to be (re)designed to handle these large-scale datasets. Some of the common machine learning domains with such magnitude of data are binary classification and recommender systems. For example, in large-scale binary classification problems, the number of examples and the feature size can be upto the order of millions. Similarly, recommender system applications may involve users, items and ratings of the order of millions or even more. Handling this magnitude of data has become a prominent challenge in the machine learning community. Even if we are able to keep it in a single machine, running a machine learning algorithm on such huge datasets takes unacceptably large amount of time.

Perhaps the simplest strategy in such situations is to reduce the dataset by discarding many examples, also known as subsampling. However, this strategy can only be useful if the problem is simple enough. However, in most of the machine learning applications, subsampling significantly affects the quality of the machine learning model as we are throwing away useful information.

4.1.1 Distributed Algorithms

So, a better solution is to run the machine learning algorithms in a distributed manner simultaneously. Distributed algorithms can be divided into two groups:

1. Shared Memory or Parallel Algorithms : These algorithms make use of multiple cores within the same machine while keeping the entire dataset in the main memory. So, all the processors have access to the data and can perform the machine learning optimization simultaneously [Zinkevich et al., 2010, Recht et al.,

2011, Jaggi et al., 2014, Leblond et al., 2016, Zhao and Li, 2016]. However, one obvious drawback is that the size of datasets can be so huge that it might not fit in the memory of single machine.

2. Shared-Nothing or Distributed Algorithms: Another line of algorithms consider a fully distributed scenario [Dean et al., 2012, Xu and Yin, 2014, Chang et al., 2015, Zhang et al., 2015, Huo and Huang, 2016], where the individual machines has its private memory which cannot be directly accessed by another machine. They are suitable for many industry scale applications, since datasets are usually collected and stored in a decentralized manner using a cluster. In such cases, it is a tedious task to move data from different machines to a single machine. In this work, we also consider the fully distributed scenario and will refer it simply with the term "distributed" throughout the rest of the thesis.

4.1.2 Desired Properties of Distributed System

In the distributed setting, information needs to be communicated over the network bandwidth, which is a limited resource. Hence, communication cost is one of the most important considerations for distributed frameworks. Moreover, it is also important to make sure that each machine runs reliably without failure, especially when the workload is increased significantly. A good distributed framework should be able to address these challenges inherent in distributed environments. Below we will discuss the main desired properties of a distributed computing system [Li, 2017]:

1. Efficiency: Distributed computing systems should incur least communication cost while making an optimal usage of the computing resources. In the distributed environments the available network bandwith is very limited and has to be shared by several machines for exchanging information. For example, the memory bandwith of a personal computer is around 400 Gbit/sec, whereas the network bandwidth on Amazon AWS is just 10 Gbit/sec [Li, 2017]. Moreover, the communication latency in such systems are significantly worse. For instance, the latency for accessing the main memory in single machine is around 100 ns, whereas in data centers it is around 0.1 ms to 1 ms.

   Also, the machines in distributed systems usually have different computing power and are running different workloads. Hence, some of them are significantly

faster/slower than the others. So, the distributed algorithms relying on synchronization between the machines do not perform well, since the slower machines become the performance bottleneck of the entire system. Hence, while designing the distributed algorithms such synchronization between the machines should be avoided.

2. Fault Tolerance: One common problem in distributed computing environments is that a machine can fail during computation. Such failures occur more in distributed systems comprising of large number of machines and handling large amount of data. Hence, a good distributed system should be robust to such failures. Many of the advanced distributed frameworks such as Spark [1], OpenMPI [2], MPICH [3] have specialized features for fault tolerance.

3. Ease of use: The programming interface of a distributed system should be simple and flexible. It should provide an interface which hides the implementation details, while being flexible enough to implement several different algorithms.

4.1.3 DISTRIBUTED FRAMEWORKS

In this age of big data, there are many programming frameworks available to devise distributed algorithms in parallel and distributed environments. Here we will review two of the widely recognized frameworks for distributed system: Message Passing Interface (MPI) [Snir, 1998] and MapReduce [Dean and Ghemawat, 2008].

1. MPI: is a message passing library specification utilizing a message passing model for parallel and distributed environments [Snir, 1998]. It is not a programming implementation by itself. There are several available implementation of MPI specification such as OpenMPI, MPICH and GridMPI [Diaz et al., 2012].
   MPI provides point-to-point, collective, one-side, and parallel I/O models for communication. Point-to-point communication allows the exchange of information between to communicating processes, whereas a collective communication refers to the broadcast of message from one process to many processes. MPI also allows message passing in various modes such as blocking and non-blocking communication. It can be used in various platforms such as Linux, OS

[1] https://spark.apache.org/
[2] https://www.open-mpi.org/
[3] https://www.mpich.org/

X, Windows, Solaris etc. It works with different file systems such as NFS, HDFS etc. The main advantage of using MPI is its flexibility of programming. In MPI, the programmer has full control over the framework, hence it can be used to devise complex architectures. Moreover, MPI supports both synchronous and asynchronous communication.

2. MapReduce: On the other hand, MapReduce is the programming paradigm used by Hadoop framework, popularly referred as the big data processing framework. Hadoop clusters comprises of thousands of commodity machines and a distributed file system called HDFS. MapReduce organizes the application as Map and Reduce pairs [Kang et al., 2015]. Normally the data read and write operations are done with HDFS. In such frameworks, programmers do not have to worry about data partitioning, process creation and synchronization. So, the main advantage of using MapReduce paradigm is its ease of use, as most of the tasks are performed behind the scene by the framework itself. Moreover, these frameworks have a better fault tolerance mechanism. However, in contrast to MPI, the downsides of MapReduce paradigm is the lack of flexibility for programmers. One recent framework, following MapReduce paradigm and running on top of Hadoop clusters is Spark. It is an open source processing engine adopted by enterprises across wide range of industries. The main advantages of using Spark over existing Hadoop MapReduce is their speed and advanced ability of fault tolerance. As opposed to existing Hadoop frameworks, Spark uses an in-memory model for computation. Hence, they are several magnitude faster than Hadoop frameworks which read and write data to and from HDFS file system.

4.2 Problem Formulation

We consider a fully distributed scenario where training sets are stored over M connected machines. Such applications have attracted much interest in both machine learning and optimization communities. In this work, we consider the minimization of a loss function which can be represented as the sum of smooth functions.

Here, depending on the application, this minimization objective can have different forms. We will list the three possible cases:

- Case 1:

$$\mathscr{L}(\mathbf{v},\mathbf{w}) = \sum_{i=1}^{M} \mathscr{L}_i(\mathbf{v}_i,\mathbf{w}) \tag{4.1}$$

  where $\mathbf{v} = (\mathbf{v}_1,..,\mathbf{v}_M)$. In this model, each loss function $\mathscr{L}_i$ depends on (i) a *local* version of parameter $\mathbf{v}$, i.e. $\mathbf{v}_i$, that does not need to be exchanged across different machines, and (ii) a *shared* parameter $\mathbf{w}$ that has to be exchanged.

- Case 2:

  This is the case where each loss $\mathscr{L}_i$, depends only on *local* versions of parameter $\mathbf{v}$, the learning problem reduces as shown below. This is a totally parallel scenario that can be solved locally on each machine in parallel.

$$\mathscr{L}(\mathbf{v}) = \sum_{i=1}^{M} \mathscr{L}_i(\mathbf{v}_i) \tag{4.2}$$

- Case 3:

  The other extreme is a more typical case where each loss $\mathscr{L}_i$, depends only on the global shared parameter $\mathbf{w}$ and the learning problem in this case reduces to:

$$\mathscr{L}(\mathbf{w}) = \sum_{i=1}^{M} \mathscr{L}_i(\mathbf{w}) \tag{4.3}$$

  This kind of problem is extremely common in ML when one wants to find the best predictor from a dataset split in several batches.

In this work, we will consider two applications, which will represent the two distributed scenarios shown above. First, we will consider distributed matrix factorization problem with Stochastic Gradient Descent (SGD) based optimization. This corresponds to Case 1 above. We will notice that in this problem, we need to update two parameters out of which one is totally local to machines, whereas the other one needs to be shared among the machines. Second, we will consider the problem of binary classification which corresponds to Case 3. In this problem, each machines locally update a parameter vector which needs to be shared among all the machines periodically.

4.3 Asynchronous Distributed Strategy

In this section, we present our proposed asynchronous distributed approach by first describing the deduced learning strategy. We then provide a consistency justification in the form of a convergence proof.

4.3.1 Description

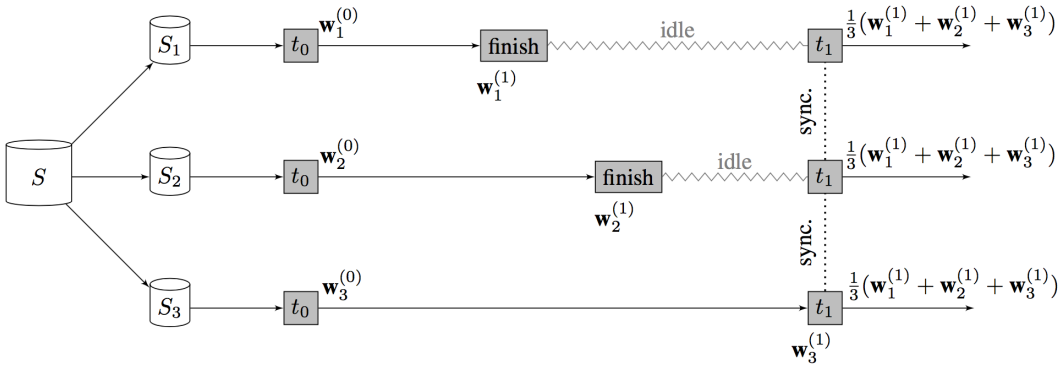
The main challenge of distributed learning is to effectively partition the data into computing nodes, and efficiently perform communication between them. Indeed, in the synchronous case, the slowest node becomes the bottleneck of the whole system and a potentially large amount of computational time is lost (Figure 4.1 (a)).

The main idea of our approach is that when a machine finishes an iteration over the subpart of the data it contains, it broadcasts its updated parameter values to the master node; which gathers the received parameter values from the workers (if any, and taking only the last one if multiple parameter values are received from one machine); and updates the parameter vector with the received updates. Then the updated parameter is broadcasted to worker nodes. In this way each computing node runs its iterations independently and gets rid of the synchronization bottleneck. Faster machines will perform their epochs faster, whereas the slower ones will be lagging on time but after finishing each epoch they will receive the most updated parameters from the master. This situation is depicted in Figure 4.1 (b).

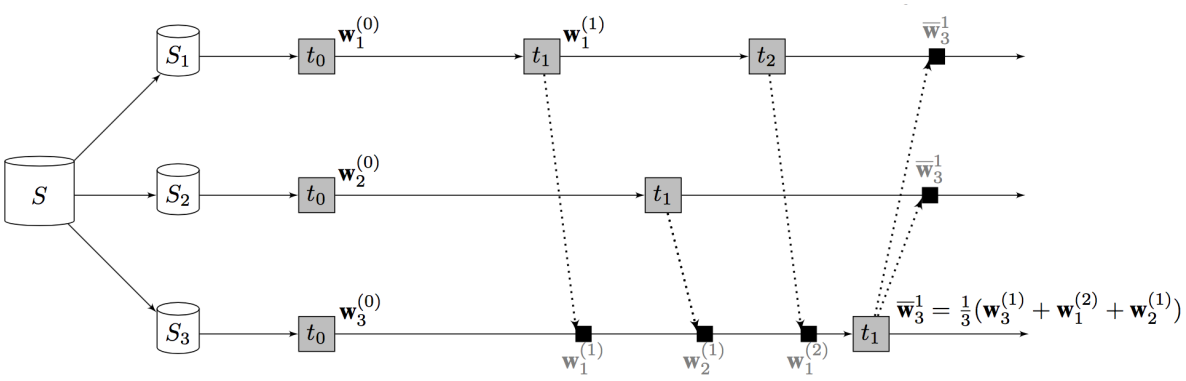
The main difference with other distributed asynchronous algorithms proposed in the literature [Zhang et al., 2015, Huo and Huang, 2016], our approach does not exchange gradients but rather parameter values updated after one complete pass over local subpart of the data. Although these quantities have the same sizes, we show that broadcasting of parameters performs better in practice, since they are exchanged after each epoch, whereas gradients need to be exchanged after every mini-batch update.

4.3.2 Consistency justification

In the case where the training data is partitioned into M batches $\{\mathcal{S}_1, \ldots, \mathcal{S}_M\}$, one for each computing machine, in the *shared parameter* case, the objective Eq. (4.3) can be



(a) Synchronous Framework



(b) Asynchronous Framework

Figure 4.1: Diagrams of the distributed synchronous (a) and asynchronous (b) frameworks.

rewritten as

$$\mathscr{L}(\mathbf{w}) = \sum_{i=1}^{M} \mathscr{L}_i(\mathbf{w}). \tag{4.4}$$

Here we may take advantage of the differentiability of $(\mathscr{L}_i)_{i=1}^M$ and use a gradient algorithm to find a minimizer of the global objective, $\mathscr{L}$. With a fixed stepsize gradient as an elementary operation before exchanging, we make the following assumptions :

Assumption 1 (on the functions)**.**

a. The objective function, $\mathscr{L}$, has a unique minimizer $\mathbf{w}^\star$;

b. Each $\mathscr{L}_i$ is differentiable and $\nabla\mathscr{L}_i$ is $\frac{1}{L}$-cococercive, that is $\forall \mathbf{w}, \mathbf{w}' \in \mathbb{R}^d$:

$$\langle \mathbf{w} - \mathbf{w}'; \nabla\mathscr{L}_i(\mathbf{w}) - \nabla\mathscr{L}_i(\mathbf{w}')\rangle \geq \frac{1}{L}\|\nabla\mathscr{L}_i(\mathbf{w}) - \nabla\mathscr{L}_i(\mathbf{w}')\|^2.$$

As a consequence of the Baillon-Haddad theorem (Th. 18.15 in [Bauschke and Combettes, 2011]); Assumption 1 (b) is notably verified whenever all functions $\mathscr{L}_i$ are convex and L_i-smooth, that is differentiable with an L_i-Lipschitz continuous gradient

Asynchronous Distributed Gradient update rule

When machine i finishes computing $\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k})$

(**Local step**) at i: $\mathbf{w}_i^{k+1} = \overline{\mathbf{w}}^{k-d_i^k} - \gamma\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k})$

(**Master step**) $\overline{\mathbf{w}}^{k+1} = \frac{1}{M}\sum_{j=1}^{M}\mathbf{w}_j^{k+1}$

Broadcast $\overline{\mathbf{w}}^{k+1}$

---

with $L = \max_i L_i$. Also, if a function $\mathscr{L}_i$ is L_i-smooth but not necessarily convex, then, considering $g_i = \mathscr{L}_i + \lambda/2\|\cdot\|^2$, it comes that ∇g_i is $1/(2\lambda)$ cocoercive for $\lambda > L$ (see Prop. 2 in [Zhu and Marcotte, 1996]). In our case, this means that if the (smooth) cost function is non-convex, then one can add a ℓ_2 regularization term so that the sum function verifies the sought property.

