BIG DATA ALGORITHMS FOR VISUALIZATION AND SUPERVISED LEARNING

A Dissertation
Submitted to
the Temple University Graduate Board

In Partial Fulfillment
of the Requirements for the Degree of
DOCTOR OF PHILOSOPHY

---

by
Nemanja Djuric
January 2014

Examining committee members:

Dr Slobodan Vucetic, Advisory Chair, Dept. of Computer and Information Sciences
Dr Zoran Obradovic, Department of Computer and Information Sciences
Dr Longin Jan Latecki, Department of Computer and Information Sciences
Dr Li Bai, External Member, Department of Electrical and Computer Engineering

ABSTRACT

Explosive growth in data size, data complexity, and data rates, triggered by emergence of high-throughput technologies such as remote sensing, crowd-sourcing, social networks, or computational advertising, in recent years has led to an increasing availability of data sets of unprecedented scales, with billions of high-dimensional data examples stored on hundreds of terabytes of memory. In order to make use of this large-scale data and extract useful knowledge, researchers in machine learning and data mining communities are faced with numerous challenges, since the data mining and machine learning tools designed for standard desktop computers are not capable of addressing these problems due to memory and time constraints. As a result, there exists an evident need for development of novel, scalable algorithms for big data.

In this thesis we address these important problems, and propose both supervised and unsupervised tools for handling large-scale data. First, we consider unsupervised approach to big data analysis, and explore scalable, efficient visualization method that allows fast knowledge extraction. Next, we consider supervised learning setting and propose algorithms for fast training of accurate classification models on large data sets, capable of learning state-of-the-art classifiers on data sets with millions of examples and features within minutes.

Data visualization have been used for hundreds of years in scientific research, as it allows humans to easily get a better insight into complex data they are studying. Despite its long history, there is a clear need for further development of visualization

methods when working with large-scale, high-dimensional data, where commonly used visualization tools are either too simplistic to gain a deeper insight into the data properties, or are too cumbersome or computationally costly. We present a novel method for data ordering and visualization. By combining efficient clustering using k-means algorithm and near-optimal ordering of found clusters using state-of-the-art TSP-solver, we obtain efficient algorithm that achieves performance better than existing, computationally intensive methods. In addition, we present visualization method for smaller-scale problems based on object matching. The experiments show that the methods allow for fast detection of hidden patterns, even by users without expertise in the areas of data mining and machine learning.

Supervised learning is another important task, often intractable in many modern applications due to time and memory constraints, considering prohibitively large scales of the data sets. To address this issue, we first consider Multi-hyperplane Machine (MM) classification model, and propose online Adaptive MM algorithm which represents a trade-off between linear and kernel Support Vector Machines (SVMs), as it trains MMs in linear time on limited memory while achieving competitive accuracies on large-scale non-linear problems. Moreover, we present a C++ toolbox for developing scalable classification models, which provides an Application Programming Interface (API) for training of large-scale classifiers, as well as highly-optimized implementations of several state-of-the-art SVM approximators. Lastly, we consider parallelization and distributed learning approaches to large-scale supervised learning, and propose AROW-MapReduce, a distributed learning algorithm for confidence-weighted models using MapReduce framework. Experimental evaluation of the proposed methods shows state-of-the-art performance on a number of synthetic and real-world data sets, further paving a way for efficient and effective knowledge extraction from big data problems.

To my parents Borka and Slavko, my sister Tijana, brother Stefan.

To Djuric, Cvijanovic, Savic families.

To Mitzi.

This dissertation is a result of their continuous love and support.

In loving memory of my grandparents

Ljubica and Milos Cvijanovic,

Zorka and Ilija Djuric.

ACKNOWLEDGEMENTS

I am deeply grateful and indebted to my advisor Dr Slobodan Vucetic for introducing me to the world of data mining and machine learning, for keeping my attention to the state-of-the-art research directions, for all his patience, invaluable advice and continuous support and encouragement during my graduate studies.

Special thanks to Dr Zoran Obradovic, a great professor and researcher from whom I learned a lot. I would also like to thank Dr Longin Jan Latecki and Dr Li Bai for serving on my dissertation committee and for providing me with useful comments and valuable suggestions.

During my graduate studies I was fortunate enough to work as a teaching assistant for Prof. Longin Jan Latecki, Prof. John Fiore, Prof. Wendy Urban, and Prof. Abbe Forman. Their experience and enthusiasm about the subjects being taught were of great inspiration throughout my years of teaching. I am also like to thankful to all my students at the Department of Computer and Information Sciences.

I thank professors and staff at the Department of Computer and Information Sciences for making such a friendly and supportive working environment. I would especially like to thank Ruth Briggs, for all the help, encouragement, and chocolate chip cookies that a graduate student may ever need, and professor Justin Y. Shi, for always being there for us graduate students.

I had a great opportunity to work closely with bright colleagues from Dr Slobodan Vucetic's lab, Mihajlo Grbovic, Vuk Malbasa, Zhuang Wang, Vladimir Coric, Liang

Lan, Ana Milicic, Yi Jia, Shanshan Zhang, and Lakesh Kansakar. Thank you all for a prosperous collaboration, useful and motivating discussions during our lab meetings.

Special thanks go to my friends at the Department of Computer and Information Sciences, Mihajlo Grbovic, Vladan Radosavljevic, Xin Li, Kristiyan Georgiev, Feipeng Zhao, Joseph Jupin, Xue Wei, Min Xiao, Meng Yi, Qingqing Cai, Moussa Taifi, Vladimir Ouzienko, Kosta Ristovski, Vuk Malbasa, Uros Midic, Vladimir Coric, Dusan Ramljak, Zhuang Wang, Mohamed Ghalwash, Suicheng Gu, Gregory Johnson, Solomon Jones, and many others, who I may have forgotten to mention, for making my life at Temple University pleasant and enjoyable.

I would also like to thank all my friends back home in Serbia and here in Philadelphia for being with me throughout all these years.

Lastly, I would like to acknowledge support from the National Science Foundation during my graduate studies, through its grants IIS-0546155 and IIS-1117433.

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGEMENTS vi

LIST OF TABLES xii

LIST OF FIGURES xiv

1 INTRODUCTION 1

1.1 Data Visualization 2

1.2 Learning classification models on large-scale data 3

2 VISUALIZATION THROUGH MATRIX REORDERING 6

2.1 Introduction 6

2.2 Background 9

2.2.1 Data visualization 10

2.2.2 Data reordering 11

2.2.3 Traveling salesman problem (TSP) 13

2.3 Methodology 15

2.3.1 Differential Predictive Coding (DPC) 15

2.3.2 Relationship between entropy minimization and ordering . . . 16

2.3.3 Reordering for entropy minimization 18

2.3.4 Feature scaling 18

2.3.5 TSP-means algorithm 19

2.3.6 Further details . . . 22
2.4 Experiments . . . 24
2.4.1 Validation of TSP-means . . . 24
2.4.2 Validation of EM-ordering . . . 28
2.4.3 Applications of EM-ordering . . . 30
3 VISUALIZATION THROUGH OBJECT MATCHING 33
3.1 Introduction . . . 33
3.2 Kernelized Sorting . . . 35
3.3 Convex Kernelized Sorting . . . 37
3.3.1 Numerical optimization . . . 40
3.3.2 Soft and hard assignments . . . 42
3.4 Experiments . . . 43
3.4.1 Data visualization . . . 44
3.4.2 Image alignment . . . 45
3.4.3 Multilingual document alignment . . . 47
4 ONLINE LEARNING OF MULTI-HYPERPLANE CLASSIFICATION MODELS 50
4.1 Introduction . . . 50
4.2 Preliminaries . . . 54
4.2.1 Multi-Class SVM . . . 54
4.2.2 Multi-Hyperplane Machine . . . 55
4.2.3 MM Training . . . 56
4.3 Adaptive Multi-Hyperplane Machine (AMM) . . . 58
4.3.1 Solving the Sub-Problem (4.7) . . . 58
4.3.2 Number of Weights . . . 60
4.3.3 Generalization Error . . . 61

4.3.4 Weight Pruning 62
4.3.5 Online AMM 65
4.3.6 Implementation Details 66
4.4 Experiments 67
4.5 Improving the representability of AMM algorithm 72
4.5.1 Connection between MM and LVQ models 73
4.5.2 Growing AMM (GAMM) algorithm 75
4.5.3 Preliminary results 78
5 BUDGETEDSVM: A TOOLBOX FOR SCALABLE SVM APPROXIMATIONS 81
5.1 Introduction 81
5.2 Non-linear Classifiers for Large-scale Data 82
5.2.1 Adaptive Multi-hyperplane Machines (AMM) 82
5.2.2 Low-rank Linearization SVM (LLSVM) 83
5.2.3 Budgeted Stochastic Gradient Descent (BSGD) 83
5.2.4 Time and space complexity 84
5.3 The Software Package 84
5.3.1 Performance comparison 86
6 DISTRIBUTED LEARNING OF CLASSIFICATION MODELS 87
6.1 Introduction 87
6.2 Confidence-Weighted Classification 90
6.3 MapReduce framework 92
6.3.1 AllReduce framework 95
6.4 Confidence-Weighted Classification using MapReduce Framework . . 96
6.4.1 Reducer optimization of AROW-MR 97
6.5 Experiments 99

6.5.1 Validation on synthetic data 99
6.5.2 Ad Latency problem description 101
6.5.3 Validation on Ad Latency data 103
7 CONCLUSION **108**
BIBLIOGRAPHY **110**

LIST OF TABLES

2.1 Complexities of the reordering algorithms 23

2.2 Evaluation of performance on *waveform* data of size 10,000; listed LK time is required for solving one TSP 25

2.3 FOM scores for benchmark data sets (EM-1 and EM-5 denote EM-ordering after 1 and 5 iterations of Algorithm 1, respectively, baseline result is FOM before reordering; also shown size n, dimensionality m, and number of classes c) 29

3.1 Average number of correctly aligned non-English documents to documents in English (numbers in the parentheses are the best obtained performances; baseline method matches the documents according to their lengths) 49

4.1 Summary of notation 53

4.2 Summary of large datasets 67

4.3 Error rate and training time comparison with large-scale algorithms (RBF SVM is solved by LibSVM unless specified otherwise. Poly2 and LibSVM results are from Chang et al. (2010)). 68

4.4 The number of weights in the classifiers 68

4.5 Error rate as a function of *checkerboard* pattern 80

5.1 Time and space complexities of the classification algorithms 84

5.2 Error rates (in %) and training times[1] on benchmark data sets 86

6.1 Features from the Ad Latency data set 104

6.2 Performance comparison of AROW and AROW-MapReduce on Ad Latency task in terms of the AUC 105

6.3 Performance of distributed logistic regression 106

LIST OF FIGURES

2.1 (a) Binary tree after recursive k-means, with $k = 2$; (b) Resulting 2^l-ary tree ($l = 2$), obtained after line 3 in Algorithm 2; (c) TSP defined on node 5 is solved on its children, together with left and right neighbor nodes 18 and 7, see Algorithm 3 19

2.2 Performance of ordering algorithms on *circles* data ($n = 500$, path length L is given in parentheses) 26

2.3 Execution times on 2-D and 3-D *uniform* data set 27

2.4 FOM and L measures on a 2-D toy data set (color encodes the order of an example in the ordered data set, ranging from white to black): (a) LK (0.038; 181.44); (b) TSP-means (0.033; 199.86) 28

2.5 Visualization of *waveform*, the last 3 columns are class assignments averaged over sliding window of length 20; FOM and L measures given in parentheses 30

2.6 Traffic data: (a) the original data set (brighter pixels denote higher volumes); (b) the ordered data set (white lines denote user-selected cluster boundaries); (c) color-coded sensor locations in Minneapolis road network (neighboring clusters were assigned similar colors) . . . 31

2.7 *stocks* data set (9 rightmost columns encode industries; dark pixels in the heatmap encode high negative returns, while bright pixels encode high positive returns): (a) the original data set; (b) the data set after reordering and clustering upon inspection of the heatmap 31

3.1 Alignment of 320 images to "AAAI 2012" letter grid 45

3.2 Number of correctly recovered images for different set sizes (error bars represent confidence intervals of one standard deviation, calculated after 10 runs) 46

3.3 Alignment results of CKS for matching of 320 left and right image halves (206 correct matches) 47

3.4 Analysis of CKS performance on the task of aligning 320 left and right images halves 48

4.1 Convergence of MM training 56

4.2 Detailed results on url data (upper two panels) and mnist8m_bin data (lower two panels) 71

4.3 (a) Visualizing proof of Theorem 4; (b) GAMM on 1×7 XOR-like data 73

4.4 Simple 1-dimensional XOR example where SGD training fails 77

4.5 (a) and (b) MM performance on 4×4 *checkerboard* data; (c) and (d) 2-D *weights* data set 78

4.6 AMM and GAMM performance on noisy 4×4 *checkerboard* data set 79

6.1 Results on the synthetic *waveform* data set (with 50,000 training examples) 100

6.2 ROC curve for AROW-MR and AROW 105

CHAPTER 1

INTRODUCTION

Recent advent of high-throughput applications which generate data sets of unprecedented scales, with billions of high-dimensional data examples stored on hundreds of terabytes of memory, such as remote sensing, crowd-sourcing, high-energy physics, social networks, or high-frequency trading, has brought forward a clear need for computational approaches that can efficiently learn from Big Data problems (Bizer et al., 2012; Labrinidis and Jagadish, 2012; Lohr, 2012). Emerging conferences that specifically address the Big Data issues, as well as the number of recent publications related to large-scale tasks, underline the significance of this emerging field. Moreover, recently introduced Big Data Research and Development Initiative by the United States government, aimed at providing support for these efforts, clearly indicates globally-recognized, strategic importance, as well as future potential and impact of research related to large-scale data (Mervis, 2012).

With the emergence of extremely large-scale data sets, researchers in machine learning and data mining communities are faced with numerous challenges related to the sheer size of the problems at hand, as many well-established supervised and unsupervised tools were not designed and are not suitable for such memory- and

time-intensive tasks. The inadequacy of standard machine learning tools in this new setting has led to investment of significant research efforts into the development of novel methods that can address such challenges.

These important problems are in the focus of this thesis, and we propose novel tools suitable for handling of large-scale data. First, we consider unsupervised approaches to data analysis, and explore and describe new visualization methods for fast knowledge extraction. We introduce data visualization problem using method based on efficient data reordering, and also describe method for smaller-scale data based on object matching. Next, we consider supervised learning setting and propose algorithms for fast training of accurate classification models on large data sets, capable of learning state-of-the-art classifiers on data sets with millions of examples and features within minutes.

1.1 Data Visualization

Data visualization has been used for hundreds of years in scientific research (Friendly, 2006) as it allows researchers to get a better insight into data they are studying. Visualization is used for exploratory analysis prior to application of statistical methods, it can also be used as a confirmatory tool to disprove or confirm hypotheses, while sometimes visual presentation is an ultimate goal (Keim et al., 2006). However, despite its long history and significant advantages of visual analysis of data (Friendly and Kwan, 2003), there still remains a need for further development of visualization methods. This is particularly evident when working with large-scale, high-dimensional data, where commonly used visualization tools are either too simplistic to gain a deeper insight into the data properties (e.g., histograms, scatter plots, pie and bar charts), or are too cumbersome and computationally costly in large-scale setting, such as parallel coordinates (Inselberg, 1985; Inselberg and Dimsdale, 1991), correlation matrix plots (Friendly, 2002), and biplots and star plots (Friendly and Kwan, 2003).

Inadequacy of standard tools has been recognized in a number of recent papers, as summarized in the statement from Vempala (2012): *Data in high dimension are difficult to visualize and understand. This has always been the case and is even more apparent now with the availability of large high-dimensional datasets and the need to make sense of them.*

In this thesis we present novel methods for data ordering and data visualization. By combining efficient clustering of data using k-means algorithm, and near-optimal ordering of found clusters using state-of-the-art TSP-solver, we obtain efficient algorithm (Djuric and Vucetic, 2013) that achieves performance better than existing ones, computationally more intensive methods. We complement this large-scale method by describing state-of-the-art object matching-based visualization based on Kernelized Sorting algorithm (Djuric et al., 2012), suitable for smaller-scale problems, which also helps to further exemplify and highlight the problems encountered when working with big data. Experimental evaluation confirms that the proposed methods allow for fast detection of interesting and useful patterns inherent to the data, even for users without expertise in data mining and machine learning.

1.2 Learning classification models on large-scale data

Supervised learning is another important task, often intractable in many modern applications due to time and memory constraints, considering prohibitively large scales of the data sets. Classification tasks are of particular interest, as the problem of classifying input data examples into one of finite number of classes can be found in many areas of machine learning. However, state-of-the-art non-linear classification methods, such as Support Vector Machines (SVMs) (Cortes and Vapnik, 1995), are not applicable to truly big data due to very high time and memory overhead, which are in general super-linear and linear in the data size N, respectively, significantly limiting their use when solving large-scale problems. Several methods have been

proposed to make SVMs more scalable, ranging from algorithmic speed-ups (Platt, 1998; Kivinen et al., 2002; Vishwanathan et al., 2003; Tsang et al., 2005; Rai et al., 2009), to parallelization approaches (Graf et al., 2004; Chang et al., 2007; Zhu et al., 2009). However, scalability of SVM training is inherently limited as non-linear SVMs are characterized by linear growth of model size with training data size N (Steinwart, 2003a). This led to an increased interest in linear SVM models (Gentile, 2002; Li et al., 2002; Shalev-Shwartz et al., 2007a; Fan et al., 2008), which have constant memory and $\mathcal{O}(N)$ training time and provide a scalable alternative to non-linear SVMs, albeit with a certain drop in prediction accuracy. Unfortunately, even linear time complexity may not be sufficiently efficient for modern data sets having petabytes of memory space, requiring researchers to develop and adopt new machine learning approaches in order to address extremely large-scale classification tasks.