In Assumption 2, we also make the rather mild assumption that the delays are bounded, meaning that no machine is infinitely slower than the others. More precisely, we consider that the duration of its computation is bounded by D in the sense that if machine i finishes its computation at time $k+1$, then the value of the averaged parameter it used is at most D ticks old. Mathematically, denoting the computation delay for machine i at time k by d_i^k, our bounded delay assumptions means that when machine i finishes, say at time k, the (outdated) value of the averaged parameter it used is $\overline{\mathbf{w}}^{k-d_i^k}$ with $d_i^k \leq D$.

Assumption 2 (on the algorithm)**.** *The delays are uniformly bounded, i.e. there is $D < \infty$ such that for any machine i and iteration k; the delay $d_i^k \leq D$.*

The proposed Asynchronous Distributed update rule, corresponding to Figure 4.1 (b), is summarized in the pseudo-code in the right. In the **local step**, all machines including the master update their parameters; and in the **master step**, once the master finishes its update, it broadcasts the aggregated parameters (from the latest received ones) to all workers. Furthermore, using a gradient step as an elementary operation, the convergence of the algorithm can be proven with the attractive properties that the considered stepsizes can be chosen fixed, as in the standard gradient algorithm, and thus do not decay or depend on the delay; and that no assumptions are made on the distribution of the delays.

Theorem 1 (Convergence)**.** *Suppose that Assumptions 1 and 2 hold. Let $\gamma \in]0, 2/L[$. Then the sequence $(\overline{\mathbf{w}}^k)_k$ produced by our* Asynchronous Distributed Gradient update rule *converges to $\boldsymbol{w}^\star$.*

Proof. From Assumption 1 (i), $\mathbf{w}^\star$ is the unique minimizer of $\mathscr{L}$ and $\nabla\mathscr{L}(\mathbf{w}^\star) = \sum_{i=1}^M \nabla\mathscr{L}_i(\mathbf{w}^\star) = 0$. Let us define for all $i = 1,..,M$ $\mathbf{w}_i^\star = \mathbf{w}^\star - \gamma\nabla\mathscr{L}_i(\mathbf{w}^\star)$. Then at time k for the updating machine i, it comes from the cocoercivity of $\nabla\mathscr{L}_i$, Assumption 1 (b); and the definition $\mathbf{w}_i^{k+1} = \overline{\mathbf{w}}^{k-d_i^k} - \gamma\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k})$:

$$\begin{aligned}
\left\|\mathbf{w}_i^{k+1} - \mathbf{w}_i^\star\right\|^2 &= \left\|\overline{\mathbf{w}}^{k-d_i^k} - \gamma\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - (\mathbf{w}^\star - \gamma\nabla\mathscr{L}_i(\mathbf{w}^\star))\right\|^2 \\
&\leq \left\|\overline{\mathbf{w}}^{k-d_i^k} - \mathbf{w}^\star\right\|^2 + \gamma^2\left\|\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathscr{L}_i(\mathbf{w}^\star)\right\|^2 - \frac{2\gamma}{L}\left\|\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathscr{L}_i(\mathbf{w}^\star)\right\|^2.
\end{aligned}$$

Now by setting $\delta = \gamma\left(\frac{2}{L} - \gamma\right) > 0$ we get:

$$\begin{aligned}
\left\|\mathbf{w}_i^{k+1} - \mathbf{w}_i^\star\right\|^2 &\leq \left\|\overline{\mathbf{w}}^{k-d_i^k} - \mathbf{w}^\star\right\|^2 - \delta\left\|\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathscr{L}_i(\mathbf{w}^\star)\right\|^2 \\
&= \left\|\frac{1}{M}\sum_{j=1}^M(\mathbf{w}_j^{k-d_i^k} - \mathbf{w}_j^\star)\right\|^2 - \delta\left\|\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathscr{L}_i(\mathbf{w}^\star)\right\|^2 \\
&\leq \frac{1}{M}\sum_{j=1}^M\left\|\mathbf{w}_j^{k-d_i^k} - \mathbf{w}_j^\star\right\|^2 - \delta\left\|\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathscr{L}_i(\mathbf{w}^\star)\right\|^2,
\end{aligned}$$

where we used the fact that

$$\sum_{j=1}^M \mathbf{w}_j^\star = \sum_{j=1}^M \mathbf{w}^\star - \gamma\sum_{j=1}^M \nabla\mathscr{L}_j(\mathbf{w}^\star) = M\mathbf{w}^\star.$$

As the gradient of the objective $\nabla\mathscr{L}(\mathbf{w}) = \sum_{j=1}^M \nabla\mathscr{L}_j(\mathbf{w})$ is null at $\mathbf{w}^\star$. The last inequality is due to the convexity of the squared norm. For all other $j \neq i$, $\left\|\mathbf{w}_j^{k+1} - \mathbf{w}_j^\star\right\|^2 = \left\|\mathbf{w}_j^k - \mathbf{w}_j^\star\right\|^2$.

Let $\mathbf{y}_d^k = (\left\|\mathbf{w}_i^{k-d} - \mathbf{w}_i^\star\right\|^2)_{i=1,..,M}$ be the size-M vector of the individual errors at time $k-d$; and let $\mathbf{y}^k$ be the size-$M(D+1)$ vector obtained by concatenating the $(\mathbf{y}_d^k)_{d=0,..,D}$. From $\mathbf{y}^k$ to $\mathbf{y}^{k+1}$, we have that i) the last M values, $\mathbf{y}_D^k$, are dropped as they cannot intervene as D is the maximal delay; ii) the other ones are moved M coordinates lower $\mathbf{y}_{d+1}^{k+1} = \mathbf{y}_d^k$ for $d = 0,..,D-1$; iii) for the first M coordinates, they are copied from time k, $\mathbf{y}_0^{k+1} = \mathbf{y}_0^k$, except for the i-th one which verifies $\|\mathbf{w}_i^{k+1} - \mathbf{w}_i^\star\|^2 \leq \frac{1}{M}\sum_{j=1}^M \|\mathbf{w}_j^{k-d_i^k} - \mathbf{w}_j^\star\|^2$ thus $\mathbf{y}_0^{k+1}(i) \leq \frac{1}{M}\sum_{j=1}^M \mathbf{y}_{d_i^k}^k(j)$. Thus one can write $\mathbf{y}^{k+1} \preceq A^{k+1}\mathbf{y}^k$ where '$\preceq$' indicates the elementwise inequality and A^{k+1} represents the linear (in)-equalities mentioned above. A^{k+1}, seen as a $(D+1)\times(D+1)$ block matrix has identities on its sub-diagonal, and the top left block is the identity except for line i which has $1/M$ coefficients on the M columns corresponding to d_i^k. One can notice that it is non-negative and the row sum is constant equal to 1.

Taking the ℓ_∞-norm, we have $\|\mathbf{y}^{k+1}\|_\infty \leq \|A^{k+1}\mathbf{y}^k\|_\infty \leq \|A^{k+1}\|_\infty\|\mathbf{y}^k\|_\infty \leq \|\mathbf{y}^k\|_\infty$ as the ℓ_∞-induced matrix $\|\cdot\|_\infty$ is the maximal row sum and all rows of non-negative

matrix A^{k+1} have unit sum. This means that $(\|\mathbf{y}^k\|_\infty)_k$ is a converging sequence, say to some value α. Now, suppose that there is some coordinate that is strictly lower than α, then it cannot be equal to α or greater anymore due to the above inequality; this means, that as the communication time is bounded, any coordinate holding the value α will have to (strictly) decrease due to the averaging with the strictly lower coordinate, which contradicts α being the limit of sequence $(\|\mathbf{y}^k\|_\infty)_k$. Thus, all errors converge to the same value which means that $\|\nabla\mathscr{L}_i(\overline{\mathbf{w}}^{k-d_i^k}) - \nabla\mathscr{L}_i(\mathbf{w}^\star)\|^2 \to 0$, implying that all $\mathbf{w}_i^k$ and thus $\overline{\mathbf{w}}^k$ converge. Furthermore, all limits points of $\overline{\mathbf{w}}^k$ null the gradient of $\mathscr{L}$; $\mathbf{w}^\star$ being unique (Assumption 1 (i)), the convergence ensues. □

One can notice that using this asynchronous framework, the machines local parameters all converge to different values while their sum converge to the sought minimizer. As this sum is received after each iteration, the agents also have individual knowledge of the full minimizer. Finally, the tools used in this proof make it adaptable to a wide range of elementary operations verifying cocoercive contraction properties. For instance, if the loss has a smooth and a non-smooth part, the gradient step can be replaced by a proximal gradient step. Other possible extensions here include the Alternating Direction Method of Multipliers (ADMM) and Primal-Dual algorithms.

4.4 Closing Remarks

In this chapter, we introduced distributed machine learning. We began by introducing distributed algorithms in two different kinds of settings: shared-memory and shared-nothing. Then, we discussed the mainly desired properties of distributed frameworks followed by popularly used frameworks. In Section 4.2, we formulated the problem, which we are going to use throughout this part of the thesis. In Section 4.3, we presented our framework for asynchronous distributed machine learning based on averaging of parameters and showed the proof of convergence in this setting. Now, in the next two chapters, we will use the proposed asynchronous distributed frameworks for the optimization in two different applications: matrix factorization for recommender systems and large-scale binary classification.

5 Application 1: Distributed Matrix Factorization for Recommender Systems

5.1 Recommender Systems

Recommender Systems (RS) represent an active area of research in data mining due to large industrial potential. The main aim of Recommender Systems is to provide a personalized recommendation of an online product or service to the users, who are usually overloaded with the available information. So, we can also define it as an information filtering system on the web. Hence, the ultimate goal of the Recommender systems is to improve the customer relationship management as well as the revenue from the industrial viewpoint. RS recommends suitable items to users by predicting the user's interest in an item based on the information about the users, items or their interactions [Bobadilla et al., 2013]. The main feature of RS is to "guess" user's preferences and interests by analyzing information related to the user and/or other users to provide them with personalized recommendation [Resnick and Varian, 1997]. Some of the popular examples of use of RS in the industry are: movie recommendation by Netflix, song recommendation by Pandora and spotify, product recommendation by Amazon, job recommendation by LinkedIn, content recommendation by Facebook, quora etc.

5.1.1 Formal Definition

Formally, a recommendation problem involves estimating the ratings for the items that has not been seen or rated by a particular user [Adomavicius and Tuzhilin, 2005]. The prediction is performed based on the user's interaction with other items or some meta

information related to the users or items.

Hence, in this user-item context, a recommendation problem can be formulated as follows. Let U and I denote set of users and items respectively. In modern applications, the set U and I are very huge, millions or billions in most cases. Recommendation system takes these two sets of users and additionally the partial ratings given by some of the users to some of the items. Usually the number of ratings are very few. The recommendation problem can be divided into two sub-problems:

1. Finding the unknown ratings
2. Sorting the ratings to provide top-k recommendation

Here, the second sub-problem is a sorting problem. Whereas the first sub-problem carries more importance. The problem of estimating the ratings can be considered as the problem of estimating a utility function. Let f be an utility function which outputs an item's importance corresponding to a user. Hence, the recommendation problem can be defined as finding a subset $i^{'} \in I$ of items to be recommended for each user $u \in U$ that maximizes the utility function f. Mathematically:

$$\forall u \in U, i^{'} = \text{argmax}_{i \in I} f(u, i) \tag{5.1}$$

In the context of RS, user's rating for the items is represented as a matrix known as rating matrix. In this matrix the rows represent the uses and the columns represent each of the items. Each cell of the matrix contains the corresponding user's rating for that item. Figure 5.1 presents a sample rating matrix, where the non-zero values indicate user's rating for items and zeros represent unknown ratings. As already discussed the main aim of the RS is to estimate those unknown values in the rating matrix. As can be seen in the figure, user's provide rating for very few items. Hence, most of the values in the rating matrix are unknown which makes the matrix sparse. In most of the real cases, the sparsity of the rating matrix can be upto 99 %.

5.1.2 Types of Recommender System Models

Broadly, RS methods can be divided into two main groups:

- Content Based Recommendation
- Collaborative Recommendation

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0	3	0	3	0
User 2	4	0	0	2	0
User 3	0	0	3	0	0
User 4	3	0	4	0	3
User 5	4	3	0	4	0

Figure 5.1: User-Rating Matrix

Source: http://katbailey.github.io/post/matrix-factorization-with-tensorflow/

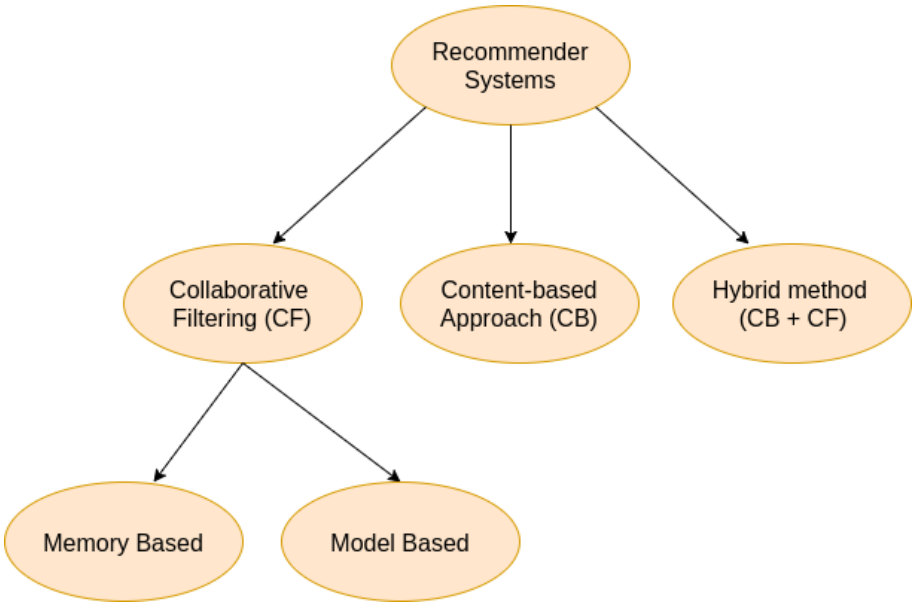


Figure 5.2: Types of Recommender Systems

5.1.2.1 Content Based Recommendation (CB)

In CB methods, items are recommended to a user, which are similar to the items user has preferred in the past [Pazzani and Billsus, 2007]. Here the notion of similarity is

derived from the metadata of the items. Based on the user's preference in the past, a list of attributes are derived known as the user-profile. Similarly, an item profile is created using the information about the item such as item descriptions or features. Hence, the recommendations are done by matching user and item profiles. For example a movie profile could include attributes such as: genre, participating actors, popularity etc and user profile might include attributes such as: demographic information or user's response to some questionnaire. The CB programs hence try to match the users and items based on the similar attributes in their profiles [Koren et al., 2009].

According to [Adomavicius and Tuzhilin, 2005], the main limitations of Content-Based RS are as follows:

- Content Scarcity: In real-case scenarios it is observed that obtaining the explicit attributes about the users and items is difficult. In many situations, these information are very scarce. Hence, matching users and items using such limited content directly affects the performance of the recommender system.

- Over-specialization: Another issue with these methods is that they tend to overfit the user's behavior based on the past preference. Hence, the users are always recommended with similar items everytime. This can be reduced by introducing randomness in the user/item profiles.

- New user problem: This is a common problem in most of the RS methods, commonly known as cold-start problem. This problem arises because of the lack of information about a user's past rating.

5.1.2.2 Collaborative-Filtering Based Recommendation (CF)

These methods identify user-item associations by analyzing the relationships between users and interdependencies among items. These methods rely on user's previous ratings to estimate unknown ratings without requiring to create explicit profiles [Koren et al., 2009]. In practice they are more accurate than the CB methods [Koren et al., 2009].

There are two types of CF-based methods used primarily in RS: memory-based methods and model based methods. Memory based methods use user rating data to compute similarity between users or items. Users are recommended new items based on the similarity. These approaches are effective and easy to implement. However, computing user/item similarity is a tedious task for large RS datasets. Model based

approches are one of the most successful approaches of RS and implemented in most of the industries. The main idea of these approaches is to learn a model from the rating data and estimate unknown ratings using the model. Most popular example of model based RS is latent factor models. These methods try to estimate the ratings by characterizing both users and items with low-dimensional factors inferred from the rating patterns.

The main limitations of CF methods includ:

- New user problem (cold-start): This is the problem same as for CB methods, arising while recommending items to a new user.
- New item problem: CF relies on ratings of similar users on an item. But if an item is not rated by enough users, then the recommendation results can be very biased.
- Sparsity: Huge sparsity in rating matrix is another prominent problem of CF methods. Since, its difficult to calculate similarity in the presence of sparsity and the models tend to overfit in such data.

5.2 Matrix Factorization for Recommender System

The most successful approach to realize latent factor model in practice is matrix factorization [Koren et al., 2009]. Matrix factorization for collaborative filtering captured much attention, especially after the Netflix prize [Koren et al., 2009]. The premise behind this approach is to approximate a large rating matrix R with the multiplication of two low-dimensional factor matrices P and Q, i.e. $R \approx \hat{R} = PQ^\top$ that model respectively users and items in the same latent space. Hence the interaction between users and items are modeled as the inner product in the latent space. These latent factors are supposed to decode hidden information which defines user's interest for items. This model is closely related to a Singular Value Decomposition (SVD), which is a popular factorization method in linear algebra. We cannot apply SVD to collaborative filtering because of the sparsity of the rating matrix, since conventional SVD is not defined if the matrix is not complete and anyway the complexity of computing SVD in large dimension is prohibitive.

5.2.1 Loss Function

For a pair of user and item (u, i) for which a rating r_{ui} exists, the corresponding instantaneous loss is defined as ℓ_2-regularized quadratic error:

$$\ell(P, Q, u, i) = \left(r_{ui} - q_i^\top p_u\right)^2 + \lambda(||p_u||^2 + ||q_i||^2), \tag{5.2}$$

where p_u (resp. q_i) is u-th line of P (resp. i-th line of Q) and $\lambda \geq 0$ is a regularization parameter. The global objective is hence :

$$\mathscr{L}(P, Q) = \sum_{(u,i):r_{ui}\text{exists}} \ell(P, Q, u, i). \tag{5.3}$$

Note that instantaneous error $\ell(P, Q, u, i)$ depends only on P and Q through p_u and q_i; however, item i may also be rated by user u' so that the optimal factor q_i depends on both p_u and $p_{u'}$.

5.2.2 Learning Algorithms

Two algorithms are popularly used to minimize the loss function of 5.2.

5.2.2.1 Alternating Least Squares (ALS)

Equation 5.2 consists of two unknowns p_u and q_i making the error function non-convex. However, we can fix one of the two unknowns and make the optimization problem quadratic, for which an optimal solution can be obtained. Hence, in ALS method, we alternately fix one variable and solve the least square problem for the other variable. This iterative process ensures the convergence of overall problem. In the following we will summarize the strong and weak points of this algorithm.

Strong points:

- This changes the originally non-convex problem to a convex (quadratic) problem, for which a closed-form solution can be obtained.
- It is easy to parallelize ALS [Koren et al., 2009] as updating individual rows of one of P or Q while fixing the other can be done simultaneously.

Weak points:

- Even though they are good for parallel applications, it becomes a challenge when the size of P or Q becomes large to be fit in the memory of a single machine.

- Per iteration convergence speed of ALS is slower as compared to SGD (which we will discuss next).
- They are not trivial to implement and their predictive performance is not as good as SGD based optimization.

5.2.2.2 Stochastic Gradient Descent (SGD)

SGD is a popular optimization algorithm in Machine Learning [Bottou, 2010], and it has also been shown to be effective for matrix factorization [Koren et al., 2009]. It has gained popularity, especially after being the winning solution for the Netflix prize competition [1].

In this case, the approach proceeds as follows: at each iteration k,

- select a user/item pair (u^k, i^k) for which a rating exists;
- perform a gradient step on $\ell(P, Q, u^k, i^k)$.

Algorithm 5 presents the algorithmic steps of SGD for matrix factorization. Here, m, n and k denote the total users, items and dimension of latent space simultaneously.

Input: A training set $\mathbb{R}^{m \times n}$, initial values $\mathbb{P}^{m \times k}$ and $\mathbb{Q}^{n \times k}$

while *not converged* **do**

  Select a training point (u,i) $\in$ R uniformly at random

  $P'_{u*} \leftarrow P_{u*} - \alpha \frac{\partial}{\partial P_{u*}} \ell(R_{ui}, P_{u*}, Q_{*i})$

  $Q_{*i} \leftarrow Q_{*i} - \alpha \frac{\partial}{\partial Q_{*i}} \ell(R_{ui}, P_{u*}, Q_{*i})$

  $P'_{u*} \leftarrow P_{u*}$

end

Algorithm 5: SGD for Matrix Factorization

Here stochasticity is used in the sense that the gradient on $\ell(P, Q, u^k, i^k)$ can be seen as an approximation of the gradient on an underlying global model but the choice of the considered users/items may or may not be random depending on the algorithm. Here, we will summarize the main strong and weak points of this algorithm.

Strong points:

- Ease of implementation.

[1] https://en.wikipedia.org/wiki/Netflix_Prize

- Better predictive performance and convergence speed.

Weak points:

- The updates of SGD are inherently sequential, hence its not straightforward to parallelize it. Moreover, the traditional convergence analysis is based on this assumption of sequential updates.

- Another drawback of a straightforward parallel implementation is that updates on factor matrices might not be independent. For example, for training points that lie on same rows (i.e. ratings corresponding to the same users), an SGD step modifies the same corresponding rows in factor matrix P; thus, these points cannot be learnt over in parallel and efficient communication between the computing nodes is necessary to synchronize the updates on factor matrices.

5.2.3 Matrix Factorization with User and Item Based Regularization

From the literature, we noticed that both memory and model based methods have their strong and weak points. Both type of methods rely on different types of information to enhance the RS performance. There is not a single method which acts flawlessly. Hence, we try to incorporate the benefits of Neighborhood based method in matrix factorization method by introducing new regularization terms for similar users and similar items.

The intuition behind similarity based regularization is that similar users have similar tastes. Hence, we impose that the factor vector of each user (resp. item) should be close to the average factor vector of its similar users (resp. items). For computing the most similar users (or items) we considered a modified version of Pearson correlation coefficient [Herlocker et al., 1999] which for two users u_i and u_j writes:

$$sim(u_i, u_j) = \frac{\sum_{i_k \in I_c}(r_{ik} - \overline{r}_{i.})(r_{jk} - \overline{r}_{j.})}{\sqrt{\sum_{i_k \in I_c}(r_{ik} - \overline{r}_{i.})^2}\sqrt{\sum_{i_k \in I_c}(r_{jk} - \overline{r}_{j.})^2}}$$

Where, I_c is the items co-rated by both users, $\overline{r}_{i.}$ and $\overline{r}_{j.}$ denote the average ratings for u_i and u_j respectively.

Hence, we are able to find the N most similar users for u_i, denoted by N_i (resp. N_j for items similar to i_j). We now propose a slight modification of the individual ratings

objective function ℓ of Eq. (5.2) above by adding another regularization term. For a pair of user and item (u_i, i_j) for which a rating r_{ij} exists, the similarity-regularized individual objective writes:

$$\ell_1(u_i, i_j, P, Q) = (r_{ij} - q_j^\top p_i)^2 + \lambda(||p_i||^2 + ||q_j||^2)$$
$$+\lambda_u \left\| p_i - \frac{1}{|N_i|} \sum_{m \in N_i} p_m \right\|^2 + \lambda_i \left\| q_j - \frac{1}{|N_j|} \sum_{n \in N_j} q_n \right\|^2 \tag{5.4}$$

where $\lambda_u \geq 0$ and $\lambda_i \geq 0$ are the regularization parameters linked to the similar-user and similar-item regularizations respectively. Performing the same updates as the conventional SGD but replacing ℓ by ℓ_1, we get Algorithm 6 for minimizing the whole matrix factorization problem (Eq. 5.3) where ℓ is replaced by ℓ_1 i.e. the similarity-regularized problem:

$$\min_{P,Q} \sum_{i,j:r_{ij}\text{exists}} \ell_1(u_i, i_j, P, Q). \tag{5.5}$$

Input: $R, \lambda, \lambda_u, \lambda_i$

Initialize: $\mathbb{P}$ and $\mathbb{Q}$ randomly **while** *not converged* **do**

Choose randomly $(u_i, i_j) \in R$

$N_i = GetSimilarUsers(i, N)$

$N_j = GetSimilarItems(j, N)$

Update p_i and q_j by a gradient step on $\ell_1(u_i, i_j, \mathbb{P}, \mathbb{Q})$ (Eq. 5.4)

end

Algorithm 6: Similarity based regularization

5.3 Related Work

Despite its simplicity, there are several computational challenges associated with this problem. As previously, performing SGD sequentially on a single machine takes unacceptably large amount of time to converge for common rating matrices of several million ratings. So, there is a need to perform SGD in an efficient distributed manner for such large datasets. In this section we will show a detailed account of performing large-scale matrix factorization in a distributed manner.

Hence, to handle the large-scale matrix factorization, we can distribute the computation across multiple workers. This gives rise to two distributed architectures:

Shared-memory (parallel) and shared-nothing. In the case of shared-memory methods, the entire data is kept in the memory of a single machine and multiple processors work parallelly on the data. However, this might not be a feasible solution if the size of dataset is very huge to fit in a single machine, which is usually the case in modern RS applications. Hence, to overcome this limitation, shared-nothing (totally distributed) approaches are used, in which the machines do not share memory and the dataset is kept in disjoint machines. Even thought the main focus of this thesis is for shared-nothing (distributed) scenario, for the sake of completeness we will present some of the popular shared-memory methods.

5.3.1 SHARED-MEMORY METHODS

Earlier work in this line include methods with the name Parallelized SGD (PSGD), in which the dataset is partitioned into several parts and SGD is run independently and in parallel on different subparts. The updated parameters corresponding to each subpart are averaged either after each pass over the data [Hall et al., 2010, McDonald et al., 2010] or once at the end [Zinkevich et al., 2010]. These methods rely on synchronization between the parallel processes, hence exhibit slow convergence rate in practice. Another popular method is HogWild [Recht et al., 2011], which randomly selects subset of rating matrix and apply the update rules simultaneously in parallel fashion without any synchronization between the threads. They also guarantee the convergence of their method when factorizing a highly sparse matrix, where one can ensure that the occurrence of over-writing problem because of multiple threads trying to update the same user/item factor is rare. In a slightly different line of work, [Gemulla et al., 2011] introduced the idea of Stratified SGD (SSGD) and introduced oen algorithm in this line: Distributed SGD (DSGD). This method partitions the rating matrix into disjoint (interchangeable) blocks in which parameter update corresponding to one block is mutually independent to another one. Hence, these methods can be easily parallelized and extended in shared-nothing settings as well. Both of these methods HogWild and DSGD suffer from problems such as: locking problem (arising because of synchronous operation) and memory discontinuity [Zhuang et al., 2013]. FPSGD [Zhuang et al., 2013] alleviates the memory locking problem by introducing a novel blocking scheme. Similarly they solve the memory discontinuity by introducing a solution called partial random method which randomly chooses a free block and accesses the block sequentially.

5.3.2 SHARED-NOTHING METHODS

These methods are designed to handle large-scale matrix factorization problems, typically when the rating matrix or the factor matrices are too large to be fit in the memory of a single machine. Here, the important assumption is that the worker machines have disjoint memory. Hence, the main challenge in shared-nothing methods is to have effective communication between the computing nodes [Makari et al., 2015]. Hence, these methods are not that popular as compared to the shared-memory methods in the literature.

A general workflow of shared-nothing methods can be represented as below:

- Partition the data and factor matrices, and dispatch them across the different computing nodes.
- Each node works on different subparts of data and updates the factor matrices accordingly.
- Nodes communicate (exchange) updated parameters between them to have an agreement on the updates. This communication is done once each epoch or multiple times in an epoch.

Depending on the type of partitioning of the data, these methods can be categorized into two types of methods:

5.3.2.1 Row (or column) wise splitting

One popular example of this type of splitting is ASGD algorithm presented in [Makari et al., 2015]. In this approach, the rating matrix is partitioned row-wise into several blocks and SGD is run on individual blocks on distinct machines. From the decomposition $\hat{R} = PQ^\top$, one can see that if the rating matrix is divided by row-blocks, $\hat{R}_b = P_b Q^\top$, that is; the block b of $\hat{R}$ depends only on the block b of P then, the block-split problem writes:

$$\min_{P,Q} \sum_{\text{blocks } b} \left[\sum_{(u,i):r_{b_{ui}} \text{exists}} \ell(P_b, Q, u, i) \right]. \tag{5.6}$$

Factor matrices are thus updated independently on each machine for the corresponding ratings. Even though the rating matrix parts on each machine are different, the factor matrix updates are not independent. So, after each epoch the factor matrices present in each machine are synchronized. However, the machines send the

updates to the master machine immediately and hence the author has referred this as an asynchronous method. But this cannot avoid the bottleneck of synchronization after every epoch.

5.3.2.2 Stratified SGD

The main idea of stratified SGD methods is to exploit the structure of matrix factorization problem and induce disjoint blocks which can be parallelized easily. Each of these disjoint parts are referred to as stratum (or blocks). One such exmple of stratification of rating matrix is shown in Figure 5.3. Here, the rating matrix is denoted as V, and the numbers in the superscript represent the row and column numbers respectively. Also, Algorithm 7 summarizes the generic algorithm of stratifies SGD method.