To address the deficiencies of standard classification tools designed for small-scale problems, in this thesis we propose novel linear and non-linear classifiers suitable for large-scale setting. We first review Multi-hyperplane Machine (MM) classification model, and propose online Adaptive MM (AMM) algorithm Wang et al. (2011) which represents a trade-off between linear and kernel Support Vector Machines (SVMs), as it trains MMs in linear time on limited memory while achieving competitive accuracies on large-scale non-linear problems. Moreover, we present a C++ toolbox for developing scalable classification models (Djuric et al., 2013a), which provides an Application Programming Interface (API) for training of large-scale classifiers. In addition to the API, the toolbox provides easy-to-use command prompt and Matlab interfaces to highly-optimized implementations of several recently proposed SVM approximators, allowing the users easy access to efficient and accurate state-of-the-art large-scale classifiers. Lastly, we consider parallelization and distributed approaches to large-scale supervised learning using Hadoop, and propose AROW-MapReduce (Djuric et al., 2013b), a distributed learning algorithm for training of confidence-

weighted models using MapReduce framework. Experimental evaluation of the proposed methods shows state-of-the-art performance on a number of synthetic and real-world data sets, further paving a way for efficient and effective knowledge extraction from big data problems.

CHAPTER 2

VISUALIZATION THROUGH MATRIX REORDERING

2.1 Introduction

Data visualization has a long history in scientific research (Friendly, 2006) as it allows researchers to get a better insight into data they are studying. Visualization is used for exploratory analysis prior to application of statistical methods, it can also be used as a confirmatory tool to disprove or confirm hypotheses, while sometimes visual presentation is an ultimate goal (Keim et al., 2006). However, despite its long history and significant advantages of visual analysis of data (Friendly and Kwan, 2003), there still remains a need for further development of visualization methods. This is particularly evident when working with large-scale, high-dimensional data, where commonly used visualization tools are either too simplistic to gain a deeper insight into the data properties (e.g., histograms, scatter plots, pie and bar charts), or are too cumbersome and computationally costly in large-scale setting, such as parallel coordinates (Inselberg, 1985; Inselberg and Dimsdale, 1991), correlation matrix plots (Friendly, 2002), and biplots and star plots (Friendly and Kwan, 2003). Inadequacy

of standard tools has been recognized in a number of recent papers, as summarized in the statement from Vempala (2012): *Data in high dimension are difficult to visualize and understand. This has always been the case and is even more apparent now with the availability of large high-dimensional datasets and the need to make sense of them.*

In this paper, we focus on visualizing data sets that can be represented as an $n \times m$ data table, where rows represent n examples and columns represent m features. Standard data exploration tools such as histograms and scatter plots provide only a basic understanding of the data; histogram is a tool for understanding distributions of each feature, while scatter plot is a tool for understanding correlations between pairs of features. A more advanced visualization approach is low-dimensional data projection, where examples are projected into a two-dimensional subspace and visualized using a scatter plot, such as Principal Component Analysis (PCA), Locally Linear Embedding (LLE) (Roweis and Saul, 2000), or Stochastic Neighborhood Embedding (SNE) (Hinton and Roweis, 2002). However, projecting examples into a 2-D subspace and visualizing them using a scatter plot often implies a significant loss of information. Moreover, while the resulting 2-D or 3-D scatter plots can provide insight into an underlying manifold structure, they may be difficult to interpret and provide actionable knowledge. This is evident when examining related publications that typically use non-linear projection methods to project a set of images (e.g., faces, digits) to a 2-D scatter plot, and then plot the original image next to the corresponding projected point to illustrate the quality of visualization. However, practical problem is that there are only several types of data sets where the projected examples can be conveniently annotated in a lower-dimensional plot.

An alternative to showing two- and three-dimensional scatter plots of the projected data is to plot the original data. By observing that a data set can be represented as a two-dimensional matrix, it becomes evident that is could be plotted as a heatmap (e.g., using `imagesc` command in Matlab and Octave). Since examples

and features in a typical data set are sorted in an arbitrary order (e.g., randomly, or by example or feature ID), heatmap of the original data might not be informative. There are two possible alternatives for improving the heatmap. One is to perform clustering (e.g., k-means clustering) and sort all examples based on which cluster they are assigned to. However, the outcome greatly depends on the chosen number of clusters, and could result in artifacts where it might appear that there are clear clusters even when this is not the case. More appropriate strategy for plotting data heatmaps is to first reorder its rows (columns), such that similar rows (columns) are placed next to each other (Bertin and Barbut, 1967; Mäkinen and Siirtola, 2000; Hahsler et al., 2007).

There are many possible approaches for ordering of data tables. One is to project examples onto the largest principal component obtained by PCA or to a principal curve obtained by LLE, and to order the examples by traversing the line or the curve. However, ordering is not an explicit objective of either PCA or LLE but only a byproduct of a manifold search, and may result in lower-quality visualization. An alternative, very popular in gene expression analysis, is to perform hierarchical clustering and to order examples by traversing leaves of the binary tree (Eisen et al., 1998). However, the resulting algorithm is computationally expensive and can be applied only to data sets with several thousand examples. Moreover, there are 2^{n-1} ways to order the resulting hierarchical tree, which may open a costly optimization problem (Bar-Joseph et al., 2002). Another interesting approach presented in Ding and He (2004) is ordering based on spectral clustering. Similarly to hierarchical clustering approaches and unlike the method proposed in this paper, in large-scale setting spectral analysis becomes time- and memory-intensive, and the algorithm may also give suboptimal results when the data consists of several clusters that are not well separated.

Data table reordering can also be seen as the Traveling Salesman Problem (TSP)

(Climer and Zhang, 2004), where examples represent cities, and the task is to find a path through all the cities such that the traversal cost is minimized. While TSP is an NP-complete problem that requires exponential computation time, there are efficient heuristics that in practice give high-quality tours. The Lin-Kernighan (LK) method (Lin and Kernighan, 1973) has been widely accepted as the method providing the best trade-off between tour quality and computational speed, scaling as $\mathcal{O}(n^{2.2})$; as such, it is applicable only to moderately-sized data sets. Moreover, treating reordering directly as TSP carries a strong assumption that the features were properly scaled from the perspective of visual quality of heatmaps.

We propose a novel ordering method that addresses shortcomings of the existing methods. Our main contributions are:

- Ordering is formalized as finding a permutation of rows (or columns) that results in a maximally compressible data set, as defined by the entropy of the residuals of predictive coding.
- The problem is solved by an Expectation-Maximization (EM)-like algorithm, which alternatively solves a TSP and reweights features based on the quality of the resulting tour.
- A fast $\mathcal{O}\big(n\log(n)\big)$ TSP solver is proposed, called the TSP-means, that finds tours with lengths comparable to those found by the LK algorithm. It is based on a construction of a binary tree by recursive use of k-means (with $k = 2$) and subsequent reordering of the tree nodes by the LK algorithm.

2.2 Background

In this section we describe the works and ideas that led to the proposed visualization method. We first introduce the existing approaches for visualization of high-dimensional data, and then present matrix reordering and data seriation techniques. Lastly, we give an overview of the TSP and the existing methods for solving this

classical combinatorial problem.

2.2.1 Data visualization

Visualization of data has been an integral part of scientific research from the earliest times, with visual representations of data appearing in scientific literature from as early as the 10^{th} century (Friendly, 2006). A great number of approaches for data visualization has been introduced since (see Keim, 2001, 2002), with visualization methods most commonly used in our everyday lives, such as histograms and pie charts often encountered in newspaper and weather reports, being in use for more than a century in a nearly unchanged form (Friendly, 2006; Wilkinson and Friendly, 2009; Loua, 1873). However, recent technological advances and emergence of large-scale data sets have clearly indicated limitation of the existing methods in this new setting (Keim et al., 2006; Vempala, 2012), and there remains a clear need for the development of novel visualization approaches.

Visualization of high-dimensional data is of particular interest (Vempala, 2012), and this problem has received significant attention in the visualization community. Often explored direction is finding lower-dimensional representation of the data, which could then be more easily visualized using the standard visualization tools. In Vadapalli and Karlapalem (2009) and Tatu et al. (2012), the authors propose methods that explore interactions between examples in subspaces of the original high-dimensional space, and plot these lower-dimensional representations in a form of similarity matrices or scatter plots in order to gain better understanding of the data. However, the methods become intractable as number of examples and dimensions grows, and may not be suitable for large-scale visualization tasks. Instead of using subspace search, another idea is to compute more involved projections of the data into 2- or 3-D spaces. This approach includes PCA, where examples are projected along the directions describing most of the variance, and non-linear projections such

as LLE (Roweis and Saul, 2000), SNE and its extension t-SNE (Hinton and Roweis, 2002; Van der Maaten and Hinton, 2008), Self-Organizing Maps (SOM) (Williams et al., 2008), Isomap (Tenenbaum et al., 2000) or Laplacian eigenmaps (Belkin and Niyogi, 2003), which attempt to project examples to a lower-dimensional, non-linear manifold. However, lower-dimensional projection methods in most cases imply a significant loss of information, and the resulting plots may also be difficult to interpret by non-experts for whom the visualization results are often intended.

To address this issue, an interesting approach is to represent examples in their original, high-dimensional space. A very popular technique implementing this idea are parallel coordinates (Inselberg, 1985; Inselberg and Dimsdale, 1991), which have been used for data visualization for more than a century (Friendly, 2006). However, although parallel coordinates can be used to quickly discover trends in moderate-sized data sets, they become cluttered when number of examples or dimensions becomes large (Fua et al., 1999; Walter et al., 2003; Artero et al., 2004), thus significantly limiting the quality of visualization. On the other hand, an alternative visualization tool are heatmaps, with very long and rich history (Wilkinson and Friendly, 2009; Loua, 1873). In contrast to parallel coordinates, heatmaps do not suffer from the deficiencies related to extreme data sizes, and can be used in large-scale setting to provide a holistic view of the data. We use this insight and propose a scalable algorithm for generating high-quality, large-scale heatmaps, obtained by data preprocessing through reordering.

2.2.2 Data reordering

Data reordering or seriation is an important step in exploratory data analysis. This family of unsupervised methods is based on the following observation: assuming that a data set is represented in a form of a two-dimensional reorderable matrix, any permutation of its columns or rows does not lead to loss of information (Mäkinen

and Siirtola, 2000). Therefore, by permuting rows (columns) so that similar rows (columns) are close, followed by visualization of the modified data matrix, we can reveal unknown regularities and patterns in the data without modifying the data. Data ordering has deep roots in a number of disciplines in social studies (e.g., archeology Petrie, 1899, anthropology Czekanowski, 1909; for an excellent overview see Liiv, 2010 and references therein). Beyond social sciences, data ordering has been popular in gene expression data analysis in bioinformatics (Eisen et al., 1998; Bar-Joseph et al., 2002) and analysis of geographical data (Guo, 2007). It is also important in bandwidth minimization (Del Corso and Manzini, 1999) and data compression (Blandford and Blelloch, 2002; Pinar et al., 2005).

We describe in more detail several popular methods for data ordering. As baseline methods we can consider LLE and PCA, two popular low-dimensional projection algorithms. One-dimensional projection by either PCA or LLE effectively induces a linear ordering in the new 1-D space, and can be used to reorder rows and columns of the data matrix. LLE method first finds k nearest neighbors in the original high-dimensional space for each example, then attempts to project the data to lower-dimensional space while keeping the relationships between neighbors the same. Due to the fact that all distances between examples need to be computed, the time complexity of the algorithm amounts to $\mathcal{O}(n^2)$. Regarding PCA, we can use the first principal component and project the data onto it. As we only need to compute the first principal component (found in $\mathcal{O}(n)$ (Roweis, 1998)), this algorithm is very fast, and projection of the data to the first principal component and subsequent sorting of the projected values result in modest $\mathcal{O}\big(n \log(n)\big)$ time complexity. We can also consider ordering based on spectral clustering (SC) (Ding and He, 2004), which can be seen as a low-dimensional, non-linear projection method. In their work, the authors show that linear ordering of examples can be found by computing the second largest eigenvector of the normalized similarity matrix. However, similarly to

LLE, time and space complexity of $\mathcal{O}(n^2)$ render the method infeasible in large-scale setting.

Two approaches popularized in bioinformatics are hierarchical clustering (HC) (Eisen et al., 1998) and hierarchical clustering with optimal leaf ordering (HC-olo) (Bar-Joseph et al., 2002). Hierarchical clustering is a bottom-up method, which starts by clustering two most similar examples and represents this new cluster with its centroid. Examples and centroids are repeatedly grouped together until all examples belong to a single, root cluster. The method finds binary tree representation of the data, resulting in $\mathcal{O}(n^2)$ time complexity as distances between all examples need to be calculated. In order to find ordering of examples, we simply read leaves of the tree from left to right. However, there are 2^{n-1} linear orderings of tree nodes that obey the obtained tree structure (i.e., we can flip each of $n-1$ internal nodes and still have the same tree structure). To solve this issue, in Bar-Joseph et al. (2002) the authors present a dynamic programming approach to find the optimal leaf ordering for a given tree structure in $\mathcal{O}(n^3)$ time, which makes the algorithm intractable for larger data sets.

2.2.3 Traveling salesman problem (TSP)

Traveling Salesman Problem is a classical problem in computer science. The problem can be described as follows: given n cities, along with non-negative costs of traveling from the i^{th} city to the j^{th} city $d(i,j), i,j \in \{1,2,...,n\}$, find a shortest path such that each city is visited exactly once and the path completes in the starting city. More formally, letting π be permutation (or ordering) of n cities, the task is to find optimal permutation π^* so that the total tour length is minimized,

$$\pi^* = \underset{\pi \in \Pi_n}{\arg\min} \left(d\big(\pi(n), \pi(1)\big) + \sum_{i=2}^{n} d\big(\pi(i-1), \pi(i)\big) \right), \tag{2.1}$$

where Π_n is the set of all permutations of the first n integers, and $\pi(i)$ denotes the i^{th} city to be visited. The TSP is NP-complete (Karp, 1972), thus very difficult to solve optimally.

Due to the NP-completeness of the problem, TSPs were historically very hard to solve with limited computational resources. One of the first large TSP problems solved to optimality involved only 49 cities, and the solution was described in 1954 in Dantzig et al. (1954). Interestingly, the authors solved the problem by manually applying ideas that later led to the cutting-plane algorithm (Gomory, 1958). Several polynomial-time heuristic methods with upper bounds on performance have been proposed since to approximately solve the TSP, including nearest neighbor, nearest insertion, furthest insertion (Rosenkrantz et al., 1977), Christofides heuristic (Christofides, 1976), and LK heuristic (see Gutin and Punnen, 2002 for a comprehensive overview of methods). Currently, the largest TSP instance solved to optimality comprises nearly 86,000 cities (Applegate et al., 2009).

One of the most powerful heuristics for finding optimal or near-optimal solutions to TSP is the LK method, having $\mathcal{O}(n^{2.2})$ time complexity. The method introduces a variable λ-opt move (where $\lambda \geqslant 2$) to reach a better tour, meaning that at each iteration we search for increasing λ number of links on the current tour that could be broken and replaced by the same number of links currently not on the tour. The λ-opt move is performed if the resulting, modified tour has lower cost than the current solution. The method starts with a random tour, and then iteratively applies λ-opt moves, until no such move leads to a better solution (it is then said that the current tour is λ-optimal). An efficient implementation of LK heuristic is presented in Helsgaun (2000), which achieved the best results on all known large-scale TSP problems.

An interesting application of TSP solvers is in matrix reordering and clustering, where each data example is considered a city. Interestingly, one of the first matrix

reordering techniques, the Bond Energy Algorithm (BEA), is in fact a simple nearest insertion TSP heuristic (McCormick et al., 1972). In Climer and Zhang (2004), the authors propose adding several "dummy" cities which have distances equal to 0 to all other cities. In this way, after computing the shortest tour through all cities, the "dummy" cities act as boundaries between different clusters. However, as discussed in Biedl et al. (2001) and as shown in the experimental section of this paper, directly applying TSP to the whole data set can lead to ordering that is very sensitive to noise inherent in the data set, and consequently to poor data visualization. In DiMaggio et al. (2008) the authors propose a TSP-based biclustering method, which first finds clusters across rows, followed by clustering across columns of a data matrix to obtain meaningful biclusters. As the proposed methods apply TSP solvers directly on the whole data set, they are not scalable to large data sets due to super-quadratic time complexity of the best available TSP solvers. In contrast to the existing methods, we present an effective algorithm with time requirement of only $\mathcal{O}(n \log(n))$, allowing for high-quality, scalable reordering and visualization of large data matrices.

2.3 Methodology

An intuitive goal of ordering is to find permutation of rows so that similar examples are grouped together. We propose a principled approach for ordering that finds permutation of rows producing a maximally compressible data set. We will explain how to order rows, while noting that columns can be ordered using the same procedure on the transposed data table.

2.3.1 Differential Predictive Coding (DPC)

Let us assume a data set D is given in a form of an $n \times m$ data table, $D = [x_{ij}]_{i=1,\ldots,n,j=1,\ldots,m}$, where the i^{th} row vector $\mathbf{x}_i = [x_{i1}, \ldots, x_{im}]$ is the i^{th} example having m numeric features. DPC replaces the i^{th} example $\mathbf{x}_i$ with its difference from

the previous example, $\boldsymbol{\varepsilon}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$, where $\boldsymbol{\varepsilon}_i$ is called the DPC residual. Therefore, DPC transforms the data table $D = [\mathbf{x}_1^{\mathrm{T}}, \mathbf{x}_2^{\mathrm{T}}, \ldots, \mathbf{x}_n^{\mathrm{T}}]^{\mathrm{T}}$ into $D_{DPC} = [\mathbf{x}_1^{\mathrm{T}}, \boldsymbol{\varepsilon}_2^{\mathrm{T}}, \ldots, \boldsymbol{\varepsilon}_n^{\mathrm{T}}]^{\mathrm{T}}$ without the loss of information, as the original data set D can be reconstructed from D_{DPC}.