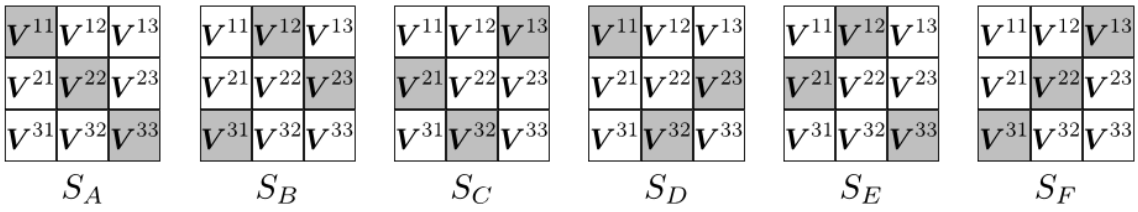


Figure 5.3: Strata used by SSGD for a 3 × 3 blocking of V [Makari et al., 2015]

Input: Incomplete matrix $\mathbb{R}^{m\times n}$, factor matrices: $\mathbb{P}^{m\times k}$ and $\mathbb{Q}^{n\times k}$, blocking parameter b

Block $\mathbb{R}$, $\mathbb{P}$ and $\mathbb{Q}$ into b × b, b × 1 and 1 × b blocks

while *not converged* **do**
  Choose step size α
  for $s = 1, ..., b$ **do**
    Pick w blocks R^{1j1}, ..., R^{bjb} to form a stratum
    for $k = 1, ..., b$ **do**
      Run SGD on the training points in $\mathbb{R}^{kj_k}$ with step size α
    end
  end
end

Algorithm 7: Generic SSGD Algorithm

One popular approach based on stratified SGD is, referred to as Distributed SGD (DSGD) [Gemulla et al., 2011, Makari et al., 2015], which we alredy discussed before in the shared memory section. This method can be extended to be applied in

distributed memory architecture as well by placing the disjoint blocks in disjoint computing nodes. Even though these methods can overcome the problem of simultaneous updates, they require several synchronizations within an epoch which hurts their computational performance.

5.4 ADG_{MF} Algorithm

As we saw in the previous section, even though the stratified partitioning helps to make disjoint sections and rules out the problem of locking or overwriting of updates, it requires synchronization even within the epochs. Hence, in this work we stick with the row-wise blocking of the rating matrix. We apply our asynchronous distributed framework introduced in Section 4.3. In order to apply the asynchronous distributed strategy to this problem (referred to as ADG_{MF} in the following), we split the rating matrix in row-wise manner. Hence, in this case, we only need to communicate the matrices Q between machines, whereas the matrices P are updated locally, corresponding to each sub-part, and are later concatenated at the end of the operation. Hence, this corresponds to Case 1 of Section 4.2. In the distributed network, one of the machines acts as the master machine, whereas the other machines act as the workers. The master machine can also act as one of the workers. In our experiments also we used the master machine as one of the workers. The overall optimization is performed into following two steps:

- Worker Step:

  As shown in the asynchronous distributed architecture 4.1b, each worker machine works on their local subpart of data. Because of the row-wise splitting, the updates on $\mathbb{P}$ matrix are disjoint and hence local to each machine. Whereas, all the worker machines update same rows of $\mathbb{Q}$ matrix. Hence, all the machines need to have an agreement on the updates made on $\mathbb{Q}$ matrix for better and faster convergence. Hence, each worker machines communicate the updated $\mathbb{Q}$ matrix to the master machine, which takes care of aggregating all the received updates from the worker machines and broadcasting it back to the workers. We will discuss more about the master step next. Soon, after sending the updated parameter to the master machine, the worker machine checks if it has received an aggregated $\mathbb{Q}$ parameter from the master or not. If it has received the updated parameter from the master machine, it will begin a new epoch with the updated

$\mathbb{Q}$ parameter, otherwise it will continue with its previously updated parameter matrices. In this way, even if the workers are disjoint, they have a common view over the whole dataset and this helps them to converge faster. Also, the slower workers will complete each epoch slow as compared to the faster machines, but once they finish their epoch, they will receive the most updated aggregated parameter from the master. The overall steps performed in a worker step are shown in Algorithm 8.

Parameters: learning rate γ
Initialize: P_j
Receive matrix Q from the master;
From the subpart of the data stored in machine j, **pick** randomly (u, i) for which r_{ui} exists ;
$(P_j, Q_j) \leftarrow (P_j, Q) - \gamma \nabla \ell(P_j, Q, u, i)$;
Send Q_j to the master;

Algorithm 8: $\texttt{ADG}_{\texttt{MF}}$ worker step in the computing machine $j \in \{1, \ldots, m\}$

- Master Step:

  Master machine performs the task of collecting the received updates from the workers and aggregating the received $\mathbb{Q}$ matrices. This aggregation can be performed in a timely manner depending on the application. In our experiments, we used master as a worker as well and performed the aggregation after each epoch on master. Hence, after each epoch the master checks for received updates from the worker machines and it averages the received updates. Soon after averaging it broadcasts the aggregated parameter to the workers so that they can use it immediately in their next epoch. The master step is shown in Algorithm 9.

Initialize: machines M and $\overline{\mathbb{Q}}$
Receive matrix $\mathbb{Q}_i$ from the subset of workers (m $\subset$ M and i $\in$ m) ;
Compute $\overline{\mathbb{Q}} = \frac{1}{m} \sum_{j=1}^{m} \mathbb{Q}_j$;

Broadcast $\overline{\mathbb{Q}}$

Algorithm 9: $\texttt{ADG}_{\texttt{MF}}$ **master step** in the master machine

Hence, we can observe that both master machine and workers perform their task independently and asynchronously. Hence, this approach avoids the performance bottleneck due to slower machines in the network.

5.5 Experimental Results

5.5.1 Experimental Setup

We conducted a number of experiments to empirically validate the proposed asynchronous framework on matrix factorization for recommendation where the recommendation matrix is split into M rows as in Problem (5.6).

1. Datasets: We performed experiments on Movielens-10M (ML-10M)[2] and the Netflix Collection[3] that are two popular corpora in collaborative filtering.

Table 5.1: Characteristics of Datasets used in our experiments. $|\mathscr{U}|$ and $|\mathscr{I}|$ denote respectively the number of users and items.

Dataset	$\|\mathscr{U}\|$	$\|\mathscr{I}\|$	γ	λ	K	training size	test size	sparsity
ML-10M	71567	10681	0.005	0.05	100	9301274	698780	98.7 %
NetFlix (NF)	480189	17770	0.005	0.05	40	99072112	1408395	99.8 %
NF-Subset	28978	1821	0.005	0.05	40	3255352	100478	93.7 %

2. Baselines: To validate the asynchronous distributed algorithm described in the previous section, we compare the following four strategies:
   - The proposed approach $\texttt{ADG}_{\texttt{MF}}$ (Section 5.4),
   - The asynchronous distributed ADMM approach (`AD-ADMM`) [Chang et al., 2015],
   - Two distributed algorithms specifically proposed for matrix factorization `ASGD` [Makari et al., 2015] and `DSGD` [Makari et al., 2015].
3. Platform: The distributed framework we considered was implemented using PySpark version 1.5.1. by connecting 7 servers with different computational power.
4. Hyper-Parameters: Various free parameters of `SGD` such as learning rate (γ), regularization parameter (λ) and number of latent factors (K) were set following [Chin et al., 2015], [Yu et al., 2014b]. For our proposed similarity-based regularization λ_u, λ_i, and the number of similar users/items N were

[2] http://grouplens.org/datasets/movielens/
[3] http://www.netflixprize.com/

chosen with values that led to the best RMSE on validation sets for each collection chosen among $\{10^{-1}, 5.10^{-2}, 10^{-2}, 5.10^{-3}, 10^{-3}, 5.10^{-4}, 10^{-4}\}$ for λ_u, λ_i and $\{10, 20, 30, 40, 50\}$ for N. These values as well as the datasets characteristics are listed in Table 6.1.

5. Evaluation Measures: In our experiments, we used the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) as performance measures. Also, we compared the convergence time for each of the algorithms in the synchronous and the asynchronous distributed settings.

5.5.2 THE EFFECT OF SIMILARITY BASED REGULARIZATION

Table 5.2: MAE and RMSE measures for different methods on MovieLens and NetFlix datasets. Best results are shown in bold.

Dataset		SGD		Similarity Based Regularization			
				Similar Users and Items		Similar Users Only	
		MAE	RMSE	MAE	RMSE	MAE	RMSE
ML-100K	ua	0.7490	0.9478	**0.7390**	**0.9332**	0.7404	0.9359
	ub	0.7619	0.9660	**0.7555**	**0.9564**	0.7540	0.9590
ML-1M	ra	0.7324	0.9706	0.7208	0.9517	**0.7188**	**0.9487**
	rb	0.6973	0.8861	**0.6928**	**0.8787**	0.6946	0.8799
ML-10M	rb	0.6523	0.8415	**0.6488**	**0.8384**	0.6512	0.8402
NF-Subset	NA	0.6498	0.8287	**0.6469**	**0.8256**	0.6477	0.8267

First, we compare the results between the traditional SGD method and the proposed Modified SGD with Similarity Based Regularization. The difference here is solely on the objective function that is minimized (Pbs. (5.3) and (5.5) respectively). We tested two scenarios: i) where only the users are regularized with similarity ($\lambda_i = 0$); and ii) when both users and items are regularized with the same parameter ($\lambda_u = \lambda_i$). Table 5.2 shows the complete results of our experiments.

It comes out that forcing the vectors of users and items to lie within the centroids of their most similar users and items found by the Pearson similarity measure is effective as the final RMSE and MAE with Algorithm 1 are always better than with classical SGD. Thus, there is a significant benefit to use this regularization in terms of learning. We also report results by looking at the effect of the similar user regularization and

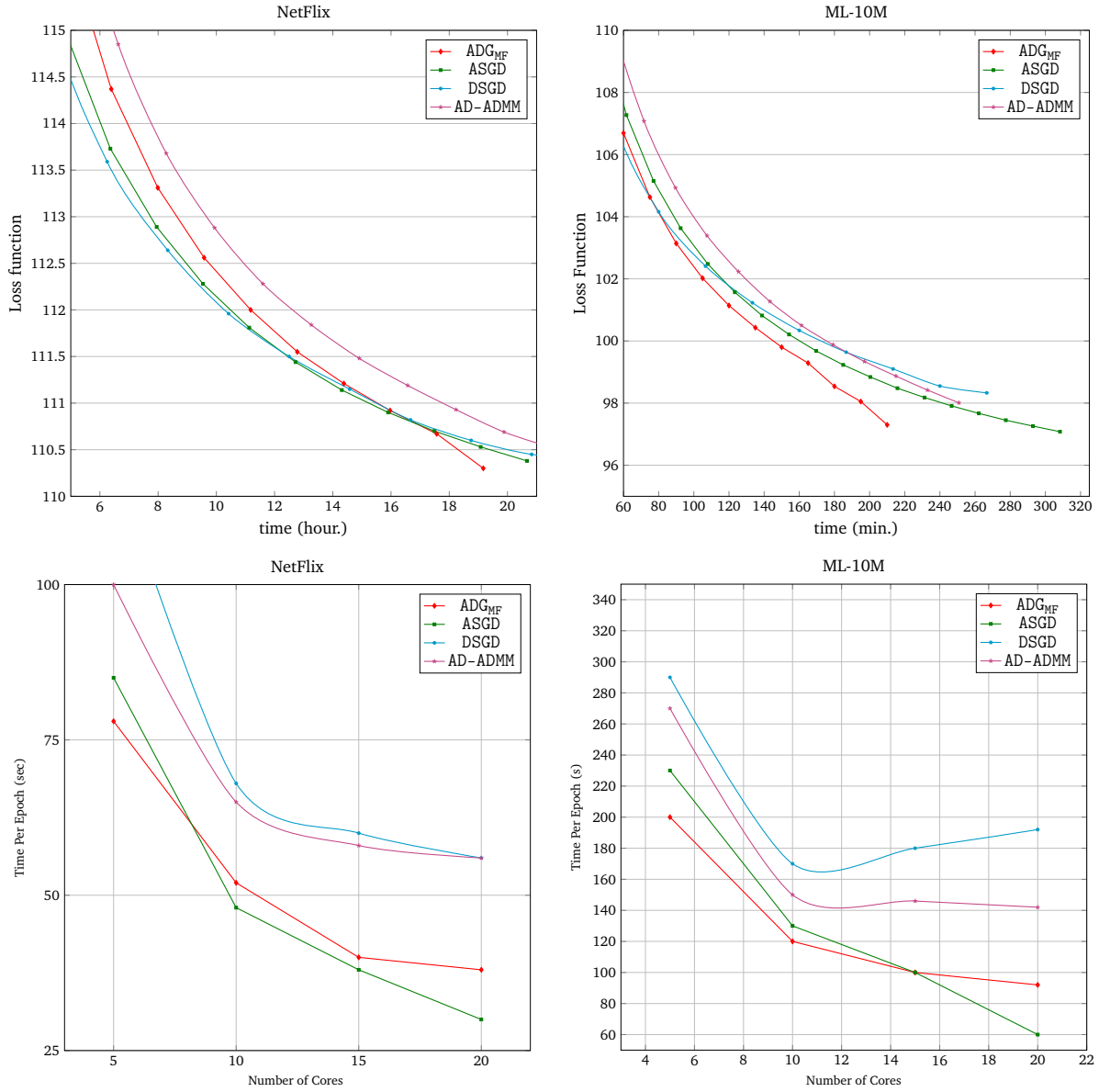


Figure 5.4: **Top**: Test RMSE curves with respect to time for $\texttt{ADG}_{\texttt{MF}}$, AD-ADMM, ASGD, and DSGD on NetFlix (left), and ML-10M (right) Datasets. **Bottom**: Total Convergence Time Vs. Number of Cores curves for $\texttt{ADG}_{\texttt{MF}}$, ASGD, DSGD and AD-ADMM on the NetFlix (left), and ML-10M (right) Datasets.

not items ($\lambda_u > 0, \lambda_i = 0$). As shown in Table 5.2, this user-only regularization also gives uniformly better results than traditional SGD, and even better than the user and item regularization on one dataset.

5.5.3 Evaluation of Convergence Time

We begin our experiments by comparing the evolution of the loss function of Eq. (5.3) with respect to time until convergence. The convergence points are shown as names of the algorithms vertically (we stopped ASGD after 20 hours on the NF dataset). Figure 5.4 (top) depicts this evolution for ML-10M and NF datasets using 10 and 15 cores respectively. Synchronization based approaches (ASGD and DSGD) aggregate all the information at each epoch and thus begin to converge more sharply at the beginning. However, with these approaches, when the fastest machines finish their computations, they have to wait for slower machines; thus, they require much more time to converge than the asynchronous methods (AD-ADMM and ADG_{MF}). Finally, it comes out that ADG_{MF} converges faster than the other algorithms on both datasets. This is mainly due to the fact that ADG_{MF} does not obey to any delay mechanism as in AD-ADMM for instance.

5.5.4 Computation and Communication Trade-off

We performed another set of experiments aimed at measuring the effect of number of cores on performance of the proposed approach and the baselines. Figure 5.4 (bottom) depicts this effect by showing the evolution of time per epoch of the SGD method used in ADG_{MF}, ASGD, DSGD and AD-ADMM with respect to increasing number of machines. From these experiments, it comes out that for all approaches the time per epoch of the method decreases as the number of machines increases.

But after a certain number of machines (10 in both experiments), the time per epoch of some approaches begin to be affected as the communication cost takes over the computation time. The approach that is the most affected by this is DSGD, as synchronizations in this case are done after each sub-epoch. We can also see that even though the per epoch speedup is best for ASGD, it requires a much higher number of epochs to converge as compared to ADG_{MF} and DSGD.

5.6 Closing Remarks

In this Chapter, we extensively studied the Recommender System application. Modern RS application involves dataset of very large magnitude. Hence, it has emerged as a challenging task. While disucssing the approaches for RS, we noticed that Matrix factorization approaches have received a good reputation in the research community because of the success promised by these methods. However, when the size of dataset becomes very huge, it becomes infeasible to perform traditional matrix factorization methods in a single machine. Hence, distributed methods are required to tackle this problem. In this Chapter, we applied our asynchronous distributed framework to perform matrix factorization in a distributed manner. Additionally, we introduced an extra regularization term to incorporate the user and item similairity in the error function of SGD for matrix factorization. The use of neighbourhood information promised to give improvement in the performance of SGD method. Also, in the experiment section we demonstrated the benefits of using asynchronous framework over the synchronization based baselines.

6 Application 2: Distributed Binary Classification

6.1 Binary Classification

6.1.1 Introduction

Binary classification refers to supervised learning problem involving the classification of an example to one of the two class labels. Majority of real-time applications involve binary classification or can be modeled as a binary classification problem. Some of the popular examples of binary classification are: spam detection, cancer detection, deciding whether or not to show an item or add to a user etc. A binary dataset is denoted as:

$$(x_i, y_i) \in \mathbb{R}^d \times \{-1, +1\}$$

where, i = 1, ..., n are the training examples, x_i is the feature vector and y_i is the corresponding label.

6.1.2 Linear Vs Non-linear models

Given a training set (x_i, y_i), binary classification methods construct the followin decision function [Yuan et al., 2012]:

$$d(x) = \mathbf{w}^T \phi(x) + b$$

Where, **w** is referred as the weight vector and b is an intercept known as bias. This decision function defines a separating plane which separates the instances corresponding to the two classes. Hence, based on the nature of this decision function the binary

classification method can be divided into two types: linear classifiers and non-linear classifiers. As the name suggests, linear classifiers use a linear separating boundary, whereas it is non-linear in the case of non-linear classifiers. Non-linear classifiers map each of the training examples to a higher dimensional vector $\phi(x)$. In contrary, the linear classifiers use the original feature space.

Since, non-linear classifiers use more features, they are supposed to have better predictive perfomrance as compread to similar linear methods. However, in many applications and experiments [Yuan et al., 2012], linear classifiers are shown to have similar accuracy as compared to the non-linear classifiers. Whereas, linear classifiers are far more efficient in terms of training and testing time. This makes them very useful for large-scale scenarios. Hence, in this work, we will focus on linear methods for binary classification.

6.1.3 BINARY LINEAR CLASSIFICATION METHODS

Linear binary classification involves the risk minimization of the following error function:

$$\mathscr{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}) + r(\mathbf{w}) \tag{6.1}$$

Where, $r(\mathbf{w})$ is the regularization term and $\ell_i(\mathbf{w})$ is the loss function associated with each example in the training set. The loss function in 6.1 penalizes the misclassified instances (x,y). Also, the type of loss function used helps to distinguish between different learning algorithms. Some of the commonly used loss functions are listed below:

$$\ell_{iL1}(\mathbf{w}) = max(0, 1 - y_i\mathbf{w}^T x_i) \tag{6.2}$$

$$\ell_{iL2}(\mathbf{w}) = max(0, 1 - y_i\mathbf{w}^T x_i)^2 \tag{6.3}$$

$$\ell_{iLR}(\mathbf{w}) = \log(1 + e^{-y_i\mathbf{w}^T x_i}) \tag{6.4}$$

Here, 6.2 and 6.3 are used in L1-loss and L2-loss Support Vector Machine (SVM) [Boser et al., 1992, Cortes and Vapnik, 1995] respectively, whereas 6.4 is used by the Logistic Regression algorithm [Cramer, 2002]. These algorithms are popularly used for binary classification. Both of them have advantage and disadvantages for different kind of datasets. The three loss functions in 6.2–6.4 are all convex and non-negative.

Hence, the popular optimization methods can be applied for their minimization. We will explore the different state of the art optimization schemes in the Related Work section (6.2).

6.2 Related Work

One of the most popularly used optimization methods to minimize the loss function of the form 4.3 and 6.1 is full gradient descent (FG) method which dates back to [Cauchy, 1847]. FG method uses the iterations of the form:

$$\begin{aligned} \mathbf{w}^{k+1} &= \mathbf{w}^k - \alpha_k \mathscr{L}'(\mathbf{w}^k) \\ &= \mathbf{w}^k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \ell_i(\mathbf{w}^k) \end{aligned} \tag{6.5}$$

The convergence of FG method is fast, it can be unappealing when n becomes significantly large. As the sum in Eq. (4.3) becomes very large, computing gradients of $\mathscr{L}$ would be computationally very expensive. To overcome this issue, it is common to use Stochastic Gradient Descent (SGD) methods where instead of updating the current iterate using a full gradient, only one (or a few) randomly selected terms of the sum are considered [Bottou, 2010, Roux et al., 2012, Shalev-Shwartz and Zhang, 2013]. Updating with this randomly subsambled gradient instead of the true one leads to a variance-like error in the iteration that has to be mitigated, for instance by using decreasing stepsizes which is harmful in practice [Bottou, 2010, Johnson and Zhang, 2013]. Another, more performing, way to deal with this variance is to use *variance reduced* variants of SGD, such as SVRG [Johnson and Zhang, 2013] or SAG/SAGA [Roux et al., 2012, Defazio et al., 2014]. These methods incorporate incremental benefits of SGD whereas reducing the variance caused by random-sampling in SGD by occasionally computing full-gradients. This reduction in variance contributes to better convergence properties and the use of fixed stepsizes.

However, with the increasing size of the datasets, it has become impossible to store and process data in a single machine. In this case, the most common approach consists in partitioning the dataset into several machines, and to solve the optimization problem in a distributed manner [Langford et al., 2009]. The majority of the distributed methods rely on a synchronization between the worker machines [Shamir et al., 2014, Boyd et al., 2011] where, the information from all the workers are gath-

ered after every iteration and the parameters are synchronized. For these methods, the loading of machines play a central role in the convergence time of the whole system and in the extreme case, the slowest machine may become a bottleneck. To overcome this shortcoming, recent studies have considered asynchrony in the communication of the shared parameter between the machines [Dean et al., 2012, Zhang and Kwok, 2014, Zhang et al., 2015, Reddi et al., 2015, Huo and Huang, 2016]. Asynchronous algorithms can be applied either in shared-memory or distributed-memory environments. Shared memory algorithms, commonly referred as parallel distributed algorithms, are mainly devised for multi-core systems. All the cores in such system share the main memory, hence the parameter vector is usually kept in main memory and is accessible by all the processing units for making any updates. Some of the prominent parallel implementation of SGD or its variants are: Hogwild! [Recht et al., 2011], CoCoA [Jaggi et al., 2014], AsySVRG [Zhao and Li, 2016] and ASAGA [Leblond et al., 2016]. Even though having promising theoretical/empirical results and ease of implementation these approaches are limited to multi-core systems.

On the other hand, distributed approaches introduce asynchrony in distributed memory environments. One popular architecture for distributed algorithms is parameter server (PS) implementation. The server keeps receiving delayed information from a subset of workers in each iteration, thus avoiding the full synchronization among all workers. One popular example of such architecture is downpour SGD [Dean et al., 2012]. Here, each worker reads the parameter vector from the server, computes the local gradient and pushes the updates to the server. Hence, the gradient updates for each mini-batch are sent back to the server, which updates the parameter vector for each received gradients. Following this architecture, recently variance reduced versions of SGD has been implemented in asynchronous distributed setting in [Zhang et al., 2015, Huo and Huang, 2016]. Even though both the methods communicate the gradients in an asynchronous fashion, they suffer from mainly two drawbacks. First after each mini-batch update the gradient should be communicated. So, if the size of dataset grows large the communication cost will become huge especially for a large number of workers. Secondly, even if the mini-batch updates are asynchronous, these algorithms synchronize after one complete pass over the data which is penalizing as in disparate distributed environments any sort of synchronization can lead to slower performance.

6.3 DISTRIBUTED SVRG ALGORITHM

In this section, we will analyze the SVRG method for binary classification. First, we will begin with the single machine SVRG algorihtm. Then, we will present our proposed asynchronous distributed version of the SVRG algorithm denoted as $\texttt{ADG}_{\texttt{BC}}$.

6.3.1 SINGLE MACHINE *SVRG*

Single machine SVRG algorithm is presented in Algorithm 10. In this method, at each time (usually after one complete pass over the data), we keep a snapshot of estimated $\mathbf{w}$, denoted as $\widetilde{\mathbf{w}}$. Then we maintain an average gradient over the whole data using the snapshot, $\widetilde{\mathbf{w}}$ as:

$$\widetilde{\mu} = \nabla \mathscr{L}(\widetilde{\mathbf{w}}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \mathscr{L}_i(\widetilde{\mathbf{w}}) \tag{6.6}$$

For each single or mini-batch updates of inner iteration, parameter $\mathbf{w}$ is updated as:

$$\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} - \frac{\gamma}{|I^t|}\sum_{i\in I^t}(\nabla \mathscr{L}_i(\mathbf{w}^{t-1}) - \nabla \mathscr{L}_i(\widetilde{\mathbf{w}}) + \widetilde{\mu}) \tag{6.7}$$

where γ is the learning rate.

This modification in update rule of SGD contributes to the reduction of variance of the algorithm near the convergence point and also leads to a linear convergence of the algorithm.

Parameters: Update frequency m, batch size B and learning rate γ
Initialize: $\mathbf{w} \in \mathbb{R}^d$
while $s = 1, 2, .., S$ **do**
  $\widetilde{\mathbf{w}} \leftarrow \mathbf{w}$; $\mathbf{w}^0 \leftarrow \mathbf{w}$
  Compute $\widetilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \nabla \mathscr{L}_i(\widetilde{\mathbf{w}})$
  for $t = 0, 1, 2.., m-1$ **do**
    Randomly pick a mini-batch I^t s.t. $|I^t| = B$
    $\mathbf{v}^t = \frac{1}{|I^t|}\sum_{i\in I^t}(\nabla \mathscr{L}_i(\mathbf{w}^t) - \nabla \mathscr{L}_i(\widetilde{\mathbf{w}}) + \widetilde{\mu})$
    update $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \gamma \mathbf{v}^t$
  end
  $\mathbf{w} \leftarrow \mathbf{w}^m$
end

Algorithm 10: Single Machine SVRG Algorithm

6.3.2 ADG_{BC} ALGORITHM

In this section, we present our proposed asynchronous distributed SVRG algorithm, using the asynchronous framework discussed in previous section. The distributed memory algorithm in the master node and the worker nodes are shown in Algorithms 11 and 12. Each machine perform parameter update on their local data and after each iteration the worker machines send the updated parameter to the server node which directly responds by sending the averaged common parameter using the last gathered updates. In this way, all the machines have an overall view of the parameter updates from whole data, while only working with the local data.

The server node can also be used as a worker, updating the parameter on its local sub-part of the data. Here, one thing to note is that we compute the average gradient, $\widetilde{\mu}$ in SVRG, only over the local data. The average gradient over local data using the parameter updated using the data on all machines gives a good approximation of the full average gradient over the whole dataset. This allows us to avoid the need for synchronization among the machines after one pass over the full data.

Initialize: Iteration k, machines M and $\overline{\mathbf{w}}$

Receive $\mathbf{w}_i$ from the subset of workers (m $\subset$ M and i $\in$ m) ;

Compute $\overline{\mathbf{w}} = \frac{1}{m}\sum_{j=1}^{m} \mathbf{w}_j$;

Broadcast $\overline{\mathbf{w}}$

Algorithm 11: ADG_{BC} master step in the master machine

Input: Maximum number of iterations T, batch size B and learning rate γ

Initialize: $*$ Receive parameter $\widetilde{\mathbf{w}} \in \mathbb{R}^d$ from the master, or use the last parameter estimation happened before a new reception ;

$* \ \mathbf{w}^0 \leftarrow \widetilde{\mathbf{w}}$;

$*$ Compute $\tilde{\mu}_j \leftarrow \bar{\nabla}\mathscr{L}_j(\widetilde{\mathbf{w}})$;

for $t = 0, .., T-1$ **do**

  Randomly pick a mini-batch I_j^t of size B in the subpart of the data stored in machine j;

  Update $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \frac{\gamma}{|I_j^t|}\sum_{(\mathbf{x}_i, y_i)\in I_j^t}(\nabla\ell(\mathbf{w}^t, \mathbf{x}_i, y_i) - \nabla\ell(\widetilde{\mathbf{w}}, \mathbf{x}_i, y_i) + \widetilde{\mu}_j)$;

end

$\widetilde{\mathbf{w}} \leftarrow \mathbf{w}^T$ and send $\mathbf{w}^T$ to the master.

Algorithm 12: ADG_{BC} local step in the computing machines $j \in \{1, \ldots, m\}$

6.4 EXPERIMENTAL RESULTS

We conducted a number of experiments aimed at showing the behaviour of the proposed $\texttt{ADG}_{\texttt{BC}}$ algorithm in learning efficient classification functions optimizing the ℓ_2-regularized logistic regression surrogate. Specifically, we study the convergence and the communication overhead of the proposed algorithm by comparing it with state-of-the-art distributed approaches.

6.4.1 EXPERIMENTAL SETUP

6.4.1.1 Datasets

We performed our experiments on three popular large-scale binary classification datasets: `Webspam`, `Epsilon` and `RCV`. The various characteristics of the datasets are presented in Table 6.1. Note that `Epsilon` is fully dense while `RCV` is the sparsest dataset.[1]

Table 6.1: Characteristics of Datasets used in our experiments.

Dataset	n	d	$\#nonzeros$
`Epsilon`	500000	2000	10^9
`Webspam` Unigram	350000	253	29,796,333
`RCV`	697641	47236	51,055,210

6.4.1.2 Baselines

The majority of distributed approaches consider the shared-memory scenario, where the parameter vector is kept in the shared memory, which can be updated by all the processors simultaneously [Zhao and Li, 2016, Leblond et al., 2016]. But the focus of this paper is for shared-nothing scenario, where the disparate machines do not share memory. Unlike most of the approaches which rely on some sort of synchronization among the machines, we consider a totally asynchronous setting. To validate the asynchronous distributed algorithm described in previous section, we compare the following strategies:

- The proposed approach ADSVRG,

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

- `Sync-SVRG`, SVRG based method with synchronization of gradients after every mini-batch update.
- `Async-SVRG`: Distributed architecture proposed in [Huo and Huang, 2016], which asynchronously communicate gradients after every mini-batch updates.