If the data table D is ordered such that similar rows are placed next to each other, DPC residuals $\boldsymbol{\varepsilon}$ would be smaller than the original examples $\mathbf{x}_i$. As a result, the entropy of rows in D_{DPC} would be smaller than the entropy of rows in D, indicating that D_{DPC} is more compressible than D. The entropy of the original examples is defined as $H_D(\mathbf{x}) = \mathbb{E}[-\log(\mathbb{P}_{\mathcal{X}}(\mathbf{x}))]$, where $\mathbb{P}_{\mathcal{X}}(\mathbf{x})$ is the probability density of vectors $\mathbf{x}$, and entropy of DPC residuals is defined as $H_{DPC}(\boldsymbol{\varepsilon}) = \mathbb{E}[-\log(\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon}))]$, where $\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon})$ is a probability density of vectors $\boldsymbol{\varepsilon}$. Thus, small entropy $H_{DPC}(\boldsymbol{\varepsilon})$ implies that DPC residuals are small, which in turn implies that D is a well-ordered data set. As a result, entropy of the DPC residuals is a good measure of ordering quality. We note that $H_{DPC}(\boldsymbol{\varepsilon})$ can be estimated as

$$H_{DPC}(\boldsymbol{\varepsilon}) = -\frac{1}{n-1}\sum_{i=2}^{n} \log \mathbb{P}_{\mathcal{E}}(\mathbf{x}_i - \mathbf{x}_{i-1}). \tag{2.2}$$

2.3.2 Relationship between entropy minimization and ordering

In data mining, data sets can often be considered as arbitrarily ordered collections of examples and features. As a consequence, any permutation π of rows from $D = [\mathbf{x}_i]_{i=1,\ldots,n}$, resulting in $D_\pi = [\mathbf{x}_{\pi(i)}]_{i=1,\ldots,n}$, does not lead to loss of information. We propose that the optimal permutation π^* is the one that results in minimization of entropy of DPC residuals,

$$\pi^* = \arg\min_{\pi \in \Pi_n} H_{DPC}^{\pi}, \tag{2.3}$$

where we used superscript π to denote the specific permutation of rows of the data table D.

We observed that when applying DPC on a number of well-ordered data sets with numerical features $\mathbb{P}_{\mathcal{E}}$ often resembles multivariate Gaussian or Laplacian distribution with diagonal covariance matrix. Let us first consider the case when $\mathbb{P}_{\mathcal{E}}$ is a multivariate Gaussian distribution,

$$\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon}) = (2 \cdot \pi)^{-m/2} |\Sigma|^{-1/2} \cdot \exp(-0.5 \cdot \boldsymbol{\varepsilon}^{\mathrm{T}} \cdot \Sigma^{-1} \cdot \boldsymbol{\varepsilon}), \tag{2.4}$$

where Σ is a diagonal covariance matrix with the j^{th} element on a diagonal equal to the variance σ_j^2 of the DPC residuals of the j^{th} feature. $H^{\pi}_{DPC}(\boldsymbol{\varepsilon})$ could then be expressed as

$$\begin{aligned} H^{\pi}_{DPC}(\boldsymbol{\varepsilon}) = \frac{n}{2(n-1)} \left(m \cdot \log(2\pi) + \sum_{j=1}^{m} \log \sigma_j \right) + \\ \frac{1}{2(n-1)} \sum_{i=2}^{n} \sum_{j=1}^{m} \frac{(x_{\pi(i),j} - x_{\pi(i-1),j})^2}{\sigma_j^2}. \end{aligned} \tag{2.5}$$

When $\mathbb{P}_{\mathcal{E}}$ is modeled as a Laplacian distribution, and assuming independence of elements of $\boldsymbol{\varepsilon}$, $\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon})$ is equal to

$$\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon}) = \prod_{j=1}^{m} \frac{1}{2b_j} \exp(-\frac{|\boldsymbol{\varepsilon}_j|}{b_j}), \tag{2.6}$$

where $b_j^2 = \sigma_j^2/2$. The corresponding $H^{\pi}_{DPC}(\boldsymbol{\varepsilon})$ is similar to (2.5), with the main difference being that $|\sigma_j|$ is used instead of σ_j^2 in the third term of (2.5).

Upon modeling $\mathbb{P}_{\mathcal{E}}$ as Gaussian or Laplacian distribution, and observing that this results in introduction of m new parameters $\{\sigma_j\}_{j=1,\ldots,m}$, we can restate (2.3) as

$$(\pi^*, \{\sigma_j^*\}_{j=1,\ldots,m}) = \underset{\pi, \{\sigma_j^*\}_{j=1,\ldots,m}}{\arg\min} \; H^{\pi}_{DPC}(\boldsymbol{\varepsilon}). \tag{2.7}$$

Solving (2.7) requires finding the best ordering and the best estimate of variance of DPC residuals for each feature.

2.3.3 Reordering for entropy minimization

We propose to solve (2.7) in an iterative manner similar to the EM algorithm. The method is given as Algorithm 1. In the M-step (line 2), by assuming that values of σ_j are known, the problem reduces to finding ordering π that minimizes the last term from (2.5), which is equivalent to solving the TSP on examples whose features are downscaled using $\{\sigma_j\}_{j=1,\dots,m}$. Given the current ordering π, the goal of the E-step (line 3) is to find $\{\sigma_j\}_{j=1,\dots,m}$ that minimizes $H^{\pi}_{DPC}(\boldsymbol{\varepsilon})$. It is evident that the E-step is equivalent to finding σ_j using the maximum likelihood approach, where σ_j is found as

$$\sigma_j^2 = \frac{1}{n-1}\sum_{i=2}^{n}(x_{\pi(i),j} - x_{\pi(i-1),j})^2. \tag{2.8}$$

Algorithm 1 converges to a local minimum, since both E- and M-steps lead to decrease in $H^{\pi}_{DPC}(\boldsymbol{\varepsilon})$. As the algorithm resembles the EM and is based on an information-theoretic principle of Entropy Minimization, we call it the EM-Ordering.

Note that successful ordering will result in small σ_j for features that are correlated to others, and large σ_j for noisy or uncorrelated features. Intuitively, if the j^{th} feature cannot be ordered well during the procedure, its DPC entropy, and thus σ_j will be increased. As a result, its importance will be reduced due to larger downscaling in (2.5).

2.3.4 Feature scaling

We observe that the proposed algorithm allows for cases when for some features and for some orderings it holds that $H_D(\mathbf{x}_j) < H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$, where $H_D(\mathbf{x}_j)$ and $H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$ are entropies of the j^{th} feature and of its DPC residuals, respectively. If this is the case, it might be preferable to ignore the j^{th} feature during ordering. Considering this observation, we propose two strategies:

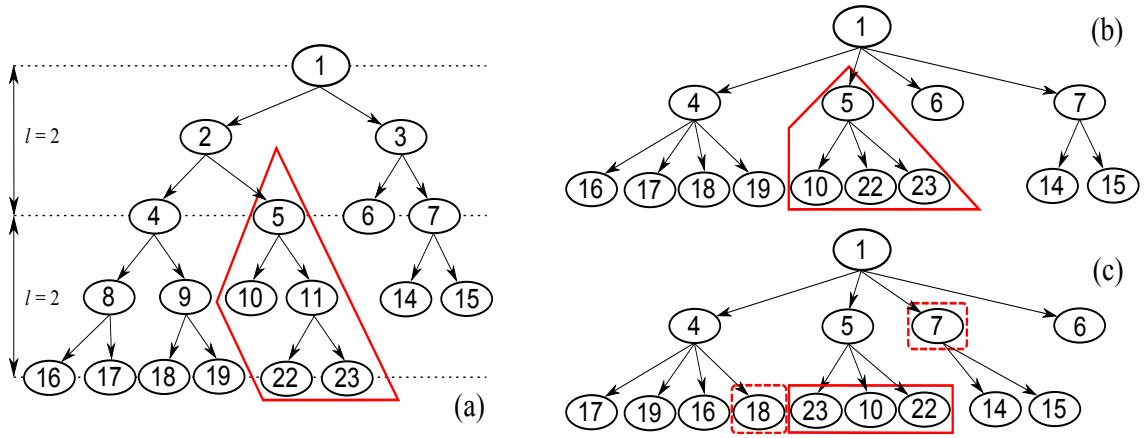


FIGURE 2.1: (a) Binary tree after recursive k-means, with $k = 2$; (b) Resulting 2^l-ary tree ($l = 2$), obtained after line 3 in Algorithm 2; (c) TSP defined on node 5 is solved on its children, together with left and right neighbor nodes 18 and 7, see Algorithm 3

1. Hard feature scaling. In this case, features for which holds that $H_D(\mathbf{x}_j) < H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$ are scaled down to zero by introducing $\sigma_j \to \infty$, in order to prevent their adverse influence on the overall entropy minimization.

2. Soft feature scaling. In this case, no action is being taken for features where $H_D(\mathbf{x}_j) < H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$.

Hard scaling results in a removal of the features that cannot be successfully ordered and, as such, can be considered a feature selection algorithm. However, to prevent removing features that at a given iteration just barely satisfy the condition $H_D(\mathbf{x}_j) < H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$, but could be successfully ordered in future iterations of Alg. 1, we can use the following criterion:

3. Hard feature scaling with tolerance. The j^{th} feature is removed if, for some $\alpha > 1$, $H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j) > \alpha H_D(\mathbf{x}_j)$.

2.3.5 TSP-means algorithm

In this section, we propose a TSP solver used in M-step of Algorithm 1, called TSP-means. By combining data clustering and efficiently solving a number of small-scale TSPs defined on the cluster centroids, we obtain a highly scalable algorithm. In

Algorithm 1 EM-ordering

Inputs: data set D; initial guess for $\{\sigma_j\}_{j=1,\ldots,m}$
Output: ordered set D; learned $\{\sigma_j\}_{j=1,\ldots,m}$

1. **repeat** until convergence
2. **run** TSP solver for current σ_j to find π
3. **calculate** σ_j for current ordering of D

Algorithm 2 Generation of 2^l-ary tree T^l

Inputs: binary tree T; subtree depth parameter l
Output: 2^l-ary tree T^l

1. **extend** leafs of T at levels $\big((i-1)\cdot l+1\big)$ through $(i\cdot l-1)$ to level $i\cdot l, i>0$
2. **select** nodes of T at levels $i\cdot l, i \geqslant 0$ for inclusion in tree T^l
3. **connect** nodes in T^l originating from level $i\cdot l$ in T, $i>0$, to their predecessor at level $(i-1)\cdot l$ from T
4. **add** children to every leaf of T^l, where the children are individual examples in that leaf's cluster

addition, as will be discussed later, by its design TSP-means often results in a more informative visualization than when LK is directly used on all examples.

The solver, summarized in Algorithm 3, begins by recursively applying k-means clustering with $k = 2$ (line 1), to create a binary-tree T representation of the data set D, as shown in Figure 2.1(a). The root of the tree corresponds to the whole data set and is represented by its centroid. Internal nodes correspond to clusters found by running k-means on their parents, and are represented by the cluster centroids. The k-means on a node is not performed if a node contains less than or exactly 2^l examples, where l is a user-defined parameter.

In the next step, as formalized in Algorithm 2 and illustrated in Figure 2.1(b), we transform binary tree T into 2^l-ary tree T^l by keeping only nodes at every l^{th} tree level, starting with a root node. Each node at level $(i\cdot l)$, $i>0$, becomes a child of its predecessor at $((i-1)\cdot l)^{\text{th}}$ level (e.g., nodes 22 and 23 become children of node 5). In addition, leafs at any level in the tree T also become leafs in the tree T^l. For

Algorithm 3 TSP-means

Inputs: data set D; subtree depth parameter l
Output: ordered list $\mathcal{L}$

1. **create** binary tree T by recursively splitting D
2. **run** Algorithm 2 on T to generate T^l
3. **set** $\mathcal{L} \leftarrow$ root r of T^l
4. **while** (not all leaf nodes of T^l in $\mathcal{L}$)
5. **for each** z in L in left-to-right order
6. **if** (z has children)
7. **solve** local TSP defined on children of z and immediate neighbors of z in the list $\mathcal{L}$
8. **replace** z in $\mathcal{L}$ by its children, in order given by the TSP solution

example, node 10 becomes a child of node 5, as shown in Figure 2.1(b). Note that leafs of the 2^l-ary tree T^l represent clusters with no more than 2^l examples, and are the same as leafs of T. Finally, we add another tree level to T^l by making its current leaf nodes parents of examples in their cluster. This results in the final 2^l-ary tree whose leaf nodes are the individual examples.

After the creation of 2^l-ary tree, we perform a breadth-first traversal of the tree from left to right (Alg. 3, lines 3 - 8). The main idea is to reorder the internal and leaf nodes so that similar clusters and examples are closer, resulting in a good ordering of the data set. For this purpose we create a list $\mathcal{L}$, which initially holds only the root of the tree (Algorithm 3, line 3). Then, we visit nodes in the list $\mathcal{L}$ sequentially from left to right and solve a local TSP defined on centroids of children of the current node in $\mathcal{L}$, giving us an ordering where similar children are close (Algorithm 3, lines 4 - 7). Before moving on to the next node of the list $\mathcal{L}$, we replace the current node in the list $\mathcal{L}$ with its children reordered according to the TSP solution (Algorithm 3, line 8). Once we reach the end of the list $\mathcal{L}$, we start again from the beginning. For example, for the tree in Figure 2.1(b), we initialize $\mathcal{L} = [1]$. We traverse the list from left to right, and, after solving TSP defined on children of the current node of the list (there is only node 1 at the moment), we replace node 1 in the list $\mathcal{L}$ with

its children in the order given by the TSP solution, resulting in $\mathcal{L} = [4, 5, 7, 6]$, see Figure 2.1(c). As we reached the end of the list, we start from the beginning and the current node of $\mathcal{L}$ becomes node 4, whose children define the next TSP to be solved. The algorithm completes when $\mathcal{L}$ contains only individual examples, and ordered data set is returned in a form of the resulting list $\mathcal{L}$.

TSP tour computed only on children of the current node of $\mathcal{L}$ would result in discontinuity between tours of the neighboring nodes in $\mathcal{L}$, as each local tour would be computed irrespectively of the neighboring nodes. Therefore, to ensure that the ordering is smooth, when solving a TSP defined on children of the current node in $\mathcal{L}$ we also include its left and right neighbors from $\mathcal{L}$ if they exist. We set the distance between left and right neighbor nodes to 0, so that they are guaranteed to become neighbors on the found tour. After solving thus defined TSP, the found tour is cut at these neighbors to obtain an ordered list, and the children of the current node are reordered according to the TSP solution. For example, given $\mathcal{L} = [17, 19, 16, 18, 5, 7, 6]$, obtained after solving TSP defined by the previous current node 4, we need to solve the TSP defined by the new current node 5. The TSP being solved includes nodes $\{18, 10, 22, 23, 7\}$, and the resulting tour is $[18, 23, 10, 22, 7]$. Before moving on to node 7, we replace node 5 with its ordered children to obtain the updated list $\mathcal{L} = [17, 19, 16, 18, 23, 10, 22, 7, 6]$, see Figure 2.1(c).

2.3.6 Further details

There are several important advantages that TSP-means offers over the existing algorithms. First, it has very favorable time complexity. As k-means clustering has time requirement linear in number of examples, it takes $\mathcal{O}\big(n \log(n)\big)$ time to build the binary tree T, assuming k-means results in nearly balanced clusters. After creating T^l in $\mathcal{O}(n)$ time, there are $\mathcal{O}(\frac{n}{2^l})$ nodes in T^l, assuming a nearly balanced tree T^l of expected depth close to $\log(n)$. Each non-leaf node in T^l requires solving a single

Table 2.1: Complexities of the reordering algorithms

Algorithm	**Time**	**Space**
PCA	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}(n)$
LLE	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
SC	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
HC	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
HC-olo	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
LK	$\mathcal{O}(n^{2.2})$	$\mathcal{O}(n)$
TSP-means	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}(n)$

TSP whose size is bounded by $(2^l + 2)$. Therefore, if LK algorithm is used for TSP, time complexity of solving each TSP is $\mathcal{O}(2^{2.2l})$. It follows that the overall time requirement of TSP-means is only $\mathcal{O}\big(2^{1.2l}n\log(n)\big)$. Summary of time and space complexities of TSP-means and the competing methods is given in Table 2.1.

TSP-means is also amenable to parallelization. In particular, clustering of non-root nodes can be distributed over multiple processors, while solving TSPs defined on higher-level nodes can be performed concurrently with clustering of lower-level nodes. In addition, the algorithm allows for interactive visualization of large data sets. Unlike the competing algorithms, where a user can plot the results only once the algorithm completes, TSP-means provides meaningful output during the execution at each depth of the tree. It can visualize the data during runtime, by first showing coarse data ordering, and then transitioning (or "zooming in") towards finer resolutions as the algorithm descends further down the tree. In particular, as the list $\mathcal{L}$ is expanded we can show centroids currently in the list, weighted by the number of examples in the cluster that they represent. In this way, useful local and global patterns can be discovered even before completely traversing the tree T^l. Note that TSP-means can be further sped up by running k-means on a sub-sampled data for each node in the tree T instead on the whole data set, resulting in constant-time calls to k-means.

2.4 Experiments

In all experiments, initial values of variances $\{\sigma_j\}_{j=1,\ldots,m}$ in EM-ordering were initialized to standard deviation of features. We used hard feature scaling with tolerance of $\alpha = 1.1$, and run EM-ordering for 5 iterations. We assumed a Gaussian distribution from (2.4) for DPC residuals. To build a tree T, at each step of recursion we run k-means ($k = 2$) on 100 randomly sampled examples. For HC and HC-olo we used the Bioinformatics toolbox in Matlab (with average linkage). SC was implemented in Matlab (similarity matrix computed using Gaussian kernel), while codes for LLE[1] (number of neighbors was set to 15), and Lin-Kernighan[2] were found online.

It is not obvious how to measure quality of visualization. As a proxy measure we used Figure of Merit (FOM) (Yeung et al., 2001) when labeled examples are available. Denoting label of the i^{th} example as $y(i)$, FOM score of ordering π is computed as

$$\text{FOM}(\pi) = \frac{1}{n-1} \sum_{i=1}^{n-1} I\Big(y\big(\pi(i)\big) \neq y\big(\pi(i+1)\big) \Big), \tag{2.9}$$

where binary indicator function $I(\cdot)$ returns 1 if the argument is true, and 0 otherwise. As a result, lower values of FOM indicate higher-quality ordering π. To evaluate TSP-means, we also report tour cost L, equal to the sum of Euclidean distances between neighboring examples in the final ordering.