Since the asynchronous methods were quite sensitive to initial point, we performed a synchronized gradient step during the first pass over the data. This gave a stable start for all the algorithms.

6.4.1.3 Experimental Settings

We shall now describe the platform, as well as the tuning of the hyper-parameters and the evaluation measures used in our experiments.

1. **Platform:** Experiments were conducted in a platform with 7 servers. The code was implemented using a python module `mpi4py` using OpenMPI[2] as the MPI library. Since the focus of the paper is for shared-nothing scenario, the disparate machines do not share memory. Three of the servers had Intel Xenon E5-2640 2.60 GHz processors with 32 cores and 256 GB memory each. Two others had Intel Xenon E5-2643 3.40 GHz processors with 24 cores and 128 GB memory each and the last two ones had have Intel Xeon E5-2407 2.20GHz processors with 4 cores and 48 GB memory each. Each core of a server here corresponds to a computing node or a machine that we considered in our analysis presented in the previous sections. Even though some of the servers have identical configuration, they were running different workloads on them making the configuration similar to a real scenario case.
2. **Hyper-parameters:** In all the experiments, we used a fixed regularization rate, $\lambda = \frac{1}{n}$, where n is the size of the dataset. The fixed learning rates were chosen from a set of values in range $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and the reported performance were the best obtained with one of those stepsizes. The mini-batch size for `Webspam`, `Epsilon` and `RCV` datasets were respectively fixed to 5, 10 and 20.
3. **Evaluation Measures:** Convergence result was evaluated in terms of minimization of objective function over time. The communication overhead incurred by

[2] `https://www.open-mpi.org/`

each algorithm in the network as well as the communication time are shown in terms of the total number of send/receive calls.

6.4.2 RESULTS

We performed experiments to assess the effect of varying broadcast frequency on the performance of asynchronous methods. We were then interested in the convergence time as well as the communication overhead and the effect of increasing the number of workers.

6.4.2.1 Effect of Varying Broadcast Frequency

We chose varying batch sizes after which the parameter vectors were broadcasted in each worker machines. The effect of varying batch size for broadcast on the minimization of objective function on training data for `Webspam` and `Epsilon` datasets are shown in Figure 6.1. As it can be observed, the broadcasting of the parameter vectors has in general a negative effect on the performance of the algorithm. In Figure 6.1, we can notice that when the parameter was broadcasted after 1/10000 or 1/1000'th pass over the local data, the minimization of the objective function is not very good, whereas broadcasting after a larger pass over the data improves the minimization up to some point when the parameters pooling do not occur often enough. This gives a good motivation for our approach of broadcasting parameter vectors only after updating the parameter vector over one pass of the local data.

6.4.2.2 Evaluation of Convergence Time

Figure 6.2 and 6.3 compare the convergence results for the three methods on all datasets. The convergence results are presented in terms of minimization of the objective function in the training sub-part of the data on the the root machine and the evolution of the test accuracy with respect to time (in seconds). As It can be observed the proposed method *ADSVRG* converges much faster than the other two approaches which is mainly due to the synchronization of gradients between the worker machines. The objective function is minimized considerably faster for *ADSVRG* than the other two methods. It can be seen that this behavior becomes more noticeable for larger datasets. For example on the `RCV` collection, *ADSVRG* converges three times faster than the other methods. As an effect the test performance reached by *ADSVRG*, at its

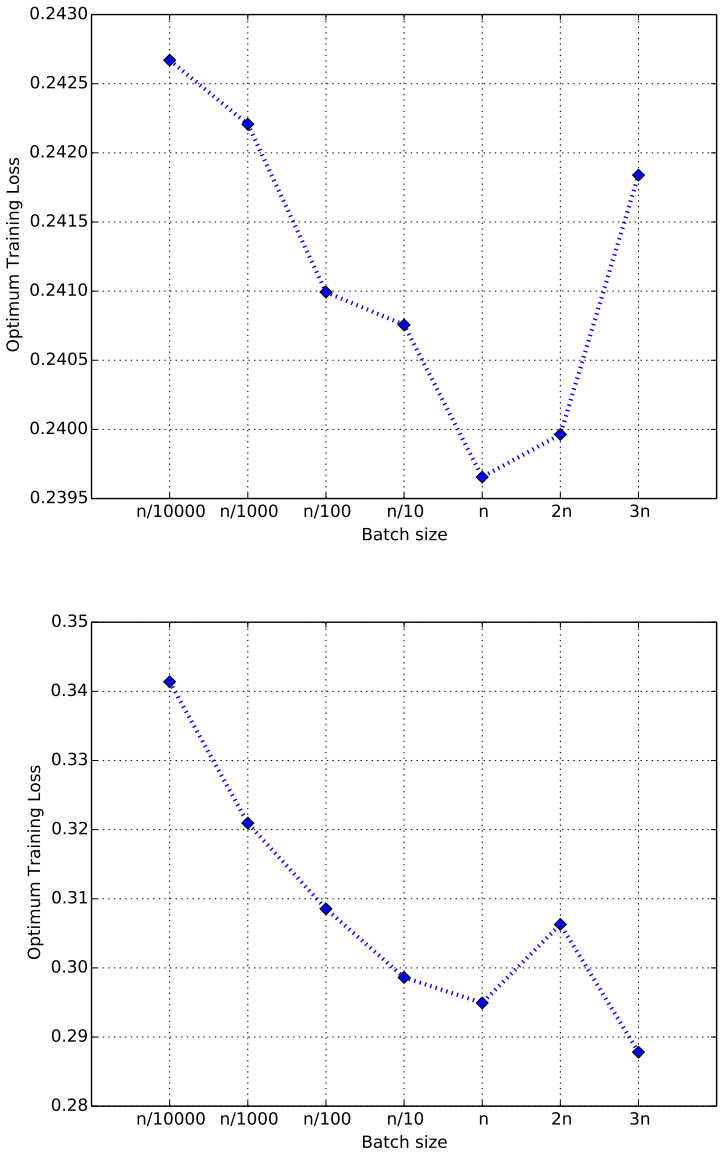


Figure 6.1: The effect of varying batch size for broadcasting parameter for (a) `Webspam` and (b) `Epsilon` Datasets

convergence, is lower than the two others, but the difference is not significant. Also it is to be noted that the difference in the convergence speed can become even larger if some of the machines are extremely overloaded, which is generally the case in the cluster environments.

6.4.2.3 Communication Overhead

We also present the communication overhead incurred by each of the methods. The total communication cost for each algorithm is compared in terms of the total number

Methods	Webspam		Epsilon		RCV	
	N_c	time (s)	N_c	time (s)	N_c	time (s)
Sync-SVRG	336021	635.44	108009	589.9	83711	4756.25
Async-SVRG	839831	42.13	108000	110.03	30701	733.47
ADSVRG	**33611**	**14.93**	**12004**	**29.6**	**8380**	**631.92**

Table 6.2: Comparison of the communication overhead of all approaches on the three collections

of communication calls (send, receive, broadcast, gather), as well as the time spent in those calls. Since for Sync-SVRG and Async-SVRG methods the convergence is very slow near the tail, we compare the communication cost till the iteration when all methods achieve the same minimization of the objective function. Table 6.2 shows the detailed results obtained for each algorithm on all datasets. It can be observed that the *ADSVRG* incurs the minimum communication overhead as the number of communication between the machines is very low. Most of the calls shown for *ADSVRG* are made during the first epoch where the gradients are synchronized. Whereas Sync-SVRG and Async-SVRG methods have to communicate large number of times in order to broadcast their local gradients to the master and receive the updated parameters from the master machine. Since Sync-SVRG is totally synchronous, the communication calls are blocking in nature and hence considerably slow. Whereas for Async-SVRG and *ADSVRG* the communication calls are mostly non-blocking and hence return immediately.

6.4.2.4 Speedup Result with Increasing Number of Workers

Finally, we evaluate the scalability of the proposed framework with respect to the increasing number of worker machines. In this experiment, we vary the number of workers from 5 to 25, each time increasing the number of workers by 5. Figure 6.4 illustrates the evolution of the loss function of *ADSVRG* on the training set (a) as well as the test accuracy (b) with respect to time (in seconds) on the Epsilon collection. Figure 6.4 (c) also depicts the speedup in convergence time with respect to the number of workers. In the ideal case, shown in red, when the number of workers double, the convergence time is divided by two; and hence the speedup is linear. From this figure, it comes out that as the number of workers increases the *ADSVRG* algorithm is able to achieve a near linear speedup, which is mainly due to the fact that, it relies on

very low communication between the workers which is is also shown in Table 6.2. On the other hand, as the number of workers increases the performance of the algorithm slightly deteriorates.

6.5 Closing Remarks

In this chapter, we analyzed the binary classification problem. We started with the introduction and the types of binary classification methods. Then we discussed the different optimization functions for linear binary classification. In the related work section we summarized the different ways of perfoming optimization for linear methods. Hence to tackle the problem of large-scale binary classification we used the proposed asynchronous distributed framework for linear binary classification algorithm. The results suggest improvement in the convergence time with the use of proposed framework. We also demonstrated the scalability of the method for large-scale binary classification datasets. Hence, the proposed framework shows potential for binary classification application.

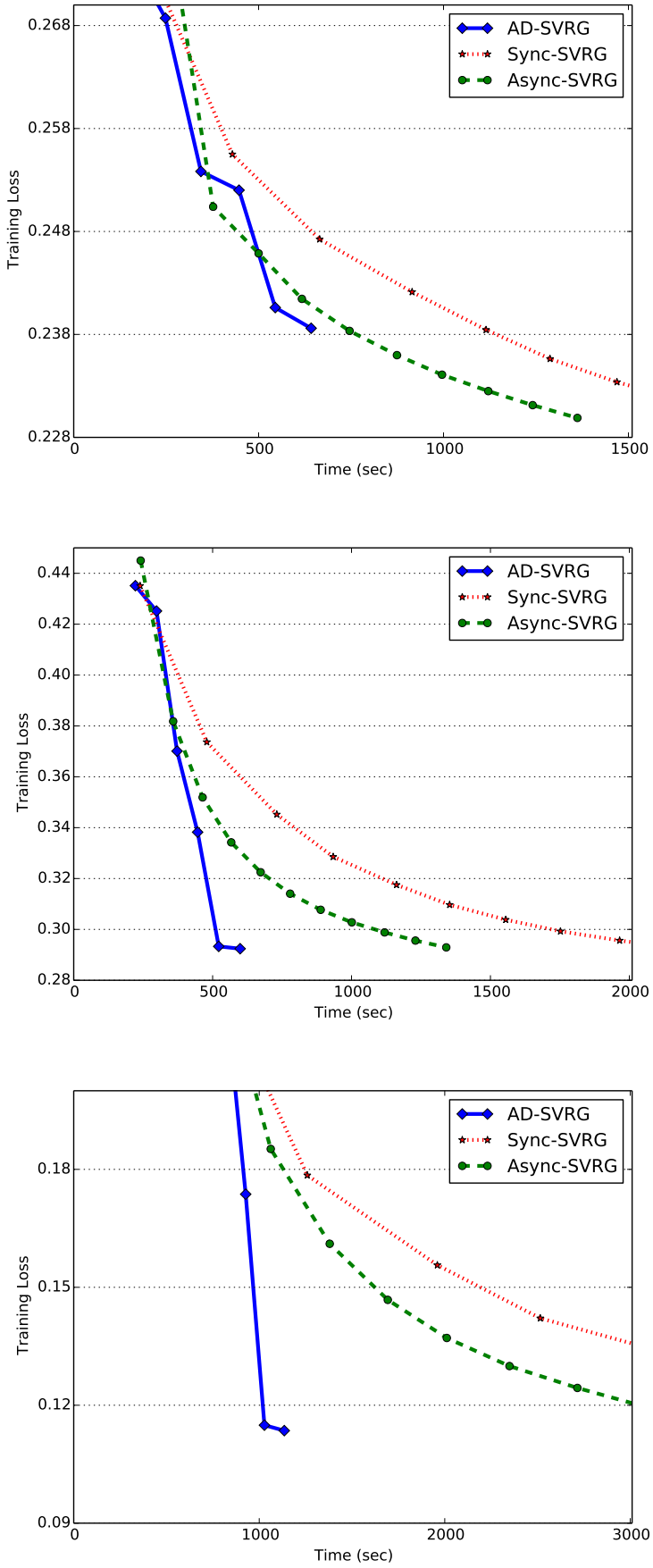


Figure 6.2: Training Loss Vs Time Plot for (a) `Webspam`, (b) `Epsilon` and (c) `RCV` Datasets

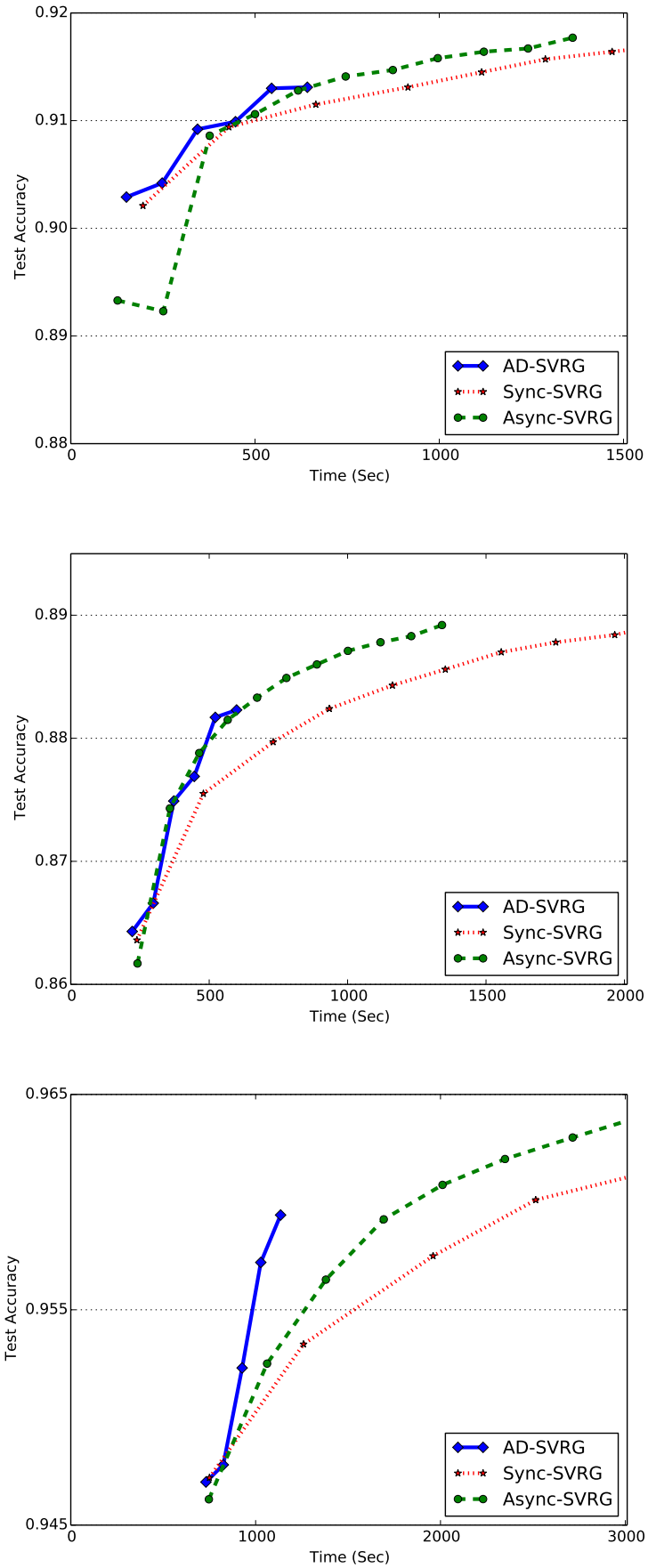


Figure 6.3: Test Accuracy Vs Time Plot for (a) `Webspam`, (b) `Epsilon` and (c) `RCV` Datasets

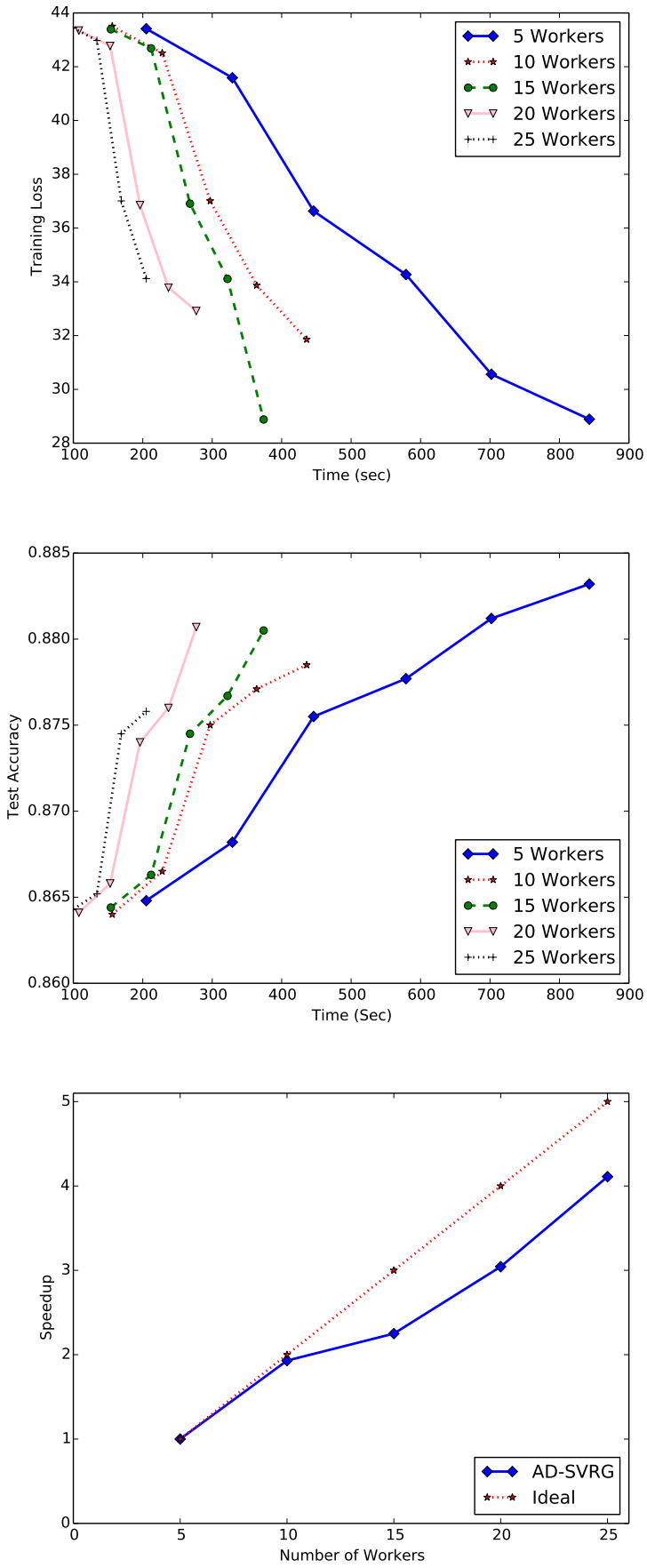


Figure 6.4: Convergence Speedup Result for `Epsilon` Dataset

CONCLUSION

The goal of this thesis was to explore ML algorithms for large-scale data and devise efficient algorithms. To achieve this objective we studied this problem from two different perspectives. First, we considered the problem of large-scale multiclass classification. Then we analyzed the distributed algorithms for ML. In both the areas we discussed the challenges of handling large-scale data and proposed efficient algorithms to solve them. We also showed a detailed theoretical and empirical analysis of the proposed algorithms to validate the claims. Analysis of both the problems constitute the two parts of this thesis.

In the first part of the thesis, we study the multiclass classification problem. We proposed an algorithm for the reduction of multiclass classification to a binary classification problem. In Chapter 2, we began with the introduction to multiclass classification. Then, we presented various state-of-the-art algorithms to solve multiclass classification problems. We noticed that there is no single method which is superior to every other methods. Each of the methods has their advantages and drawbacks. We also discussed about the popular challenges associated with multiclass classification, typically for large-class scenario.

To solve the discussed challenges, we introduced our first reduction algorithm referred to as Naive reduction algorithm (`mRb`) in Chapter 3. The reduction technique is based on binarization of the original problem considering pairwise subtraction of joint representations of examples and classes. The preliminary results showed the efficacy of the reduction technique in solving the popular challenges of multiclass classification. However, we noticed that the proposed algorithm introduced some new challenges. The pairwise subtraction of all possible pairs is a tedious task if the number of classes is very large. First of all it takes tediously long time for extraction of joint features and the subtraction task. Secondly, it results in large number of examples in the reduced binary dataset. Also, during prediction phase, we need to create joint representation of each test example with respect to all the classes. These problems

significantly magnify as the number of classes increase.

Hence, to overcome these new challenges, we extended the Naive reduction algorithm and proposed the extended algorithm referred as Double-Sampled multi to binary reduction (DS-mRb) algorithm. We made extensions in two problematic areas of the Naive algorithm. First, we incorporated a double sampling strategy during the reduction phase. First sampling is with respect to the number of examples considered from each class. Secondly, we randomly sample classes to consider for making joint representation and subtraction. Using such double sampling significantly improved the runtime of the reduction process as well as the total memory used for keeping the reduced examples. The second extension is with respect to the prediction phase, where we pre-select a small subset of candidates to consider for the final prediction. This helped to make the prediction phase faster.

The performance of the proposed method was validated using popular datasets for text classification application. Also, we compared the results with a number of popular baseline methods. The analysis of the result shows the efficacy of the proposed algorithm. The comparison helped us to conclude that the proposed method is the best performance compromise considering various aspects of evaluation such as the total runtime, memory usage and predictive performance. Additionally, we presented a detailed theoretical analysis of the proposed model. Our reduction strategy brings inter-dependency between the pairs containing the same observation and its true class in the original training set. Thus, we derive new generalization bounds using local fractional Rademacher complexity showing that even with a shift in the original class distribution and also the inter-dependency between the pairs of example, the empirical risk minimization principle over the transformation of the sampled training set remains consistent.

In this part of thesis we showed effectiveness of the proposed double-sampled multi to binary reduction (DS-mRb) algorithm for large-class multiclass classification. This work opens several research directions. First, even though this algorithm works very well for text classification, its performance on other applications is still untested. This remains as an open question for potential future works. However, finding meaningful joint features can be a challenging task in many multiclass classification application. Another future research direction would be to extend this approach to handle multi-label classification, where one example may belong to more than one classes at once. In such problems, it will be interesting to incorporate the label dependencies in the algorithm.

In the second part of the thesis, we studied distributed approaches for performing ML optimization. In distributed computing, we partition the data across several machines and simultaneously perform the learning task. We consider the scenarios where the memory is not shared between the machines. In such setting, the main consideration is to minimize the communication between the machines and to avoid the bottleneck of synchronization between the machines. We introduced a framework which overcomes these two challenges. The proposed framework is based on asynchronous distributed optimization. We showed the effectiveness of the framework by considering two applications: matrix factorization for recommender system and large-scale binary classification.

In Chapter 4, we introduced distributed ML. We began by discussing different settings of distributed computing, their desired properties and popularly used distributed computing tools. Later, we formulated the problem structure that we considered throughout the second part of the thesis. Then, we presented our framework for asynchronous distributed machine learning based on averaging of parameters and showed the proof of convergence in this setting. This chapter introduced the proposed framework, which opened the door for utilizing it for ML applications.

In Chapter 5, we used the proposed asynchronous distributed framework for distributing matrix factorization in Recommender System application. We began with introducing the background of Recommender systems and popularly used methods in this domain. We focused on matrix factorization, since it is one of the most popularly used methods. Additionally, we introduced an extra regularization term to incorporate the user and item similarity in the error function of SGD for matrix factorization. The use of neighbourhood information promised to give improvement in the performance of SGD method. Also, in the experiment section we demonstrated the benefits of using asynchronous framework over the synchronization based baselines.

Finally, in Chapter 6, we performed the distributed optimization of binary classification algorithm using the proposed asynchronous distributed framework. In this chapter, we started with the introduction and the types of binary classification methods. Then we discussed different loss functions popularly used for linear binary classification. We presented different optimization schemes for binary classification. Hence to tackle the problem of large-scale binary classification we used the proposed asynchronous distributed framework for linear binary classification algorithm. The results suggest improvement in the convergence time with the use of proposed framework. We also demonstrated the scalability of the method for large-scale binary classifica-

tion datasets. Hence, the proposed framework shows potential for binary classification application.

The results in this part of thesis give an indication for a potential research direction. With the popularity of big data and distributed frameworks such as Spark and MPI, asynchronous communication is already an important consideration and will be very essential for applications in future. Because of time and infrastructure constraints we could not analyze various aspects of this research, which leads to several open questions and potential future extensions of this thesis. One interesting direction would be to test such framework in industrial scale applications with large clusters consisting of thousands of machines. However, in such scale the main challenge will be to control the communication frequency to utilize the network bandwidth optimally.

PUBLICATIONS

1. B. Joshi, M.R. Amini, I. Partalas, F. Iutzeler, Y. Maximov, Aggressive Sampling for Multi-class to Binary Reduction with Applications to Text Classification, NIPS 2017

2. B. Joshi, F. Iutzeler, M.R. Amini. An Asynchronous Distributed Framework for Large-scale Learning Based on Parameter Exchanges. International Journal of Data Science and Analytics, IJDSA (Under Review)

3. B. Joshi, F. Iutzeler, M.R. Amini. Asynchronous Distributed Matrix Factorization with Similar User and Item Based Regularization. Proceedings of the 10th ACM Conference on Recommender Systems (RecSys'16), Boston, USA, 2016.

4. B Joshi, MR Amini, I Partalas, L Ralaivola, N Usunier, E Gaussier. On Binary Reduction of Large-Scale Multiclass Classification Problems. International Symposium on Intelligent Data Analysis, (IDA), 2015.

5. B Joshi, MR Amini, I Partalas, L Ralaivola, N Usunier, E Gaussier. Multi-class to Binary reduction of Large-scale classification Problems. International Workshop on Big Multi-Target Prediction ECML/PKDD 2015.

BIBLIOGRAPHY

[Abu-El-Haija et al., 2016] Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B., and Vijayanarasimhan, S. (2016). Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*.

[Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749.

[Allwein et al., 2000] Allwein, E. L., Schapire, R. E., and Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of machine learning research*, 1(Dec):113–141.

[Aly, 2005] Aly, M. (2005). Survey on multiclass classification methods. *Neural Netw*, pages 1–9.

[Babbar et al., 2014a] Babbar, R., Metzig, C., Partalas, I., Gaussier, E., and Amini, M.-R. (2014a). On power law distributions in large-scale taxonomies. *ACM SIGKDD Explorations Newsletter*, 16(1):47–56.

[Babbar et al., 2014b] Babbar, R., Metzig, C., Partalas, I., Gaussier, E., and Amini, M. R. (2014b). On power law distributions in large-scale taxonomies. *SIGKDD Explorations*, 16(1).

[Balasubramanian and Lebanon, 2012] Balasubramanian, K. and Lebanon, G. (2012). The landmark selection method for multiple output prediction. *arXiv preprint arXiv:1206.6479*.

[Bauschke and Combettes, 2011] Bauschke, H. H. and Combettes, P. L. (2011). *Convex analysis and monotone operator theory in Hilbert spaces*. Springer Science & Business Media.

[Bengio et al., 2010] Bengio, S., Weston, J., and Grangier, D. (2010). Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, pages 163–171.

[Beygelzimer et al., 2009a] Beygelzimer, A., Langford, J., Lifshits, Y., Sorkin, G., and Strehl, A. (2009a). Conditional probability tree estimation analysis and algorithms. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 51–58. AUAI Press.

[Beygelzimer et al., 2009b] Beygelzimer, A., Langford, J., and Ravikumar, P. (2009b). Error-correcting tournaments. In *Proceedings of the 20th International Conference on Algorithmic Learning Theory*, ALT'09, pages 247–262.

[Bhatia et al., 2015] Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. (2015). Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pages 730–738.

[Bi and Kwok, 2013] Bi, W. and Kwok, J. (2013). Efficient multi-label classification with many labels. In *International Conference on Machine Learning*, pages 405–413.

[Bishop, 2006] Bishop, C. M. (2006). Pattern recognition. *Machine Learning*, 128:1–58.

[Bobadilla et al., 2013] Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems*, 46:109–132.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

[Bottou, 2010] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[Boyd et al., 2011] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122.

[Cauchy, 1847] Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538.

[Chang et al., 2015] Chang, T., Hong, M., Liao, W., and Wang, X. (2015). Asynchronous distributed ADMM for large-scale optimization- part I: algorithm and convergence analysis. *ArXiv e-prints*, 1509.02597.

[Chen and Lin, 2012] Chen, Y.-N. and Lin, H.-T. (2012). Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*, pages 1529–1537.

[Chin et al., 2015] Chin, W.-S., Zhuang, Y., Juan, Y.-C., and Lin, C.-J. (2015). A learning-rate schedule for stochastic gradient methods to matrix factorization. In *PAKDD (1)*, pages 442–455.

[Choromanska et al., 2013] Choromanska, A., Agarwal, A., and Langford, J. (2013). Extreme multi class classification. In *NIPS Workshop: eXtreme Classification, submitted*.

[Choromanska and Langford, 2015] Choromanska, A. and Langford, J. (2015). Logarithmic time online multiclass prediction. In *NIPS*.

[Cinbis et al., 2012] Cinbis, R. G., Verbeek, J., and Schmid, C. (2012). Image categorization using fisher kernels of non-iid image models. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2184–2191. IEEE.

[Cisse et al., 2013] Cisse, M. M., Usunier, N., Artieres, T., and Gallinari, P. (2013). Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*, pages 1851–1859.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Cramer, 2002] Cramer, J. S. (2002). The origins of logistic regression.