2.4.1 Validation of TSP-means

We first evaluated influence of parameter l in Algorithm 3. The results for *waveform* data set of size 10,000, for l ranging from 2 to 10, are given in Table 2.2, where we report number of calls to LK and k-means sub-routines, time required to solve a single TSP, as well as FOM and L performance measures. As can be seen, setting l

[1] http://cs.nyu.edu/∼roweis/lle/code.html, accessed November 2013

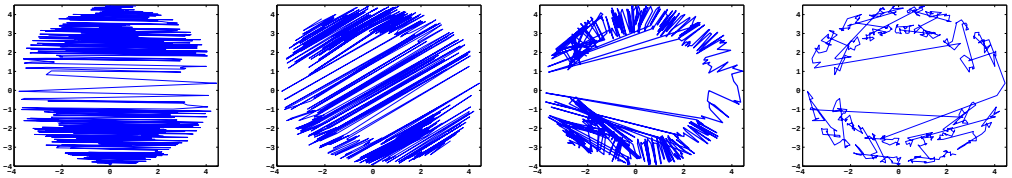
[2] http://www.tsp.gatech.edu/concorde.html, accessed November 2013

Table 2.2: Evaluation of performance on *waveform* data of size 10,000; listed LK time is required for solving one TSP

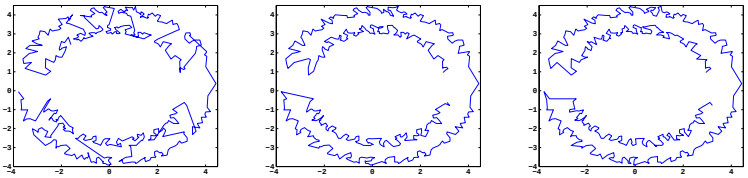
l	# LK	LK time [sec]	# k-means	FOM	L
2	4,721	0.000	5,224	0.234	31,663
4	2,496	0.002	3,541	0.235	31,244
6	2,304	0.035	3,604	0.233	30,925
7	533	0.067	1,369	0.229	29,840
8	257	0.253	255	0.228	30,823
9	504	0.516	506	0.233	30,420
10	985	1.093	1,019	0.230	30,007

to 7 led to good results. This can be explained by the fact that depth of the tree T (after step 1 in Algorithm 3) was 14, as expected since $\lceil \log_2(10{,}000) \rceil < 14$, meaning that the depth of T^l was only 2. For $l < 7$ the depth of T^l was larger than 2, leading to more calls to LK and k-means subroutines. On the other hand, for $l > 7$ the TSP of root node of T^l had 2^l children, while TSPs of non-root nodes had less than 2^{14-l} children, thus not fully exploiting near-optimal ordering found by LK at the lower levels. Taking this result into consideration, in the remaining experiments we set $l = \lceil 0.5 \ \log_2(n) \rceil$ as a default value.

To get an insight into relative performance of the competing algorithms, we generated 2-dimensional data set called *circles*. We uniformly at random sampled half of the examples from a circle of radius 3, and the second half from the circle of radius 4. We also added small noise to all examples. We set the size of the *circles* data set to 500 for the clarity of visualization. The resulting ordering of all methods is shown in Figure 2.2, where neighboring examples in the final linear ordering are connected by a line. Performance of PCA was expected (Figure 2.2a), as the method projects examples onto a straight line. We can see that LLE failed to find a desired principal curve that would consist of two connected circles (Figure 2.2b). As can be seen in Figure 2.5c, SC found better ordering than PCA and LLE and clearly separated upper and lower parts of the data set, but failed to find good ordering within the clusters. HC resulted in relatively smooth ordering with occasional jumps between



(a) PCA (1,131.9) (b) LLE (956.3) (c) SC (432.1) (d) HC (164.5)

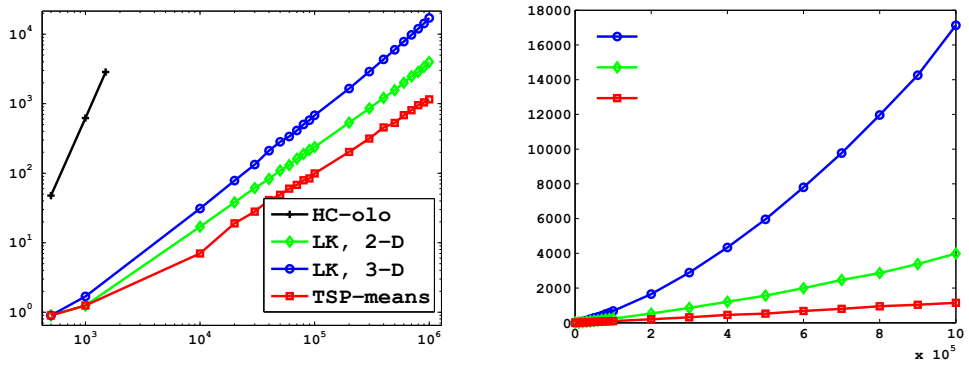


(e) HC-olo (94.5) (f) LK (83.2) (g) TSP-means (84.5)

FIGURE 2.2: Performance of ordering algorithms on *circles* data ($n = 500$, path length L is given in parentheses)

circles, see Figure 2.2d. We can see in Figure 2.2e that HC-olo reordered the HC tree to provide smoother ordering. As expected, LK had the shortest route, while the proposed TSP-means was very close to LK (Figures 2.2f and 2.2g, respectively).

To compare the execution times for HC-olo, LK and TSP-means, we uniformly sampled 2-D and 3-D examples from a square and a cube of width 1, respectively, and increased data size from 500 to 1,000,000. In Figure 2.3 we show times in both logarithmic and linear scale in order to better illustrate the performance of algorithms for small- and large-scale data sets. In Figure 2.3a we can see that HC-olo method required prohibitively long processing time as n increased, and that it could not process data sets with more than few thousand examples. For 1 million examples TSP-means completed in around 20 minutes for both 2- and 3-D data, while for LK it took around 1 and 5 hours, respectively. We note that HC-olo and TSP-means scale linearly with the data dimensionality. On the other hand, LK algorithm uses k-d trees to speed up computations in lower-dimensions, but for higher-dimensional data the benefit of k-d trees decreases sharply. That is why results for LK in Figure 2.3 are representative only for 2- and 3-D data, while the execution time scaling for



(a) Logarithmic scale (b) Linear scale

FIGURE 2.3: Execution times on 2-D and 3-D *uniform* data set

higher dimensions would be higher than shown and scale as $\mathcal{O}(n^{2.2})$. We did not run experiments for higher-D data since in that case the LK implementation we used requires distance matrix, which becomes infeasible for large n.

As observed in Biedl et al. (2001), although using LK on the whole data table results in the smallest tour length, it does not necessarily translate into the visually informative ordering. This can be explained by the fact that, unlike TSP-means, LK is not constrained by the underlying clustering structure in the data, rendering it sensitive to noisy examples. This is illustrated in Figure 2.4, where we generated 200 examples from each of the 2-D clusters representing 4 classes sampled from two-dimensional Gaussians centered at $(0, 0)$, $(0, 4.5)$, $(4.5, 0)$ and $(4.5, 4.5)$, with identity covariance matrices. We can see that LK achieved smaller tour length than TSP-means. However, TSP-means first visited examples from cluster 1, followed by cluster 2, then cluster 3, to finish by visiting examples from cluster 4 (see Figure 2.4b). On the other hand, in Figure 2.4a we see that LK jumped between clusters visiting them in the $\{3, 4, 1, 2, 1, 4, 3, 2, 3\}$ order, resulting in a lower quality of visualization. As discussed previously, this indicates that TSP-means accounts for clustering structure present in the data.

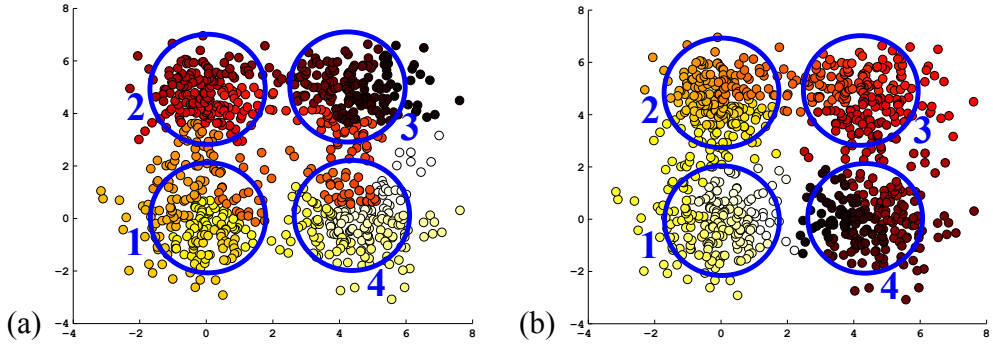


FIGURE 2.4: FOM and L measures on a 2-D toy data set (color encodes the order of an example in the ordered data set, ranging from white to black): (a) LK (0.038; 181.44); (b) TSP-means (0.033; 199.86)

2.4.2 Validation of EM-ordering

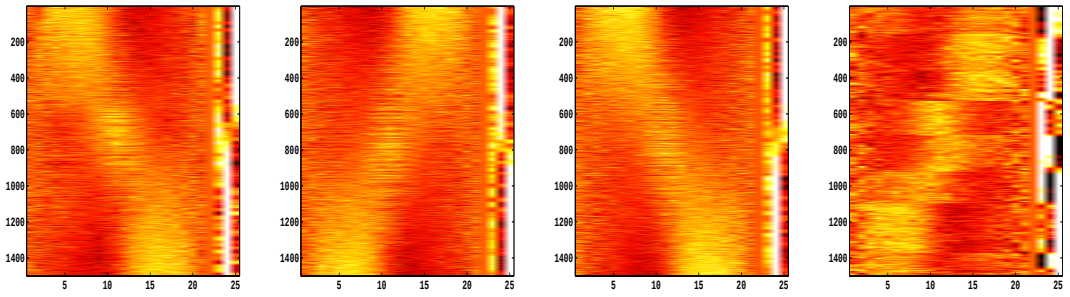
In this section, we present performance of algorithms on benchmark labeled sets from UCI repository. Noisy *waveform* was created by appending 21 noise features to *waveform*. All data sets were normalized to zero-mean and unit variance. We limited ourselves to data sets of size $n = 1{,}500$ in order to be able to compare our method with resource-intensive SC and HC methods, and report average FOM after 5 experiments.

In Table 2.3 we report FOM performance of ordering methods on 11 classification data sets. Interestingly, in nearly all tasks the three lower-dimensional projection methods (PCA, LLE, and SC) were significantly outperformed by the competing techniques. EM-ordering was best on 7 out of 11 data sets, with the additional advantage of being much faster than the closest competitors HC-olo and LK. Results after one iteration of Algorithm 1 were better than after five iterations in only two cases, indicating the benefits on feature scaling. Feature scaling was especially beneficial for *wine*, *madelon* and noisy *waveform*, where FOM dropped by the highest margin. Moreover, on *madelon* and noisy *waveform* data, for which irrelevant, noisy features were known beforehand, EM-ordering detected more than 90% of noisy features.

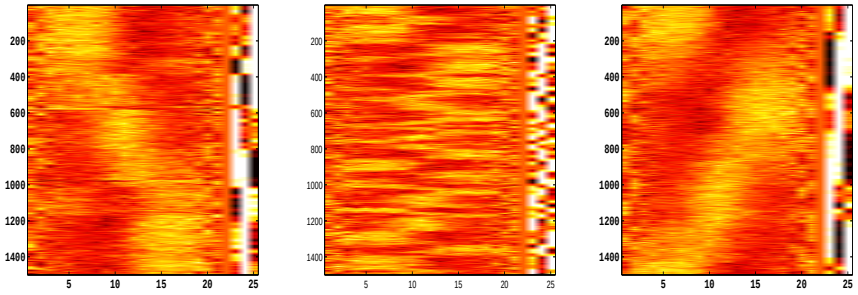
Table 2.3: FOM scores for benchmark data sets (EM-1 and EM-5 denote EM-ordering after 1 and 5 iterations of Algorithm 1, respectively, baseline result is FOM before reordering; also shown size n, dimensionality m, and number of classes c)

data set	n	m	c	baseline	PCA	LLE	SC	HC	HC-olo	LK	EM-1	EM-5
iris	150	3	3	0.637	0.188	0.161	0.187	0.181	0.134	**0.114**	0.121	**0.114**
wine	178	13	3	0.649	0.225	0.531	0.056	0.062	0.056	0.045	0.047	**0.032**
breast	277	9	2	0.413	0.337	**0.324**	0.330	0.344	0.340	0.344	0.339	0.338
adult	1,500	123	2	0.374	0.267	n/a	0.235	0.276	0.267	0.252	0.238	**0.232**
banana	1,500	2	2	0.514	0.348	0.356	0.279	0.151	0.152	**0.147**	**0.147**	**0.148**
covertype	1,500	54	7	0.630	0.620	0.612	0.583	0.430	0.395	**0.382**	0.424	0.438
gauss	1,500	2	2	0.491	0.316	0.286	0.286	0.263	0.269	**0.260**	0.266	0.267
madelon	1,500	500	2	0.506	0.447	0.494	0.489	0.464	0.449	0.444	0.439	**0.399**
magic	1,500	10	2	0.464	0.416	0.451	0.360	0.258	**0.235**	0.243	0.244	0.259
waveform	1,500	21	3	0.680	0.462	0.461	0.461	0.266	0.250	0.249	0.239	**0.238**
wave noisy	1,500	42	3	0.680	0.472	0.493	0.466	0.344	0.309	0.310	0.282	**0.237**

In Figure 2.5 we show heatmaps of reordered *waveform* data set, where rows correspond to examples, columns correspond to features, and color intensity represents a numeric value. As the data set has 3 classes, in the 3 rightmost columns we encode the class membership of examples. We averaged class membership over a column-wise sliding window of length 20 for easier interpretation of the results. Darker pixels indicate that examples within the window had the same label, thus indicating successful ordering. As seen in Figures 2.5a, 2.5b, and 2.5c, lower-dimensional projection methods PCA, LLE, and SC obtained similar visualization results with very smooth-appearing heatmaps, but FOM results and the last three columns suggest that they did not provide compact grouping of examples with the same class labels. In contrast, HC and HC-olo resulted in better ordering, as shown in Figures 2.5d and 2.5e, respectively. However, similarly to the results in Figure 2.2, tours often jumped between classes, reducing FOM and the quality of visualization. LK algorithm found the shortest path, as illustrated in Figure 2.5f, which did not translate into good ordering. LK frequently jumped between classes, resulting in visually unappealing ordering. On the other hand, EM-ordering had the best FOM, as can be seen from Figure 2.5g.



(a) PCA (0.462; 7,231) (b) LLE (0.461; 7,211) (c) SC (0.461; 7,244) (d) HC (0.266; 5,265)



(e) HC-olo (0.250; 4,817) (f) LK (0.249; 4,577) (g) EM-ordering (0.239; 4,921)

FIGURE 2.5: Visualization of *waveform*, the last 3 columns are class assignments averaged over sliding window of length 20; FOM and L measures given in parentheses

2.4.3 Applications of EM-ordering

To illustrate the usefulness of ordering, we applied EM-ordering on two real-world data sets. The first is a set of traffic volumes (number of cars per minute) reported every 10 minutes by 3,265 sensors on highways in Minneapolis, MN, on December $23^{\text{rd}}, 2005$. The original, unordered data set is shown in Figure 2.6a, from which little can be seen beyond presence of heavy traffic volume during early morning and late afternoon. In Figure 2.6b we show an ordered data matrix, where it becomes clear that there are several types of traffic patterns (heavy traffic during morning or afternoon only, light or heavy traffic during most of the day, etc.). To further illustrate the benefits of ordering, upon visual inspection we split the ordered sensors manually into 11 clusters. In Figure 2.6c we show the geographical locations of

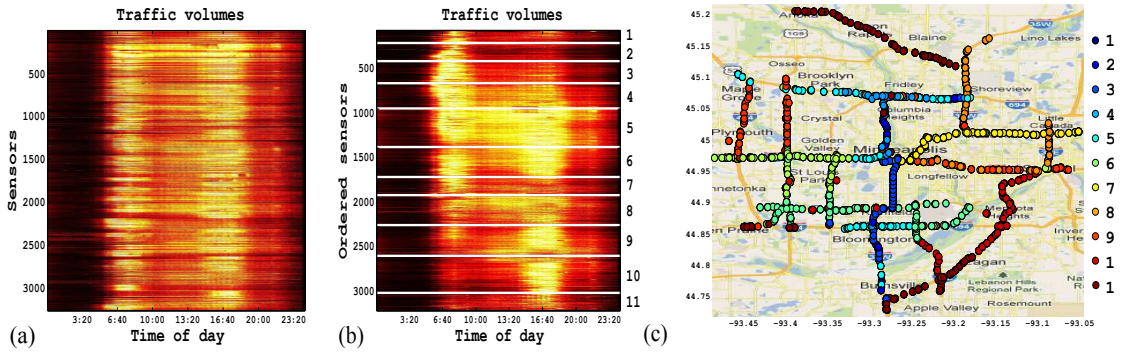


FIGURE 2.6: Traffic data: (a) the original data set (brighter pixels denote higher volumes); (b) the ordered data set (white lines denote user-selected cluster boundaries); (c) color-coded sensor locations in Minneapolis road network (neighboring clusters were assigned similar colors)

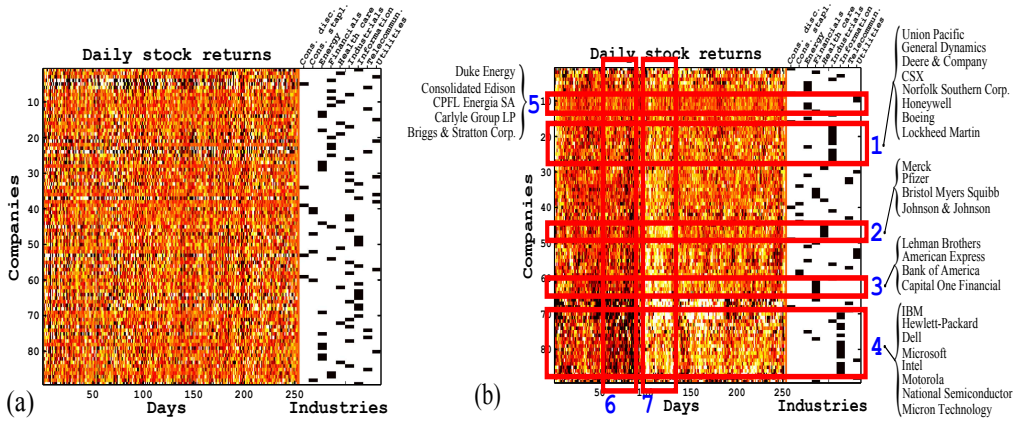


FIGURE 2.7: *stocks* data set (9 rightmost columns encode industries; dark pixels in the heatmap encode high negative returns, while bright pixels encode high positive returns): (a) the original data set; (b) the data set after reordering and clustering upon inspection of the heatmap

sensors, colored according to their cluster labels. We can see that the sensors along the same road segments were clustered together, and that nearby road segments were assigned to similar clusters. This example illustrates how ordering can be used for interactive exploration and visualization of data, which could be very useful to traffic engineers and planners. We note that this modestly-sized data set cannot be ordered using HC-olo and LK algorithms on a regular computer.

We also ran EM-ordering on *stocks* data set, representing 252 daily stock returns

of 89 companies from 9 sectors of industry. Not much can be seen from the original data set shown in Figure 2.7a (the companies were sorted alphabetically). After reordering rows and columns of the data table interesting patterns emerged, as seen in Figure 2.7b, which may provide useful insights to stock traders. We can observe that the ordering revealed several clusters of companies operating in industrials, health care, financials, and information technologies sectors (clusters 1 - 4, respectively), having specific patterns of daily returns. We also detected companies from energy and utilities sectors in cluster 5, whose daily returns, unlike returns of the companies from other sectors, did not fluctuate much. Lastly, after reordering the transposed data matrix (i.e., ordering days instead of companies), bear and bull trading days can be easily detected (clusters 6 and 7, respectively).

CHAPTER 3

VISUALIZATION THROUGH OBJECT MATCHING

3.1 Introduction

Object matching has been recognized as an important problem in many areas of machine learning. The problem can be described as follows: given two sets of objects, match the objects from the first set to the objects in the second set so that they are "similar". Applications and research areas as diverse as graph matching in computer vision (Luo and Hancock, 2001), document alignment in natural language processing (Jagarlamudi et al., 2010), and multiple sequence alignment in bioinformatics (Bacon and Anderson, 1986), all rely on the assumption that there is an underlying correspondence between two sets of objects (possibly originating from two quite different domains) which can be learned. Object matching is also closely related to transfer learning (Wang and Yang, 2011), as the cross-domain alignments can be used to transfer knowledge to new domains. This common thread can be extended to many other areas, and a number of recently published works on the topic indicates the importance of the matching problem.

Assuming that the objects from two sets are directly comparable, the task amounts to finding matches so that certain cross-domain measure of similarity is maximized. However, if the objects originate from different domains and are represented in different feature spaces, specifying a similarity measure can be very challenging. For instance, if we have a set of documents in French and a set of documents in Chinese language, it is not readily obvious how to find similar documents without a bilingual dictionary. Several ideas have been proposed to solve this unsupervised problem. Some approaches follow a simple rationale: match two objects that have comparable local neighborhoods, each in their own domain. A direct realization of this idea is Manifold Alignment (MA) (Wang and Mahadevan, 2009), where the information about local relationships is translated into cross-domain measure of similarity. Following different line of reasoning, Haghighi et al. (2008) and Tripathi et al. (2011) describe a method that aligns objects by maximizing the correlation between two sets using Canonical Correlation Analysis (Hotelling, 1936). In order to find one-to-one alignments they propose an EM-style algorithm, which iteratively finds new canonical variables and the best alignments in a new common feature space shared by the two groups of objects until convergence.

Following the idea from MA, Kernelized Sorting (KS) (Quadrianto et al., 2010) tries to match objects with similar neighborhoods. However, unlike MA that uses only information about local context, KS method computes two kernel matrices, one for each object set. Then, using the kernel matrices, an alignment of objects across domains is found so that the dependency between matched pairs is maximized, measured in terms of Hilbert-Schmidt Independence Criterion (HSIC). Although conceptually very powerful, the technique is highly unstable as the object matching problem is defined as *maximization* (not minimization) over a convex function, subject to linear constraints. This renders KS algorithm very vulnerable to local optima and poor initialization. In Jagarlamudi et al. (2010), authors propose several mod-

ifications to the original algorithm to obtain a more stable method, but the local optima issue persists nevertheless. Note that some authors propose maximization of independence criteria other than HSIC (Yamada and Sugiyama, 2011), but as the optimization problem is essentially unmodified, the methods are still plagued with the same instability problems. To solve this issue, we propose a convex formulation of the object matching problem, resulting in a robust algorithm guaranteed to find the optimal solution.

3.2 Kernelized Sorting

Let $X = \{x_1, x_2, ..., x_m\}$ and $Y = \{y_1, y_2, ..., y_m\}$ be two sets of equal size m of objects from two possibly different domains $\mathcal{X}$ and $\mathcal{Y}$, respectively. Further, let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be two kernels associated with domains $\mathcal{X}$ and $\mathcal{Y}$. Given kernel matrices K and L capturing the relationships between objects within X and Y sets, namely $K_{ij} = k(x_i, x_j)$ and $L_{ij} = l(y_i, y_j)$, the task is to find a one-to-one correspondence between objects in X and objects in Y. The cross-domain correspondence is encoded in $m \times m$ permutation matrix $\pi \in \Pi_m$, where $\pi_{ij} = 1$ if objects x_j and y_i are matched, $\pi_{ij} = 0$ otherwise, and Π_m is the set of all $m \times m$ permutation matrices. In the remainder of the paper, we will use notation π to denote both the permutation matrix and the corresponding object alignment, where the usage should be clear from the context.

In Quadrianto et al. (2010), authors propose a method to find alignment π by maximizing the dependency between K and L as measured by Hilbert-Schmidt Independence Criterion. Given two random variables x and y drawn from some joint probability distribution $\mathbb{P}_{xy}$, HSIC (Smola et al., 2007) measures the dependence between variables x and y by computing the Hilbert-Schmidt norm Δ^2 of the cross-covariance operator between Reproducing Kernel Hilbert Spaces (RKHS) on $\mathcal{X}$ and $\mathcal{Y}$, and the norm equals 0 if and only if variables x and y are independent. If we de-

fine the projection matrix $H = I - 1/m$, which centers the data in the feature space, to obtain centered versions of K and L as $\overline{K} = HKH$ and $\overline{L} = HLH$, respectively, the Hilbert-Schmidt norm can be empirically estimated (Smola et al., 2007) as

$$\Delta^2 = m^{-2} \cdot \text{trace}(\overline{K} \cdot \overline{L}). \tag{3.1}$$

In order to find the best correspondence between objects across two domains, we seek such permutation π^* of rows and columns of kernel matrix $\overline{L}$ (i.e., the alignment between two sets of objects) that maximizes the dependency (3.1) of two kernel matrices. Thus, we obtain the following optimization problem,

$$\pi^* = \arg\max_{\pi \in \Pi_m} \text{trace}(\overline{K}\pi^{\mathrm{T}}\overline{L}\pi). \tag{3.2}$$

The optimization problem is an instance of quadratic assignment problem, therefore NP-hard in general, and the authors propose an iterative, approximate method for finding the matrix π^*. Given permutation matrix π_i at i^{th} iteration, the optimization problem (3.2) is converted into Linear Assignment Problem (LAP) and solved for optimal π_{i+1} using one of the available LAP solvers (Kuhn, 1955; Jonker and Volgenant, 1987). Note that the optimization problem (3.2) involves *maximization* over convex function of π. As a result, the optimization procedure is unstable and highly sensitive to initial value of the permutation matrix π_0 due to local optima issues.

Instead of HSIC, other independence criteria can also be used. In Yamada and Sugiyama (2011), authors propose two variants of KS algorithm, using normalized cross-covariance operator and least-squares mutual information as measures of dependence between matrices K and L. However, as both the optimization problem and the optimization procedure are essentially the same as the ones used in KS method from Quadrianto et al. (2010), the algorithms are still susceptible to local optima issues.

In order to mitigate the instability issue of KS methods, authors of Jagarlamudi et al. (2010) describe two modifications to the original algorithm. Although the authors were motivated by the problem of application of KS to Natural Language Processing (NLP), it is straightforward to use their method in other areas as well. The first improvement they propose is p-smooth, which involves using sub-polynomial kernel on the original kernel matrices. Given a kernel matrix M, each element of a kernel matrix is raised to the power of p (with $0 < p \leqslant 1$) to obtain a matrix M_p. This is followed by normalization of rows to unit length and calculation of a new kernel matrix as $M \leftarrow M_p \cdot M_p^{\mathrm{T}}$. The second improvement is to use a set of seed alignments, which will be favored during the optimization procedure. In this way, the effect of low-confidence, thus possibly incorrect alignments can be reduced during the optimization process. Although the modifications result in considerably more robust algorithm (with the cost of introducing an additional parameter p), the issue of local optima due to non-convex optimization problem remains.

3.3 Convex Kernelized Sorting

In this section, we present a method that is, unlike Kernelized Sorting described in the previous section, guaranteed to find a global optimum solution. We begin by observing that, given two $m \times m$ matrices $\overline{K}$ and $\overline{L}$, value of trace$(\overline{K} \cdot \overline{L})$ is maximized if rows and columns of $\overline{K}$ and $\overline{L}$, respectively, are permuted such that rows of $\overline{K}$ and corresponding columns of $\overline{L}$ are identical up to a constant multiplier. Then, we can define a convex optimization problem for finding the optimal permutation matrix π^* as a minimization of the difference between the left half and transpose of the right half of matrix product under trace operator in (3.2), or more formally

$$\underset{\pi \in \Pi_m}{\text{minimize}} \quad ||\overline{K} \cdot \pi^{\mathrm{T}} - (\overline{L} \cdot \pi)^{\mathrm{T}}||_F^2, \tag{3.3}$$

where $||\cdot||_F$ is Frobenius (or Hilbert-Schmidt) matrix norm. The product $\overline{K} \cdot \pi^{\mathrm{T}}$ permutes columns of $\overline{K}$, while $(\overline{L} \cdot \pi)^{\mathrm{T}}$ permutes rows of $\overline{L}$, effectively aligning objects in the same way as when both rows and columns of only one of the kernel matrices are permuted. The optimization problems defined in equations (3.2) and (3.3) are equivalent, as shown by the following lemma.

Lemma 1. *The problems (3.2) and (3.3) are equivalent.*

Proof. Let $A = \overline{K} \cdot \pi^{\mathrm{T}}$ and $B = \overline{L} \cdot \pi$, and let a_{ij} and b_{ij} be values in i^{th} row and j^{th} column of A and B. Further, let $\mathcal{L}_1 = ||\overline{K} \cdot \pi^{\mathrm{T}} - (\overline{L} \cdot \pi)^{\mathrm{T}}||_F^2$ and $\mathcal{L}_2 = \mathrm{trace}(\overline{K}\pi^{\mathrm{T}}\overline{L}\pi)$. Then, it follows

$$\begin{aligned}
\mathcal{L}_1 = &\sum_{i=1}^{m}\sum_{j=1}^{m}(a_{ij} - b_{ji})^2 = \\
&\sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij}^2 + \sum_{i=1}^{m}\sum_{j=1}^{m} b_{ji}^2 - 2 \cdot \sum_{i=1}^{m}\sum_{i=1}^{m}(a_{ij} \cdot b_{ji}) = \\
&\sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij}^2 + \sum_{i=1}^{m}\sum_{j=1}^{m} b_{ji}^2 - 2 \cdot \mathcal{L}_2.
\end{aligned}$$

The first two elements of the final sum do not depend on π, as right multiplication with permutation matrix only permutes columns while not changing the values of the original elements of the matrix. Consequently, it follows

$$\arg\min_{\pi \in \Pi_m} \mathcal{L}_1 = \arg\max_{\pi \in \Pi_m} \mathcal{L}_2.$$

□

In order to obtain a convex optimization problem, we relax the constraint that π is strictly a permutation matrix, and define the following problem (by $\mathbf{1}_m$ we denote

an m-dimensional column vector of all ones)

$$\begin{aligned} \underset{\pi}{\text{minimize}} \quad & ||\overline{K} \cdot \pi^{\mathrm{T}} - (\overline{L} \cdot \pi)^{\mathrm{T}}||_F^2 \\ \text{subject to} \quad & \pi_{ij} \geqslant 0, \text{for } i, j \in \{1, 2, ..., m\}, \\ & \pi \cdot \mathbf{1}_m = \mathbf{1}_m, \\ & \pi^{\mathrm{T}} \cdot \mathbf{1}_m = \mathbf{1}_m, \end{aligned} \tag{3.4}$$

where the permutation matrix binary constraint $\pi_{ij} \in \{0, 1\}$ is replaced by the interval constraint $\pi_{ij} \in [0, 1]$. The constraints in (3.4) ensure that the matrix π is doubly-stochastic (permutation matrix is a special case), meaning that it is a matrix with positive elements with both rows and columns summing up to 1. Thus, we are directly solving the problem (3.2) by converting it into convex problem (3.4). The resulting optimization problem is convex because the objective function is a composition of convex non-decreasing square function and convex norm function with affine argument, subject to linear equality and inequality constraints.

The relaxation of the constraint which restricts π to be strictly permutation matrix comes with several advantages. Mainly, as shown above, the problem of finding cross-domain object alignments can be stated as convex optimization problem, lending itself to simpler optimization, effective computational methods, and globally optimal solution (Boyd and Vandenberghe, 2004). In this way we also avoid sensitivity to initialization from which the method from Quadrianto et al. (2010) suffered. In addition, soft assignments sum up to 1 and, consequently, assignment π_{ij} can now be interpreted as a probability that objects x_j and y_i match. Therefore, the output of the algorithm is more informative than hard alignments, giving us a degree by which two objects should be aligned.

In order to numerically solve the problem (3.4), we compute first and second derivatives, with respect to the elements of matrix π, of the objective function $\mathcal{L} =$

$||\overline{K} \cdot \pi^{\mathrm{T}} - (\overline{L} \cdot \pi)^{\mathrm{T}}||_F^2$. The gradient can be calculated using

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \pi_{ij}} = 2 \cdot \sum_{k=1}^{m} \Big(-\overline{L}_{ki} \cdot \sum_{l=1}^{m} (\overline{K}_{jl} \cdot \pi_{kl} - \pi_{lj} \cdot \overline{L}_{kl}) + \\ \overline{K}_{kj} \cdot \sum_{l=1}^{m} (\overline{K}_{kl} \cdot \pi_{il} - \pi_{lk} \cdot \overline{L}_{il}) \Big),\end{aligned} \tag{3.5}$$

while a Hessian matrix of the objective function $\mathcal{L}$ can be calculated using

$$\begin{aligned}\frac{\partial^2 \mathcal{L}}{\partial \pi_{ij} \partial \pi_{rn}} = 2 \cdot \Bigg(\sum_{k=1}^{m} \Big(I(i = r) \cdot \overline{K}_{kj} \cdot \overline{K}_{kn} + \\ I(j = n) \cdot \overline{L}_{ki} \cdot \overline{L}_{kr} \Big) - 2 \cdot \overline{K}_{jn} \cdot \overline{L}_{ir} \Bigg),\end{aligned} \tag{3.6}$$

where $i, j, r, n \in \{1, 2, ..., m\}$, and $I(arg)$ is a binary indicator function equal to 1 if arg is true, and 0 otherwise.

3.3.1 Numerical optimization

The optimization problem (3.4) can be easily solved using one of the high-level optimization packages such as `CVX`[1], a package for specifying and solving convex programs. However, since the number of variables scales as $\mathcal{O}(m^2)$, this and other similar solvers are not suitable if we want to align larger sets of objects. To make the numerical optimization more scalable, we present another convex variant of the optimization problem (3.4). In order to reduce the number of constraints and retain only inequalities, we reformulate (3.4) into the following dual problem

$$\begin{aligned}\underset{\pi}{\text{minimize}} \quad & ||\overline{K} \cdot \pi^{\mathrm{T}} - (\overline{L} \cdot \pi)^{\mathrm{T}}||_F^2 + \\ & C \cdot \sum_{k=1}^{m} \left((\sum_{l=1}^{m} \pi_{kl} - 1)^2 + (\sum_{l=1}^{m} \pi_{lk} - 1)^2 \right) \\ \text{subject to} \quad & \pi_{ij} \geqslant 0, \text{for } i, j \in \{1, 2, ..., m\},\end{aligned} \tag{3.7}$$

[1] http://cvxr.com/cvx/, accessed November 2013

where we set regularization parameter C to some large value (we set $C = 10$ in our experiments). Then, since we now have only inequalities without any equality constraints, we can use the trust-region-reflective algorithm (Coleman and Li, 1996) for solving convex problem (3.7) (algorithm implemented in, e.g., `fmincon` solver available in Matlab). With this modification, the gradient and the Hessian of the objective function are slightly different than (3.5) and (3.6). More specifically, $2 \cdot C \cdot (\sum_{k=1}^{m}(\pi_{jk} + \pi_{ki}) - 2)$ is added to (3.5), while Hessian is modified by adding $4 \cdot C$ to diagonal elements, and $2 \cdot C$ to off-diagonal elements if $(i = r)$ or $(j = n)$ in (3.6).

It is important to mention that, due to the nature of the trust region method and convexity of the optimization problem, approximate, diagonal-only Hessian can be used, where all off-diagonal elements are set to 0. Use of the exact Hessian leads to fewer iterations until convergence of the trust-region-reflective algorithm, while, on the other hand, the use of approximate Hessian requires significantly less memory and floating-point operations per iteration to reach globally optimal solution.