[Crammer and Singer, 2002] Crammer, K. and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292.

[Daume III et al., 2016] Daume III, H., Karampatziakis, N., Langford, J., and Mineiro, P. (2016). Logarithmic time one-against-some. *arXiv preprint arXiv:1606.04988*.

[Dean et al., 2012] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[Defazio et al., 2014] Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 1646–1654.

[Deng et al., 2010] Deng, J., Berg, A., Li, K., and Fei-Fei, L. (2010). What does classifying more than 10,000 image categories tell us? *Computer Vision–ECCV 2010*, pages 71–84.

[Diaz et al., 2012] Diaz, J., Munoz-Caro, C., and Nino, A. (2012). A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on parallel and distributed systems*, 23(8):1369–1386.

[Dietterich and Bakiri, 1995] Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286.

[Fan et al., 2008a] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008a). Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874.

[Fan et al., 2008b] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008b). Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

[Ferng and Lin, 2011] Ferng, C.-S. and Lin, H.-T. (2011). Multi-label classification with error-correcting codes. In *Asian Conference on Machine Learning*, pages 281–295.

[Fleury et al., 2013] Fleury, A., Noury, N., and Vacher, M. (2013). Improving supervised classification of activities of daily living using prior knowledge. *Digital Advances in Medicine, E-Health, and Communication Technologies*, page 131.

[Friedman, 1996] Friedman, J. (1996). Another approach to polychotomous classification. Technical report, Technical report, Department of Statistics, Stanford University.

[Garcia-Pedrajas and Ortiz-Boyer, 2006] Garcia-Pedrajas, N. and Ortiz-Boyer, D. (2006). Improving multiclass pattern recognition by the combination of two strategies. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(6):1001–1006.

[Gemulla et al., 2011] Gemulla, R., Nijkamp, E., Haas, P. J., and Sismanis, Y. (2011). Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM.

[Guo et al., 2016] Guo, Y., Zhang, L., Hu, Y., He, X., and Gao, J. (2016). Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *European Conference on Computer Vision*, pages 87–102. Springer.

[Hall et al., 2010] Hall, K. B., Gilpin, S., and Mann, G. (2010). Mapreduce/bigtable for distributed optimization. In *NIPS LCCC Workshop*.

[Herlocker et al., 1999] Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM.

[Ho et al., 2013] Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J. K., Gibbons, P. B., Gibson, G. A., Ganger, G., and Xing, E. P. (2013). More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems 26*, pages 1223–1231.

[Hsu and Lin, 2002] Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425.

[Hsu et al., 2009] Hsu, D. J., Kakade, S. M., Langford, J., and Zhang, T. (2009). Multi-label prediction via compressed sensing. In *Advances in neural information processing systems*, pages 772–780.

[Hu, 2011] Hu, R. (2011). Active learning for text classification.

[Huo and Huang, 2016] Huo, Z. and Huang, H. (2016). Asynchronous stochastic gradient descent with variance reduction for non-convex optimization. *arXiv preprint arXiv:1604.03584*.

[Jaggi et al., 2014] Jaggi, M., Smith, V., Takác, M., Terhorst, J., Krishnan, S., Hofmann, T., and Jordan, M. I. (2014). Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076.

[Jain et al., 2016] Jain, H., Prabhu, Y., and Varma, M. (2016). Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944. ACM.

[Janson, 2004] Janson, S. (2004). Large deviations for sums of partly dependent random variables. *Random Structures and Algorithms*, 24(3):234–248.

[Jasinska and Karampatziakis, 2016] Jasinska, K. and Karampatziakis, N. (2016). Log-time and log-space extreme classification. *arXiv preprint arXiv:1611.01964*.

[Johnson and Zhang, 2013] Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323.

[Joshi et al., 2015a] Joshi, B., Amini, M., Partalas, I., Ralaivola, L., Usunier, N., and Gaussier, É. (2015a). On binary reduction of large-scale multiclass classification problems. In *Advances in Intelligent Data Analysis XIV - 14th International Symposium, IDA 2015*, pages 132–144.

[Joshi et al., 2017] Joshi, B., Amini, M.-R., Partalas, I., Iutzeler, F., and Maximov, Y. (2017). Aggressive sampling for multi-class to binary reduction with applications to text classification. *arXiv preprint arXiv:1701.06511*.

[Joshi et al., 2015b] Joshi, B., Amini, M.-R., Partalas, I., Ralaivola, L., Usunier, N., and Gaussier, E. (2015b). On binary reduction of large-scale multiclass classification problems. In *International Symposium on Intelligent Data Analysis*, pages 132–144. Springer.

[Kang et al., 2015] Kang, S. J., Lee, S. Y., and Lee, K. M. (2015). Performance comparison of openmp, mpi, and mapreduce in practical problems. *Advances in Multimedia*, 2015:7.

[Kapoor et al., 2012] Kapoor, A., Viswanathan, R., and Jain, P. (2012). Multilabel classification using bayesian compressed sensing. In *Advances in Neural Information Processing Systems*, pages 2645–2653.

[Koren et al., 2009] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, (8):30–37.

[Langford et al., 2009] Langford, J., Smola, A. J., and Zinkevich, M. (2009). Slow learners are fast. *Advances in Neural Information Processing Systems*, 22:2331–2339.

[Leblond et al., 2016] Leblond, R., Pedregosa, F., and Lacoste-Julien, S. (2016). Asaga: Asynchronous parallel saga. *arXiv preprint arXiv:1606.04809*.

[Lehmann, 1975] Lehmann, E. (1975). *Nonparametric Statistical Methods Based on Ranks*. McGraw-Hill, New York, USA.

[Li, 2017] Li, M. (2017). *Scaling Distributed Machine Learning with System and Algorithm Co-design*. PhD thesis, Intel.

[Liu, 2011] Liu, T.-Y. (2011). *Learning to rank for information retrieval*. Springer Science & Business Media.

[Liu et al., 2009] Liu, T.-Y. et al. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331.

[Liu et al., 2007] Liu, T.-Y., Xu, J., Qin, T., Xiong, W., and Li, H. (2007). Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*, pages 3–10.

[Lorena et al., 2008] Lorena, A. C., Carvalho, A. C., and Gama, J. a. M. (2008). A review on the combination of binary classifiers in multiclass problems. *Artif. Intell. Rev.*, 30(1-4):19–37.

[Mairal, 2015] Mairal, J. (2015). Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25(2):829–855.

[Makari et al., 2015] Makari, F., Teflioudi, C., Gemulla, R., Haas, P., and Sismanis, Y. (2015). Shared-memory and shared-nothing stochastic gradient descent algorithms for matrix completion. *Knowledge and Information Systems*, 42(3):493–523.

[Manning et al., 2008] Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

[Maurer and Pontil, 2009] Maurer, A. and Pontil, M. (2009). Empirical bernstein bounds and sample variance penalization. *arXiv preprint arXiv:0907.3740*.

[McDiarmid, 1989] McDiarmid, C. (1989). On the method of bounded differences. *Survey in Combinatorics*, pages 148–188.

[McDonald et al., 2010] McDonald, R., Hall, K., and Mann, G. (2010). Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464. Association for Computational Linguistics.

[Mohri and Rostamizadeh, 2009] Mohri, M. and Rostamizadeh, A. (2009). Rademacher complexity bounds for non-i.i.d. processes. In *Advances in Neural Information Processing Systems 21*, pages 1097–1104.

[Park and Fürnkranz, 2014] Park, S. and Fürnkranz, J. (2014). Efficient implementation of class-based decomposition schemes for naïve bayes. *Machine Learning*, 96(3):295–309.

[Park and Fürnkranz, 2012] Park, S.-H. and Fürnkranz, J. (2012). Efficient prediction algorithms for binary decomposition techniques. *Data Mining and Knowledge Discovery*, 24(1):40–77.

[Parkhi et al., 2015] Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep face recognition. In *BMVC*, volume 1, page 6.

[Partalas et al., 2015] Partalas, I., Kosmopoulos, A., Baskiotis, N., Artieres, T., Paliouras, G., Gaussier, E., Androutsopoulos, I., Amini, M.-R., and Galinari, P. (2015). LSHTC: A Benchmark for Large-Scale Text Classification. *ArXiv e-prints*.

[Pazzani and Billsus, 2007] Pazzani, M. and Billsus, D. (2007). Content-based recommendation systems. *The adaptive web*, pages 325–341.

[Perronnin et al., 2010] Perronnin, F., Sénchez, J., and Xerox, Y. L. (2010). Large-scale image categorization with explicit data embedding. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2297–2304. IEEE.

[Prabhu and Varma, 2014] Prabhu, Y. and Varma, M. (2014). Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–272. ACM.

[Qin et al., 2010] Qin, T., Liu, T.-Y., Xu, J., and Li, H. (2010). Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374.

[Raghavan and Wong, 1986] Raghavan, V. V. and Wong, S. M. (1986). A critical analysis of vector space model for information retrieval. *Journal of the American Society for information Science*, 37(5):279.

[Ralaivola and Amini, 2015] Ralaivola, L. and Amini, M. (2015). Entropy-based concentration inequalities for dependent variables. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2436–2444.

[Ralaivola et al., 2010] Ralaivola, L., Szafranski, M., and Stempfel, G. (2010). Chromatic PAC-Bayes Bounds for Non-IID Data: Applications to Ranking and Stationary β-Mixing Processes. *Journal of Machine Learning Research*, 11:1927–1956.

[Recht et al., 2011] Recht, B., Re, C., Wright, S., and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701.

[Reddi et al., 2015] Reddi, S. J., Hefny, A., Sra, S., Poczos, B., and Smola, A. J. (2015). On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pages 2647–2655.

[Resnick and Varian, 1997] Resnick, P. and Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3):56–58.

[Rifkin and Klautau, 2004] Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classification. *Journal of machine learning research*, 5(Jan):101–141.

[Rocha and Goldenstein, 2014] Rocha, A. and Goldenstein, S. K. (2014). Multiclass from binary: Expanding one-versus-all, one-versus-one and ecoc-based approaches. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):289–302.

[Roux et al., 2012] Roux, N. L., Schmidt, M., and Bach, F. R. (2012). A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671.

[Schapire and Singer, 1999] Schapire, R. E. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336.

[Schiolkopf et al., 1995] Schiolkopf, B., Burges, C., and Vapnik, V. (1995). Extracting support data for a given task. In *Proceedings, First International Conference on Knowledge Discovery & Data Mining. AAAI Press, Menlo Park, CA*, pages 252–257.

[Shalev-Shwartz and Zhang, 2013] Shalev-Shwartz, S. and Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599.

[Shamir et al., 2014] Shamir, O., Srebro, N., and Zhang, T. (2014). Communication-efficient distributed optimization using an approximate newton-type method. In *ICML*, volume 32, pages 1000–1008.

[Snir, 1998] Snir, M. (1998). *MPI–the Complete Reference: The MPI core*, volume 1. MIT press.

[Sokolova and Lapalme, 2009] Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437.

[Sra, 2012] Sra, S. (2012). Scalable nonconvex inexact proximal splitting. In *Advances in Neural Information Processing Systems 25*, pages 530–538.

[Steinwart and Christmann, 2010] Steinwart, I. and Christmann, A. (2010). Fast learning from non-i.i.d. observations. In *Advances in Neural Information Processing Systems 22*, pages 1768–1776.

[Tai and Lin, 2012] Tai, F. and Lin, H.-T. (2012). Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542.

[Titsias, 2016] Titsias, M. (2016). One-vs-each approximation to softmax for scalable estimation of probabilities. In *Advances in Neural Information Processing Systems 29*, pages 4161–4169.

[Usunier et al., 2005] Usunier, N., Amini, M., and Gallinari, P. (2005). Generalization error bounds for classifiers trained with interdependent data. In *Advances in Neural Information Processing Systems 18 (NIPS)*, pages 1369–1376.

[Usunier et al., 2006] Usunier, N., Amini, M. R., and Gallinari, P. (2006). Generalization error bounds for classifiers trained with interdependent data. In *Advances in Neural Information Processing Systems 18*, pages 1369–1376.

[Van Rijsbergen, 1979] Van Rijsbergen, C. (1979). Information retrieval. dept. of computer science, university of glasgow. *URL: citeseer. ist. psu. edu/vanrijsbergen79information. html*, 14.

[Vondrick et al., 2013] Vondrick, C., Patterson, D., and Ramanan, D. (2013). Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, 101(1):184–204.

[Weston et al., 2011a] Weston, J., Bengio, S., and Usunier, N. (2011a). Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770.

[Weston et al., 2011b] Weston, J., Bengio, S., and Usunier, N. (2011b). Wsabie: Scaling up to large vocabulary image annotation. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*.

[Weston et al., 2013] Weston, J., Makadia, A., and Yee, H. (2013). Label partitioning for sublinear ranking. In *ICML (2)*, pages 181–189.

[Weston and Watkins, 1998] Weston, J. and Watkins, C. (1998). Multi-class support vector machines. Technical report, Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London.

[Xu and Yin, 2014] Xu, Y. and Yin, W. (2014). A globally convergent algorithm for nonconvex optimization based on block coordinate update. *ArXiv e-prints:1410.1386*.

[Yen et al., 2016] Yen, I. E., Huang, X., Zhong, K., Ravikumar, P., and Dhillon, I. S. (2016). Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification. In *Proceedings of the 33nd International Conference on Machine Learning*.

[Yu et al., 2014a] Yu, H.-F., Jain, P., Kar, P., and Dhillon, I. (2014a). Large-scale multi-label learning with missing labels. In *International Conference on Machine Learning*, pages 593–601.

[Yu et al., 2014b] Yu, Z.-Q., Shi, X.-J., Yan, L., and Li, W.-J. (2014b). Distributed stochastic admm for matrix factorization. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1259–1268. ACM.

[Yuan et al., 2012] Yuan, G.-X., Ho, C.-H., and Lin, C.-J. (2012). Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603.

[Zhang and Kwok, 2014] Zhang, R. and Kwok, J. T. (2014). Asynchronous distributed admm for consensus optimization. In *ICML*, pages 1701–1709.

[Zhang et al., 2015] Zhang, R., Zheng, S., and Kwok, J. T. (2015). Fast distributed asynchronous sgd with variance reduction. *CoRR, abs/1508.01633*.

[Zhang and Schneider, 2011] Zhang, Y. and Schneider, J. G. (2011). Multi-label output codes using canonical correlation analysis. In *AISTATS*, pages 873–882.

[Zhao and Li, 2016] Zhao, S.-Y. and Li, W.-J. (2016). Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *AAAI*, pages 2379–2385.

[Zhu and Marcotte, 1996] Zhu, D. L. and Marcotte, P. (1996). Co-coercivity and its role in the convergence of iterative schemes for solving variational inequalities. *SIAM Journal on Optimization*, 6(3):714–726.

[Zhuang et al., 2013] Zhuang, Y., Chin, W.-S., Juan, Y.-C., and Lin, C.-J. (2013). A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 249–256. ACM.

[Zinkevich et al., 2010] Zinkevich, M., Weimer, M., Li, L., and Smola, A. J. (2010). Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603.