Note that after solving the convex problem (3.7), matrix π is not necessarily doubly-stochastic, although sums of rows and sums of columns will be very close to 1. Therefore, as a final step in our algorithm, we can project matrix π, the solution of problem (3.7), to the set of doubly-stochastic matrices. Finding this closest projection is an old problem, and we can use one of the proposed, approximate methods (Sinkhorn and Knopp, 1967; Parlett and Landis, 1982). Alternatively, we can define an additional convex optimization problem in order to find the optimal, closest doubly-stochastic matrix π^{DS} as follows

$$\begin{aligned} \underset{\pi^{DS}}{\text{minimize}} \quad & ||\pi - \pi^{DS}||_F^2 \\ \text{subject to} \quad & \pi_{ij}^{DS} \geqslant 0, \text{for } i, j \in \{1, 2, ..., m\}, \\ & \pi^{DS} \cdot \mathbf{1}_m = \mathbf{1}_m, \\ & (\pi^{DS})^{\mathrm{T}} \cdot \mathbf{1}_m = \mathbf{1}_m, \end{aligned} \tag{3.8}$$

which can be solved as a constrained linear least-squares problem or constrained Euclidean norm, and an optimal solution can be found by a number of solvers (e.g., `SeDuMi`[2] or `lsqlin` solver available in Matlab).

3.3.2 Soft and hard assignments

Soft assignments contained in matrix π provide deeper understanding of the cross-domain alignment between two sets of objects. As such, output of Convex Kernelized Sorting (CKS) can be used to rank the objects by sorting them by the probabilities that they should be aligned. For instance, if we are matching documents in French with documents written in Chinese language, for each French document we can provide a user with top 5 matches together with the degree showing how confident about each match are we.

Similarly to the existing methods, CKS can also give hard alignments, defined as a one-to-one correspondence between two sets of objects. In order to obtain the hard alignments, we use the Hungarian algorithm (Kuhn, 1955) to solve LAP defined by the learned doubly-stochastic matrix π. The Hungarian algorithm is a polynomial-time algorithm for combinatorial optimization, efficiently solving the linear assignment problem. More specifically, if we have m workers each requiring certain pay to work on one of m jobs, the algorithm finds the lowest-cost one-to-one assignment of workers to jobs in $\mathcal{O}(m^3)$ time.

Finally, quality of the obtained alignments greatly depends on the choice of kernel functions k and l (Yamada and Sugiyama, 2011). Using CKS we can visualize the alignments and thus provide a user with additional evidence that appropriate kernels were used. For instance, if Gaussian kernel is used to generate kernel matrices K and L, choice of kernel-width parameter will critically influence the alignment. One simple way to verify that appropriate value of the parameter is chosen is to plot

[2] http://sedumi.ie.lehigh.edu/, accessed June 2013

matrix π (e.g., using function `imagesc` in Octave and Matlab) and visually check if there are clear matches between objects. By visualization of the results, a user can be more confident that appropriate kernel functions and parameters are used.

3.4 Experiments

In this section, we empirically evaluate performance of our CKS algorithm. We used the trust-region-reflective algorithm to solve convex problem (3.7), and stopped the optimization either after 10,000 iterations or when objective function stopped decreasing by more than 10^{-6}. In all our experiments we used the diagonal approximation of the Hessian, as experiments showed that this leads to substantial gains in memory and time. Unlike the competing algorithms, CKS does not depend on the initialization of π, and as an initial point for the trust-region-reflective algorithm we simply set $\pi = \mathbf{1}_m \cdot \mathbf{1}_m^{\mathrm{T}}/m$. To find hard alignments, we used implementation of the Hungarian algorithm available online[3]. Lastly, we note that the projection of nearly doubly-stochastic matrix π to the set of doubly-stochastic matrices was not necessary, as we found that this additional step did not significantly affect the results.

We compared CKS performance to Kernelized Sorting algorithm from Quadrianto et al. (2010), and to p-smooth Kernelized Sorting (KS-p) method from Jagarlamudi et al. (2010). We also compared our algorithm to Least-Squares Object Matching method[4] (LSOM) (Yamada and Sugiyama, 2011), which maximizes least-squares mutual information criterion instead of HSIC. Except for CKS, which requires only a single run, all experiments were repeated 10 times with both random and PCA-based initializations of matrix π, and the average and the best accuracy are reported. For

[3] http://www.mathworks.com/matlabcentral/fileexchange/20652-hungarian-algorithm-for-linear-assignment-problems-v2-3, accessed November 2013

[4] code from sugiyama-www.cs.titech.ac.jp/~yamada, accessed November 2013

each run of p-smooth KS method we tried all p values from 0.1 to 1 in increments of 0.1, and used the best performance as a result for that run. Manifold Alignment algorithm (Wang and Mahadevan, 2009) was also considered, but this local-neighborhood method performed poorly in the experiments, which can be explained by the high dimensionality of the matching tasks.

First, we show application of CKS to data visualization, and then explore performance on the task of alignment of left and right halves of images from a set of images. Here, we used a set of 320 images from Quadrianto et al. (2010). The images were processed in the same manner as in Quadrianto et al. (2010). Namely, images were resized to 40×40 pixels, then converted from RGB to LAB color space and vectorized. For experiments in which we used the image set, kernel matrices were generated using Gaussian RBF kernel $k(x, x') = l(x, x') = exp(-\gamma \cdot ||x - x'||^2)$, setting γ to inverse median of $||x - x'||^2$. Finally, application to document alignment is presented, where we used linear kernel function to compute the kernel matrices.

3.4.1 Data visualization

One of possible applications of CKS is data visualization. Although we can use the existing algorithms for low-dimensional projection of high-dimension data (e.g., Principal Component Analysis), they cannot be used if we want to project data to arbitrary fixed structures (Quadrianto et al., 2010). KS methods, on the other hand, can be used for projection of data in such cases.

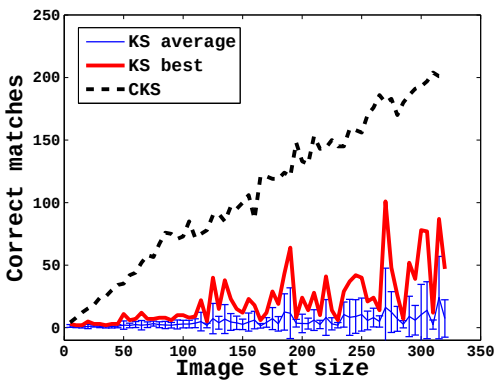
Figure 3.1 shows a low-dimensional projection of 320 images from the data set to a 2-dimensional "AAAI 2102" letter grid. The pattern in the layout is discernible, as similar images are closer in the grid, and lighter images are located near the middle of the grid while darker ones are closer to the edges.

FIGURE 3.1: Alignment of 320 images to "AAAI 2012" letter grid

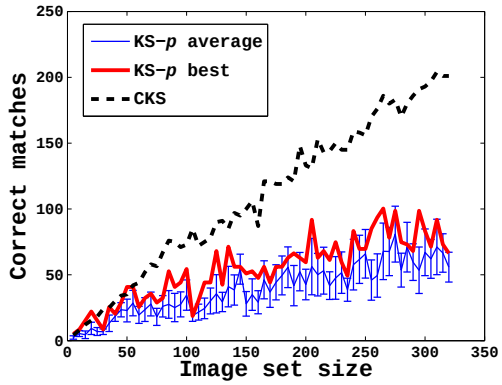
3.4.2 Image alignment

As done in Quadrianto et al. (2010), we split the 320 images into left and right image halves. The task is to align a set of left halves with the set of right halves, and measure how many halves are correctly matched. We increased the number of images from 5 to 320 in increments of 5, and the performances of 4 algorithms for all image set sizes are given in Figure 3.2. It can be seen that CKS consistently recovers more original images than the competing algorithms. The problem with local optima of KS method is clearly illustrated in Figure 3.2a. The results for two consecutive image set sizes are very inconsistent, due to the sensitivity of the method. The modifications introduced by KS-p result in more robust algorithm (Figure 3.2b), but the method still achieves smaller number of correct matches than both LSOM and CKS. LSOM method is for most image set sizes quite competitive when considering the best achieved accuracy, although the average performance and the wide confidence intervals indicate the instability of the method, see Figure 3.2c. Figure 3.3 illustrates the outcome of matching of all 320 images using CKS, which had 64.38% matching accuracy after applying the Hungarian algorithm to compute hard assignments.

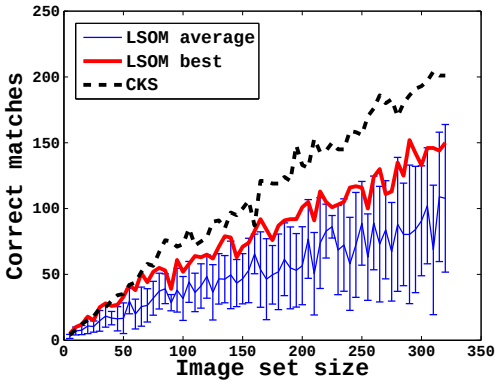
One of the advantages of CKS is that it outputs soft assignments. This can be used to rank the alignments by a probability of a match, and is illustrated in Figure



(a) CKS vs. KS



(b) CKS vs. KS-p



(c) CKS vs. LSOM

FIGURE 3.2: Number of correctly recovered images for different set sizes (error bars represent confidence intervals of one standard deviation, calculated after 10 runs)

3.4a where we calculated recall for top N ranks. If we sort the soft assignments, then 81.25% of correct image pairs are ranked among the top 5, while for the top 13 matches the recall jumps to nearly 90%. Furthermore, for each left image half, Figure 3.4b shows the maximum learned probability among right image halves versus the learned probability of the correct match (as a reference we plot $y = x$ line). As can be seen, the probability of the correct match is either the highest, or is very close to the highest probability.

Finally, in order to visually verify quality of the alignment and evaluate appropriateness of kernel parameters, we can also plot the learned matrix π (see Figure

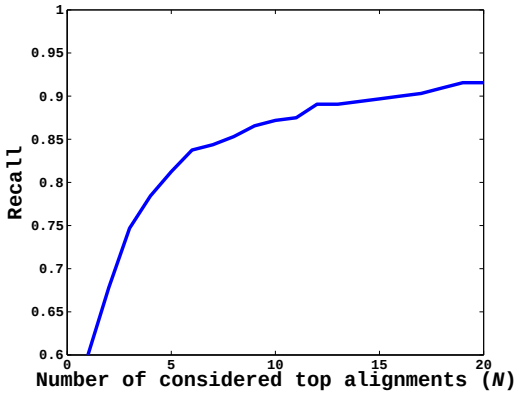
FIGURE 3.3: Alignment results of CKS for matching of 320 left and right image halves (206 correct matches)

3.4c, darker color corresponds to higher probability of alignment). To make the result more clear, we arranged two sets of image halves in such a way so that the i^{th} image in the left-halves set should be matched to the i^{th} image in the right-halves set. Consequently, the perfect matrix π would be equal to the identity matrix. As shown in Figure 3.4c, CKS recovered π very close to the identity matrix. Too few or too many dark peaks in the plot can be used as a visual clue that the choice of parameters might be wrong, thus providing an additional, visual insight into the matching problem.

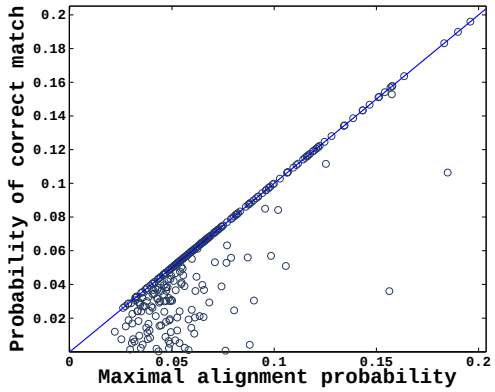
3.4.3 Multilingual document alignment

We evaluate CKS on the task of aligning multilingual documents. We used the Europarl Parallel Corpus[5] (Koehn, 2005), a collection of proceedings of the European Parliament available in 10 different European languages. The objective was to align a set of English documents with a set of corresponding documents written in other

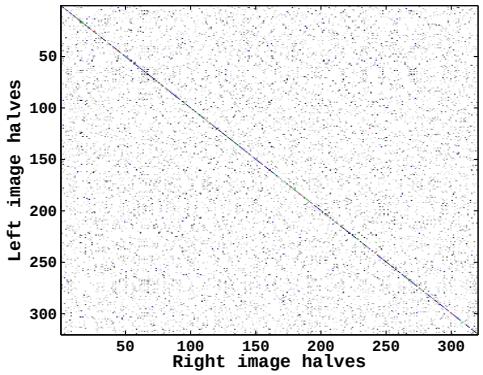
[5] http://www.statmt.org/europarl/archives.html, version 1, accessed November 2013



(a) Recall for top N-ranked alignments



(b) Maximal learned probability vs. correct-match probability (each circle represents a single object)



(c) Matrix π for alignment of 320 image halves

FIGURE 3.4: Analysis of CKS performance on the task of aligning 320 left and right images halves

languages. As a preprocessing step, stopwords were removed and stemming was performed using Snowball[6]. Then, the documents were represented using TF-IDF features of a bag-of-words kernel, followed by ℓ_2 normalization of feature vectors to unit length. As the resulting kernel matrices were notably diagonally dominant, for KS method we zeroed-out the diagonal, as suggested by authors in Quadrianto et al. (2010). For the remaining algorithms, no post-processing was done and the original kernel matrices were used.

[6] http://snowball.tartarus.org/, accessed November 2013

Table 3.1: Average number of correctly aligned non-English documents to documents in English (numbers in the parentheses are the best obtained performances; baseline method matches the documents according to their lengths)

Language	**Corpus size**	Baseline	KS	KS-p	LSOM	CKS
Danish	387	39	261 (318)	258 (273)	159 (173)	**379**
Dutch	387	50	266 (371)	237 (317)	146 (375)	**383**
Finnish	308	54	19 (32)	22 (38)	10 (10)	**114**
French	356	64	319 (356)	320 (334)	354 (354)	**356**
German	356	50	282 (344)	258 (283)	338 (350)	**356**
Italian	387	49	341 (382)	349 (353)	378 (381)	**385**
Portuguese	356	46	308 (354)	326 (343)	342 (356)	**356**
Spanish	387	48	342 (365)	351 (364)	386 (387)	**387**
Swedish	337	76	20 (39)	20 (33)	5 (5)	**97**

The results are presented in Table 3.1, where we report the rounded average and the best achieved (given in parentheses) number of correctly aligned documents after 10 runs. As a baseline reference, we calculated the performance of a simple method which aligns the documents according to their lengths. We can see that CKS algorithm recovered more correct matches than all other competing methods on all 9 NLP tasks. Even if we consider only the best performances achieved by the competing algorithms, our method still outperforms the competition. Considering that CKS algorithm requires only a single run, this is an impressive result. Furthermore, for most languages our method found nearly all the correct matches, except on the tasks of alignment of documents written in Finnish and Swedish languages. Although the two Nordic languages proved to be very challenging for all methods, CKS achieved around 5 times more correct alignments than the previously proposed algorithms. Interestingly, on these two tasks the baseline method is the second best, actually performing better than the existing algorithms.

CHAPTER 4

ONLINE LEARNING OF MULTI-HYPERPLANE CLASSIFICATION MODELS

4.1 Introduction

In the environment where new large-scale problems are emerging in various disciplines and pervasive computing applications are becoming more common, there is an urgent need for machine learning algorithms that can achieve high accuracy in computationally efficient way. Support Vector Machines (SVMs) (Cortes and Vapnik, 1995) provide a powerful paradigm for solving complex classification problems. However, this ability comes at the price of significant computational costs. There has been active research over the last decade to improve efficiency of SVM training (Vishwanathan et al., 2003; Zhang, 2004; Bordes et al., 2005; Tsang et al., 2005; Collobert et al., 2006; Joachims, 2006; Rahimi and Recht, 2007; Shalev-Shwartz et al., 2007b; Hsieh et al., 2008; Bordes et al., 2009; Zhu et al., 2009; Teo et al., 2010; Wang and Vucetic, 2010b; Chang et al., 2010; Sonnenburg and Franc, 2010; Yu et al., 2010; Wang et al., 2010) and today's state-of-the-art solvers are able to tackle data sets having hundreds of thousands or even millions of highly dimensional examples. De-

spite the recent advances, there is a fundamental limitation of SVMs with nonlinear kernels that prevents applying them on truly large data sets or data streams. This is a well documented and understood property that on the noisy or highly nonlinear data sets, the size of SVM model, measured in the number of support vectors, grows linearly with number of training examples (Steinwart, 2003b). This limitation could explain the recent increased interest in linear SVMs whose model size remains fixed regardless of the data size. However, this comes at a price of significantly reduced representational power of linear SVMs.

As a result, there is a large scalability and representability gap between linear and kernel SVMs. On the kernel SVM side, recent empirical results show that training on data with millions of examples can take days even when using parallel processing techniques and result in a classifier consisting of hundreds of thousands of support vectors. On the linear SVM side, training on such large data takes only seconds on a regular PC and results in a single weight vector as the final classifier. The scalability and short prediction time of linear SVMs make them popular in applications such as text categorization, where examples are sufficiently high dimensional to make the classification problem nearly linearly separable. However, in many other applications, high representational power of kernel SVMs makes them significantly more accurate than linear SVMs. Filling the representability and scalability gap between linear and nonlinear classification is the challenge we intend to address in this paper.

We propose the Adaptive Multi-hyperplane Machine (AMM), a fast learning algorithm applicable to large-scale nonlinear classification problems, which provides a computationally efficient alternative to kernel SVMs. The AMM model consists of a set of weights, each assigned to one of the classes, and it predicts based on the class of weight that provides the largest prediction. The idea of using multiple weights for classification can be traced back to the multi-class SVM (Crammer and Singer, 2002), which assigns a single weight to each class and predicts the class whose

weight gives the largest prediction. By defining the cost function as the regularized margin-based training error, the resulting learning problem becomes convex and can be solved by standard tools of convex optimization. An extension of this idea that allows multiple weights per class has been originally proposed in Aiolli and Sperduti (2005). Since each weight defines a hyperplane, we refer to this approach as the Multi-hyperplane Machine (MM). There are two important properties of the MM approach. The first is that the resulting optimization problem is nonconvex, such that only convergence to a local optimum can be guaranteed. The second is that it has higher representational power than the single-hyperplane approach. It has been demonstrated experimentally that the benefits of increased representational power outweigh problems with nonconvexity and that MM could accurately solve nonlinear classification problems.

There are two issues that prevent use of MM on large-scale classification problems. The first is that it uses a Sequential Minimal Optimization (SMO) (Platt, 1998) based training algorithm, which is a batch solver that is not suitable for large-scale data. Another is that it requires a user to pre-specify the number of weights per class. Since the optimal number of weights depends on a particular classification problem, this creates a need to select this hyperparameter using an expensive cross-validation procedure. The proposed Adaptive MM approach addresses both of these issues, which makes it practically applicable to very large-scale nonlinear classification problems, as will be illustrated in the experiments.

AMM achieves its computational efficiency through the Stochastic Gradient Descent (SGD), a popular optimization technique that has recently attracted much interest as an alternative to the traditional batch-mode convex optimization for training of SVMs on large-scale data (e.g., Shalev-Shwartz et al., 2007b). SGD works by sampling labeled examples one at a time and updating the model through (sub-)gradient descent over the instantaneous objective function. Its sequential access to

Table 4.1: Summary of notation

$\mathbf{x}$	Instance
y	Label
t, n	Indexes for $\mathbf{x}$
$\mathbf{W}$	Weight matrix
$\mathbf{w}_{i,j}$	j-th weight vector of i-th class (column vector)
i, j, k, z	Indexes for $\mathbf{w}$
$A^{(t)}$	A at the t-th iteration
$P(\mathbf{W})$	Objective function parameterized by $\mathbf{W}$
$P(\mathbf{W}\|\mathbf{z})$	Function of $\mathbf{W}$ given the fixed $\mathbf{z}$
$P^{(t)}(\mathbf{W})$	Instantaneous version of $P(\mathbf{W})$ upon $(\mathbf{x}_t, y_t)$
$\mathbf{W}^*$	The optimal solution of min$P(\mathbf{W}$—$\mathbf{z})$

data makes it very suitable for large-scale or online learning. We will show analytically that our implementation of SGD allows AMM to converge to a local optimum, a property shared by the MM.

AMM addresses the problem of selection of number of weights by an adaptive procedure that allows SGD algorithm to automatically select an appropriate number of weights. Specifically, the SGD algorithm starts with a single zero weight assigned to each class and adds new weights as necessary. On simple problems such as linear or near-linear classification, the number of weights remains low, while it can become relatively high on more complex problems. Since the generalization error of AMM grows with the number of weights, as shown by an extension of the generalization theorem from Aiolli and Sperduti (2005), AMM also contains a pruning mechanism that removes small weights, such that it provably results in only a minor degradation in optimization accuracy. In addition to improving generalization error, pruning is also useful because it reduces the cost of training and prediction.

With efficient training and prediction, comparable to linear SVM, and the ability to solve nonlinear classification problems, similar to kernel SVM, AMM fills the scalability and representability gap between linear and nonlinear classification. As such, AMM can be an appealing option when solving large-scale nonlinear classification problems.

4.2 Preliminaries

In this paper we focus on multi-class problems. To facilitate reading, the main notation used in the paper is summarized in Table 1. Let us assume we are given a set of training examples $S = \{(\mathbf{x}_n, y_n), n = 1, ..., N\}$, where instance $\mathbf{x}_n \in \mathbb{R}^D$ is a D-dimensional feature vector and $y_n \in \mathcal{Y} = \{1, ..., M\}$ is the multi-class label. The goal is to learn a function $f : \mathbb{R}^D \to \mathcal{Y}$ that accurately predicts the label of new instances. Next, we will give an overview of multi-class SVMs proposed in Crammer and Singer (2002) and their extension, the Multi-hyperplane Machine (MM) proposed in Aiolli and Sperduti (2005). This overview will provide the necessary background needed for description of the proposed AMM algorithm, given in Section 3.

4.2.1 Multi-Class SVM

In multi-class SVM (Crammer and Singer, 2002), the model $f(\mathbf{x})$ is of the form

$$f(\mathbf{x}) = \text{argmax}_{i \in \mathcal{Y}} g(i, \mathbf{x}), \tag{4.1}$$

where

$$g(i, \mathbf{x}) = \mathbf{w}_i^T \mathbf{x} \tag{4.2}$$

is parameterized by the weight vector $\mathbf{w}_i \in \mathbb{R}^D$ of the i-th class. Thus, the predicted label of $\mathbf{x}$ is the class of the weight vector that achieves the maximum value $g(i, \mathbf{x})$. By concatenating all the class-specific weight vectors, we can construct

$$\mathbf{W} = [\mathbf{w}_1 \ \mathbf{w}_2 \ ... \ \mathbf{w}_M]$$

as the $D \times M$ weight matrix representing $f(\mathbf{x})$. Under this setup, the multi-class SVM problem was defined in Crammer and Singer (2002) as

$$\min_{\mathbf{W}} P(\mathbf{W}) \equiv \frac{\lambda}{2}||\mathbf{W}||^2 + \frac{1}{N}\sum_{n=1}^{N} l(\mathbf{W}; (\mathbf{x}_n, y_n)), \tag{4.3}$$

where $\lambda > 0$ is a regularization parameter that trades off the model complexity defined as the Frobenius norm on $\mathbf{W}$,

$$||\mathbf{W}||^2 = \sum_{i \in \mathcal{Y}} ||\mathbf{w}_i||^2,$$

and the margin-based training loss

$$l(\mathbf{W}; (\mathbf{x}_n, y_n)) = \max \left(0, 1 + \max_{i \in \mathcal{Y} \setminus y_n} g(i, \mathbf{x}_n) - g(y_n, \mathbf{x}_n) \right). \tag{4.4}$$

It can be seen from (4.4) that the training loss is zero if the prediction from the correct class is larger by at least one than the maximal prediction from the incorrect classes; otherwise, linear penalty is charged.

4.2.2 Multi-Hyperplane Machine

In order to increase the expressiveness of the classifier, Aiolli and Sperduti (2005) extended the multi-class SVM by allowing multiple weights per class. Let us denote $\mathbf{w}_{i,j}$ as the j-th weight of the i-th class and let us assume that the i-th class has a total of b_i weights. By redefining $g(i, \mathbf{x})$ from (4.2) as

$$g(i, \mathbf{x}) = \max_j \mathbf{w}_{i,j}^T \mathbf{x}, \tag{4.5}$$

the classifier $f(\mathbf{x})$ from (4.1) returns the associated class of the weight with the maximal prediction. With this modification to multi-class SVM, the training loss (4.4) equals zero if the maximal prediction from the weights of the correct class is by at least one larger than any prediction from weights of the incorrect classes.

We can now concatenate all weights and define

$$\mathbf{W} = \left[\begin{array}{c|c|c|c} \mathbf{w}_{1,1} ... \mathbf{w}_{1,b_1} & \mathbf{w}_{2,1} ... \mathbf{w}_{2,b_2} & ... & \mathbf{w}_{M,1} ... \mathbf{w}_{M,b_M} \end{array} \right] \tag{4.6}$$

where $b_1, ..., b_M$ are the numbers of weights assigned to each of the classes and each block in (4.6) is a set of class-specific weights. Given this setup, $\mathbf{W}$ is learned by

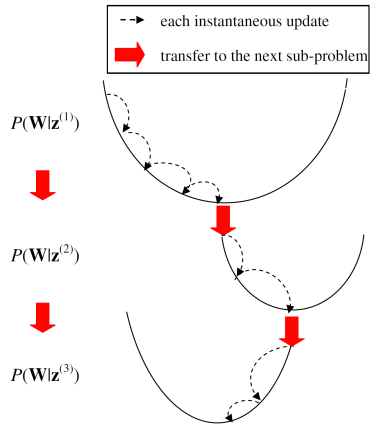


FIGURE 4.1: Convergence of MM training

solving the optimization problem (4.3) where $g(i, \mathbf{x})$ is defined as in (4.5) and the model complexity is calculated as the Frobenius norm on $\mathbf{W}$ from (4.6). We call this algorithm the Multi-hyperplane Machine (MM).[1]

We should note that although the MM algorithm allows using different number of weights (i.e. b_i) for different classes, due to practical difficulties in determining these numbers, in Aiolli and Sperduti (2005) all classes are assigned the same number of weights b. This can be suboptimal since, depending on their distribution, different classes might require different numbers of weights.

4.2.3 MM Training

Let us consider finding the optimal MM weight matrix (4.6) by minimizing (4.3). As explained in Aiolli and Sperduti (2005), the cost function $P(\mathbf{W})$ is nonconvex and finding the global optimum cannot be guaranteed. To find a local optimum, an iterative procedure was proposed that solves a series of convex upper bounds of $P(\mathbf{W})$. The convex-approximated problem is defined as

$$\min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}) \equiv \frac{\lambda}{2}||\mathbf{W}||^2 + \frac{1}{N}\sum_{n=1}^{N} l_{cvx}(\mathbf{W}; (\mathbf{x}_n, y_n); z_n), \tag{4.7}$$

[1] In Aiolli and Sperduti (2005) this algorithm was called the Multi-Prototype SVM.

where the non-convex loss function l in (4.3) is replaced by its convex upper bound

$$\begin{array}{l} l_{cvx}(\mathbf{W};(\mathbf{x}_n,y_n);z_n) \\ = \max\left(0, 1+\max_{i\in\mathcal{Y}\setminus y_n} g(i,\mathbf{x}_n) - \mathbf{w}_{y_n,z_n}^T\mathbf{x}_n\right) \end{array}, \tag{4.8}$$

that replaces the concave term $-g(y_n,\mathbf{x}_n)$ in (4.4) with the convex term $-\mathbf{w}_{y_n,z_n}^T\mathbf{x}_n$. Element z_n of vector $\mathbf{z} = [z_1...z_N]$ determines which weight from the correct class of n-th example is used to calculate (4.8). Values $\mathbf{z}$ are fixed during optimization of (4.7).

The resulting MM algorithm can be described in the following way. At step $r = 1$, initialize $\mathbf{z}^{(1)}$, typically by random weight assignment. Then, solve the convex optimization problem (4.7) to find model weights as

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}^{(r)}).$$

Then, at step $(r+1)$, recalculate assignments $\mathbf{z}$ such that the objective function is minimized,

$$\mathbf{z}^{(r+1)} = \arg\min_{\mathbf{z}} P(\mathbf{W}^*|\mathbf{z}). \tag{4.9}$$

This can be achieved by calculating $z_n^{(r+1)}$ as

$$z_n^{(r+1)} = \arg\max_k (\mathbf{w}_{y_n,k}^*)^T\mathbf{x}_n, \tag{4.10}$$

which is the correct class weight that gives the highest prediction on n-th example. The procedure of optimizing $\mathbf{W}$ and reassigning $\mathbf{z}$ is then repeated until convergence to a local optimum. The convergence is guaranteed because each step of the algorithm leads to a decrease of the objective function $P(\mathbf{W}|\mathbf{z})$. Convergence of MM is illustrated in Figure 1, where it can be seen that

$$\min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}^{(r)}) \geqslant \min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}^{(r+1)}).$$

4.3 Adaptive Multi-Hyperplane Machine (AMM)

In this section we describe the AMM algorithm. We start by describing AMM training using Stochastic Gradient Descent (SGD). Then, we explain that SGD allows us to easily address the problem of selecting the number of class-specific weights. After pointing out that the generalization error increases with the number of weights, we propose a pruning strategy that allows reducing the generalization error while having a provably small negative impact on the optimization. Finally, we propose a simplified version of AMM that is suitable for online learning.

4.3.1 Solving the Sub-Problem (4.7)

We propose an SGD algorithm to solve the convex optimization problem (4.7). The SGD is initialized with the zero weight matrix $\mathbf{W}^{(1)} = \mathbf{0}$ and it reads examples one by one and modifies the weight matrix accordingly. Let us for now assume that there are b_i weights for each class. Upon receiving example $(\mathbf{x}_t, y_t) \in S$ at t-th round, $\mathbf{W}^{(t)}$ is updated using the sub-gradient of the instantaneous objective on t-th example defined as

$$P^{(t)}(\mathbf{W}|\mathbf{z}) \equiv \frac{\lambda}{2}||\mathbf{W}||^2 + l_{cvx}(\mathbf{W}; (\mathbf{x}_t, y_t); z_t). \tag{4.11}$$

As seen, the instantaneous objective differs from (4.7) in that it only calculates the convex loss on t-th example. The model weights are updated in the negative direction of the sub-gradient as

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta^{(t)} \nabla^{(t)}, \tag{4.12}$$

where $\eta^{(t)} = 1/(\lambda t)$ is the learning rate and the sub-gradient matrix $\nabla^{(t)}$ is defined as

$$\nabla^{(t)} = \left[\begin{array}{c|c|c|c} \nabla^{(t)}_{1,1} ... \nabla^{(t)}_{1,b_1} & \nabla^{(t)}_{2,1} ... \nabla^{(t)}_{2,b_2} & ... & \nabla^{(t)}_{M,1} ... \nabla^{(t)}_{M,b_M} \end{array} \right] \tag{4.13}$$

where $\nabla^{(t)}_{i,j} = \nabla_{\mathbf{w}^{(t)}_{i,j}} P^{(t)}(\mathbf{W}^{(t)}|\mathbf{z})$ is a column vector. If convex loss $l_{cvx}(\mathbf{W}; (\mathbf{x}_t, y_t); \mathbf{z}_t)$

Algorithm 4 Training Algorithm for AMM

Input: Training set S, the regularization parameter λ
Output: $\mathbf{W}^{(t)}$
Initialize: $\mathbf{W}^{(1)} = \mathbf{0}$, $t = 1, r = 1$;
1: initialize $\mathbf{z}^{(1)}$; /* *Build 1-st sub-problem* $P(\boldsymbol{W}|\boldsymbol{z}^{(1)})$ */
2: **repeat**
3: /* *Solve each sub-problem* $P(\boldsymbol{W}|\boldsymbol{z}^{(r)})$*: lines* $4 \sim 7$*/
4: **repeat**
5: $(\mathbf{x}_t, y_t) \leftarrow t$-th example from S;
6: compute $\mathbf{W}^{(++t)}$ using (4.12);
7: **until** (enough epochs)
8: compute $\mathbf{z}^{(++r)}$ using (4.9); /* *Reassign* $\boldsymbol{z}$ */
9: **until** ($\mathbf{z}^{(r+1)} == \mathbf{z}^{(r)}$ or enough epochs)

equals zero, then $\nabla_{i,j}^{(t)} = \lambda \mathbf{w}_{i,j}^{(t)}$; otherwise,

$$\nabla_{i,j}^{(t)} = \begin{cases} \lambda \mathbf{w}_{i,j}^{(t)} + \mathbf{x}_t, & \text{if} i = i_t,\ j = j_t \\ \lambda \mathbf{w}_{i,j}^{(t)} - \mathbf{x}_t, & \text{if} i = y_t,\ j = z_t \\ \lambda \mathbf{w}_{i,j}^{(t)}, & \text{otherwise,} \end{cases} \tag{4.14}$$

where

$$i_t = \arg\max_{k \in \mathcal{Y} \setminus y_t} g(k, \mathbf{x}) \quad \text{and} \quad j_t = \arg\max_{k} (\mathbf{w}_{i_t,k}^{(t)})^T \mathbf{x}_t.$$

The update rule (4.12) can be summarized as follows. At every round, all model weights are reduced towards zero. In addition, if the convex loss on the t-th example is positive, then the class weight from the true class indexed by z_t, $\mathbf{w}_{y_t,z_t}^{(t)}$, is moved towards $\mathbf{x}_t$, while the class weight with the maximum prediction from the remaining classes $\mathbf{w}_{i_t,j_t}^{(t)}$ is moved away from $\mathbf{x}_t$. The SGD updates are summarized in Algorithm 1 (Steps 4-7).

The convergence to the global optimum of the convex sub-problem (4.7) by SGD can be shown by the following theorem. Without the loss of generality, let us assume $||\mathbf{x}|| \leqslant 1$.

Theorem 1. *Run the SGD update rule (4.12) to solve the optimization problem*

(4.7). Let $\boldsymbol{W}^*$ *be the optimal solution of (4.7). Then we have*

$$\frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{(t)}|\mathbf{z}) - \frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{*}|\mathbf{z}) \leqslant \frac{8(\ln(T)+1)}{\lambda T}.$$

The proof of Theorem 1 is given later together with the proof of Theorem 3. The theorem tells that when T is large the averaged instantaneous loss of the algorithm converges towards that of the optimal solution. The following corollary can be obtained by following the proof of Theorems 2 and 3 from Shalev-Shwartz et al. (2007b).

Corollary 1. *Assume that the conditions stated in Theorem 1 hold and for all* t $(\boldsymbol{x}_t, y_t)$ *is i.i.d. sampled from* S*. Let* $\delta \in (0,1)$*. Then, with probability of at least* $1-\delta$*, we have*

$$P(\mathbf{W}^{(T)}|\mathbf{z}) \leqslant P(\mathbf{W}^{*}|\mathbf{z}) + \frac{8(\ln(T)+1)}{\delta\lambda T}.$$

The corollary tells that when T is large the weight matrix $\mathbf{W}^{(t)}$ converges to the optimal solution in the limit and that the convergence rate is inversely proportional to the confidence parameter δ.

4.3.2 Number of Weights

As mentioned in Section 2.2, a drawback of MM is the need to pre-specify the number of model weights. In our AMM algorithm that uses SGD, we address this issue by simply setting the number of weights per class to infinity. Let us discuss the consequences of this idea.

First, we should observe that SGD initializes all model weights to zero. At t-th round, zero weight $\mathbf{w}_{i,j}$ becomes non-zero only if the convex loss $l_{cvx}(\mathbf{W};(\mathbf{x}_t, y_t); z_t)$ is positive and either (i) $i = i_t$ and $j = j_t$ (the largest prediction from nonzero weights of incorrect classes is less than zero) or (ii) $i = y_t$ and $j = z_t$ (the assigned weight

from true class is zero). Therefore, at most two zero weights can become nonzero at each round. In practice, as the AMM classifier improves during training, it will become less likely that any zero weight satisfies either condition (i) or (ii). On more complex and noisy data sets, we can expect to have more nonzero weights than on the simpler classification problems.

Second, let us discuss the implementation issue of storing an infinite number of model weights. We address it by storing all nonzero weights and a single reserved zero weight per class. Any time a zero weight becomes nonzero, we create a space for the newly created nonzero weight. In this way, a compact storage is achieved that is equivalent to storing an infinite number of weights.

4.3.3 Generalization Error

We now discuss the generalization error of the learned AMM model. Let us denote by b_i the number of AMM weights for i-th class, which is the sum of all its nonzero weights plus one for the reserved zero weight. In Aiolli and Sperduti (2005), the authors derived a generalization error of MM model under the assumption that training data can be separated by the model. Their proof follows the framework of Decision Directed Acyclic Graph (DDAG) (Platt et al., 2000) and the techniques used in Crammer and Singer (2002). The proof was based on the assumption that each class has equal number of weights. Here, we provide an extension of the proof in case when the number of weights per class varies.

By considering the classification process of AMM model as a Decision Directed Acyclic Graph (DDAG) with $\frac{1}{2}\sum_{i=1}^{M} b_i \sum_{j\neq i}^{M} b_j$ nodes, and following the proof of Lemma 3 in Aiolli and Sperduti (2005), we can obtain a margin-based bound. By upper bounding the margin-relevant terms by the norm of model weights, we have the following error bound.

Theorem 2. *Suppose we are able to correctly classify an i.i.d sampled training set S using the AMM model*

$$\mathbf{W} = [\mathbf{w}_{1,1}...\mathbf{w}_{1,b_1}\mathbf{w}_{2,1}...\mathbf{w}_{2,b_2}...\mathbf{w}_{M,1}...\mathbf{w}_{M,b_M}],$$

then we can upper bound the generalization error with probability greater than $1-\delta$ as

$$\frac{130}{N}\left(||\mathbf{W}||^2 B\log(4eN)\log(4N) + \log(\frac{2(2N)^K}{\delta})\right),$$

where $B = \sum_{i=1}^{M} b_i + 1 + b_{\max}^2 - b_{\max} - b_{\min}$, $K = \frac{1}{2}\sum_{i=1}^{M} b_i \sum_{j\neq i}^{M} b_j$, $b_{\min} = \min_{i=1,...,M}\{b_i\}$ *and* $b_{\max} = \max_{i=1,...,M}\{b_i\}$.

Theorem 2 shows that the generalization error is proportional to $||\mathbf{W}||^2 B$ and that it can be reduced either by reducing the model weight norm or the number of weights. This clearly suggests that AMM should attempt to use as few nonzero weights as possible. This conclusion is a motivation for weight pruning strategy proposed next.

4.3.4 Weight Pruning

Every round of SGD training results in shrinking AMM model weights towards zero. As a result, the class weights that are rarely updated can become very small and start resembling zero weights. Such weights are typically generated during initial stages of training when AMM model is less accurate or during model updates caused by noisy examples. Since, by Theorem 2, nonzero weights negatively impact the generalization error, and since replacing small weights with zero weights is not likely to significantly influence AMM accuracy, we propose a pruning step that occasionally removes weights with sufficiently small norms. In addition to reducing the generalization error, weight removal is beneficial from computational viewpoint because it can speed-up both training and prediction and reduce model size.

The pruning step can be formulated as

$$\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t+1)} - \Delta\mathbf{W}^{(t)}, \tag{4.15}$$

where $\Delta\mathbf{W}^{(t)}$ is a sparse matrix of the same size as $\mathbf{W}^{(t+1)}$ whose nonzero columns correspond to removed weights. For example, if $\mathbf{w}_{i,j}^{(t)}$ is removed, $\Delta\mathbf{w}_{i,j}^{(t)} = \mathbf{w}_{i,j}^{(t)}$, and all other columns of $\Delta\mathbf{W}^{(t)}$ are zero. In AMM, pruning is performed periodically after every k rounds and only on the weights that are below a threshold. The following theorem analyzes the impact of pruning on the convergence of SGD.

Theorem 3. *Consider the SGD update rule (4.12). Let $\boldsymbol{W}^*$ be the optimal solution of the problem (4.7) . Define a pruning constant $c \geqslant 0$ and execute the pruning step (4.15) after each k iterations with $\Delta\mathbf{W}^{(t)} \leqslant c/((t-1)\lambda)$. Then we have*

$$\frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{(t)}|\mathbf{z}) - \frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{*}|\mathbf{z}) \leqslant \frac{(8+c)(\ln(T)+1)}{\lambda T} + \frac{2(4+c)c}{k\lambda}.$$

Unified Proof of Theorems 1 & 3. First, we rewrite the update rule of SGD with the optional pruning step as $\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t)} - \eta^{(t)}\partial^{(t)}$, where $\partial^{(t)} = \nabla^{(t)} + E^{(t)}$ and $E^{(t)} = \mathbf{0}$ if no pruning is used. The relative progress towards the optimal solution $\mathbf{W}^*$ at t-th round $D^{(t)}$ can be lower bounded as

$$\begin{aligned}
&D^{(t)} = ||\mathbf{W}^{(t)} - \mathbf{W}^*||^2 - ||\mathbf{W}^{(t)} - \eta^{(t)}\nabla^{(t)} - \eta^{(t)}E^{(t)} - \mathbf{W}^*||^2 \\
&= -(\eta^{(t)})^2||\partial^{(t)}||^2 + 2\eta^{(t)}(E^{(t)})^T(\mathbf{W}^{(t)} - \mathbf{W}^*) \\
&+2\eta^{(t)}(\nabla^{(t)})^T(\mathbf{W}^{(t)} - \mathbf{W}^*) \\
&\geqslant_1 -(\eta^{(t)})^2G^2 - 2\eta^{(t)}||E^{(t)}||\tfrac{4+c}{\lambda} \\
&+2\eta^{(t)}\left(P^{(t)}(\mathbf{W}^{(t)}) - P^{(t)}(\mathbf{W}^*) + \tfrac{\lambda}{2}||\mathbf{W}^{(t)} - \mathbf{W}^*||^2\right).
\end{aligned} \tag{4.16}$$

In $\geqslant_1$, we assume there is a constant $G \geqslant 0$ such that $||\partial^{(t)}|| \leqslant G$ and we will quantify G later on. For the second term, we first bound $\mathbf{W}^{(t)}$ as

$$\begin{aligned}
&||\mathbf{W}^{(t)}|| \leqslant ||(1-\eta^{(t-1)}\lambda)\mathbf{W}^{(t-1)}|| + 2\eta^{(t-1)} + ||\Delta\mathbf{W}^{(t-1)}|| \\
&\leqslant \tfrac{t-2}{t-1}||\mathbf{W}^{(t-1)}|| + \tfrac{2}{(t-1)\lambda} + \tfrac{c}{(t-1)\lambda} \\
&\leqslant \tfrac{1}{t-1}||\mathbf{W}^{(0)}|| + \tfrac{2(t-1)}{(t-1)\lambda} + \tfrac{(t-1)c}{(t-1)\lambda} = \tfrac{2+c}{\lambda},
\end{aligned}$$

and then use triangle inequality to bound $||\mathbf{W}^{(t)} - \mathbf{W}^*|| \leqslant (2+c)/\lambda + 2/\lambda$ by using the fact $||\mathbf{W}^*|| \leqslant 2/\lambda$ according to the result in Kivinen et al. (2002). Then, we bound the third term using function $P^{(t)}(\mathbf{W}^{(t)})$'s λ-strong convexity (Shalev-Shwartz and Singer, 2007).

Dividing both sides of inequality (4.16) by $2\eta^{(t)}$ and rearranging, we obtain

$$\begin{aligned} &P^{(t)}(\mathbf{W}^{(t)}) - P^{(t)}(\mathbf{W}^*) \leqslant \tfrac{D^{(t)}}{2\eta^{(t)}} - \tfrac{\lambda}{2}||\mathbf{W}^{(t)} - \mathbf{W}^*||^2 \\ &+ \tfrac{\eta^{(t)}G^2}{2} + \tfrac{(4+c)||E^{(t)}||}{\lambda}. \end{aligned} \tag{4.17}$$

Summing over all t, we get

$$\begin{aligned} &\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{(t)}) - \sum_{t=1}^{T} P^{(t)}(\mathbf{W}^*) \leqslant \sum_{t=1}^{T} \tfrac{D^{(t)}}{2\eta^{(t)}} \\ &- \sum_{t=1}^{T} \tfrac{\lambda}{2}||\mathbf{W}^{(t)} - \mathbf{W}^*||^2 + \tfrac{G^2}{2}\sum_{t=1}^{T}\eta^{(t)} + \tfrac{(4+c)}{\lambda}\sum_{t=1}^{T}||E^{(t)}||. \end{aligned}$$

We bound the first and second terms in inequality (4.17) as

$$\begin{aligned} &\tfrac{1}{2}\sum_{t=1}^{N}\left(\tfrac{D^{(t)}}{\eta^{(t)}} - \lambda||\mathbf{W}^{(t)} - \mathbf{W}^*||^2\right) = \tfrac{1}{2}\Big((\tfrac{1}{\eta^{(1)}} - \lambda)||\mathbf{W}^{(1)} \\ &-\mathbf{W}^*||^2 + \sum_{t=2}^{N}\left(\tfrac{1}{\eta^{(t)}} - \tfrac{1}{\eta^{(t-1)}} - \lambda\right)||\mathbf{W}^{(t)} - \mathbf{W}^*||^2 \\ &-\tfrac{1}{\eta^{(N)}}||\mathbf{W}^{(T+1)} - \mathbf{W}^*||^2\Big) \\ &=_1 -\tfrac{1}{2\eta^{(N)}}||\mathbf{W}^{(T+1)} - \mathbf{W}^*||^2 \leqslant 0. \end{aligned} \tag{4.18}$$

In $=_1$, the first and second terms vanish after plugging $\eta_t \equiv 1/(\lambda t)$. Next, we bound the third term in inequality (4.17) according to the divergence rate of harmonic series,

$$\frac{G^2}{2}\sum_{t=1}^{T}\eta^{(t)} = \frac{G^2}{2\lambda}\sum_{t=1}^{T}\frac{1}{t} \leqslant \frac{G^2}{2\lambda}(\ln(T)+1). \tag{4.19}$$

Then, we quantify G as

$$||\partial^{(t)}|| \leqslant ||\nabla^{(t)}|| + ||E^{(t)}|| \leqslant (\lambda||\mathbf{W}^{(t)}|| + 2) + 2 + c \leqslant 6 + c \equiv G.$$

If no pruning is used, $||E^{(t)}|| = 0$. For the iterations when the pruning is executed, we bound $||E^{(t)}||$ as $||E^{(t)}|| \leqslant 2c$ using the bound on $\Delta\mathbf{W}^{(t)}$. Combining inequality (4.18) with (4.19) and dividing two sides of inequality by T, we get the stated bounds as in Theorems 1 and 3. □

Theorem 3 quantifies the upper bound on the difference between the averaged instantaneous loss of the AMM with pruning and the optimal solution. It can be seen that the gap is proportional to the pruning threshold c and inversely proportional to the pruning interval k. When $c = 0$, Theorem 3 is equivalent to Theorem 1. Large c and small k correspond to a more aggressive pruning that enforces a simpler classifier but with a wider gap with the optimal solution; while small c and large k shrink the gap but lead to larger AMM model. Next, we state the following important property of pruning.

Proposition 1. *Let us consider the case where pruning (4.15) is executed after each k iterations with threshold c. Let us consider a weight which is being updated at least c times through the first two cases in (4.13) during the previous k iterations. It can be shown that such weight will not be pruned.*

We omit the proof due to lack of space. The Proposition 1 provides a very useful interpretation of the effect of the pruning threshold c in Theorem 3. To be consistent with Theorem 3, pruning in AMM is implemented as follows: every k rounds, AMM weights are pruned starting from the one with the smallest norm, and continued as long as the cumulative Frobenius norm of the pruned weights is below threshold $c/((t-1)\lambda)$.

4.3.5 Online AMM

AMM described in Algorithm 1 is suitable for offline application where the whole data set is residing on a computer and multiple passes through the data are allowed.

This is due to the procedure that iteratively solves an approximate convex problem (4.7) and recalculates assignments $\mathbf{z}$ on the whole data set. Here, we propose a modification of AMM that allows its use in an online setting, where the data set is observed sequentially in a single pass. Online AMM calculates assignment of the incoming t-th example as

$$z_t = \arg\max_k((\mathbf{w}_{y_t,k}^{(t)})^T \mathbf{x}_t)$$

and updates $\mathbf{W}^{(t)}$ to $\mathbf{W}^{(t+1)}$ using SGD (4.12) with this assignment. The resulting online training procedure is summarized in Algorithm 2.

The main difference between the offline (Algorthm 1) and online (Algorithm 2) versions of AMM is that the online version does not wait for convergence of (4.7) but changes the assignment $\mathbf{z}$ after every update of SGD. As a result, instead of having to periodically calculate assignment $\mathbf{z}$ on the whole data set, it only has to be done on the single incoming example. This allows a single-pass AMM training, and it also leads to savings in computational speed and memory. The price to be paid is that convergence of online AMM to a local optimum cannot be guaranteed, because the online AMM greedily minimizes the non-convex instantaneous objective (4.3). As will be seen from the experimental results, the online AMM behavior is very similar to the offline AMM. Coupled with computational efficiency, ease of implementation, and applicability for stream learning, this makes online AMM a highly attractive learning algorithm.

4.3.6 Implementation Details

A naïve implementation of the update rule (4.12) would require $\mathcal{O}(D)$time for weight shrinking $(1 - \eta^{(t)}\lambda)\mathbf{w}_{i,j}^{(t)}$ and $\mathcal{O}(D)$ time to perform weight update. This computational burden is unnecessary when instances are highly dimensional and sparse (e.g. in text/image data). To circumvent this problem we apply an approach used to

Algorithm 5 Online Algorithm

Input: Training set S, the regularization parameter λ
Output: $\mathbf{W}^{(t)}$
Initialize: $\mathbf{W}^{(1)} = \mathbf{0}$, $t = 1$
1: **repeat**
2: $(\mathbf{x}_t, y_t) \leftarrow t$-th example from S;
3: calculate $\mathbf{z}_t$ by (4.10);
4: update $\mathbf{W}^{(++t)}$ by (11);
5: **until** (data set exhausted)

Table 4.2: Summary of large datasets

Datasets	#train	#test	#dim	#class	non-zero %	file size	domain
a9a	32,561	16,281	123	2	11.3%	2M	social survey
ijcnn	49,990	91,701	22	2	66.7%	7.5M	time series
webspam	280,000	70,000	254	2	41.9%	327M	web text
rcv1_bin	677,399	20,242	47,236	2	0.2%	35M	news text
url	1,976,130	420,000	3,231,961	2	0.004%	1.7G	Internet data
mnist8m_bin	8,000,000	10,000	784	2	19.3%	18G	OCR images
mnist8m_mc	8,000,000	10,000	784	10	19.3%	18G	OCR images

speed-up linear SVM training (Shalev-Shwartz et al., 2007b) that represents $\mathbf{w}_{i,j}$ as $\mathbf{w}_{i,j} = s_{i,j}\mathbf{v}_{i,j}$, where $s_{i,j}$ is a scalar and $\mathbf{v}_{i,j} \in \mathbb{R}^D$. In this case, the update rule can be decomposed into $s_{i,j}^{(t)} \leftarrow (1 - \eta^{(t)}\lambda)s_{i,j}^{(t)}$ and $\mathbf{v}_{i,j}^{(t)} \leftarrow \mathbf{v}_{i,j}^{(t)} \pm \eta^{(t)}/s_{i,j}^{(t)}\mathbf{x}_t$ in time only proportional to the number of non-zero features in $\mathbf{x}_t$.

4.4 Experiments

In this section, we evaluate the proposed batch and online AMM algorithms and compare them with the large-scale Kernel and Linear SVM algorithms on several large real-life data sets. The competing algorithms are listed as follows:

- **LibSVM** (Chang and Lin, 2011) A popular SMO-based SVM solver which is scalable to hundreds of thousands examples.
- **LaSVM** (Bordes et al., 2005) A large-scale online SVM algorithm that accesses

Table 4.3: Error rate and training time comparison with large-scale algorithms (RBF SVM is solved by LibSVM unless specified otherwise. Poly2 and LibSVM results are from Chang et al. (2010)).

Datasets	Error rate (%)					Training time (seconds)[1]				
	AMM *batch*	AMM *online*	Linear (Pegasos)	Poly2 SVM	RBF SVM	AMM *batch*	AMM *online*	Linear (Pegasos)	Poly2 SVM	RBF SVM
a9a	15.03±0.11	16.44±0.23	15.04±0.07	14.94	14.97	2	0.2	1	2	99
ijcnn	2.40±0.11	3.02±0.14	7.76±0.19	2.16	1.31	2	0.1	1	11	27
webspam	4.50±0.24	6.14±1.08	7.28±0.09	1.56	0.80	80	4	12	3,228	15,571
mnist_bin	0.53±0.05	0.54±0.03	2.03±0.04	NA	0.43[2]	3084	300	277	NA	2 days[2]
mnist_mc	3.20±0.16	3.36±0.20	8.41±0.11	NA	0.67[3]	13864	1200	1180	NA	8 days[3]
rcv1_bin	2.20±0.01	2.21±0.02	2.29±0.01	NA	NA	1100	80	25	NA	NA
url	1.34±0.21	2.87±1.49	1.50±0.39	NA	NA	400	24	100	NA	NA

[1] excludes data loading time.
[2] achieved by parallel training P-packSVMs on 512 processors; results from Zhu et al. (2009).
[3] achieved by LaSVM; results from Loosli et al. (2007).

Table 4.4: The number of weights in the classifiers

Datasets	a9a	ijcnn	webspam	mnist8m_bin	mnist8m_mc	rcv1	url
batch AMM	11±1	11±1	13±2	20±1	65±2	22±2	4±0
online AMM	16±2	15±1	10±1	13±1	61±3	44±5	5±1

one example at a time.

- **P-packSVM** (Zhu et al., 2009) An SVM solver that parallelizes the SGD style training on multiple processors.

- **Pegasos** (Shalev-Shwartz et al., 2007b) The state-of-the-art Linear SVM solver which is based on SGD.

- **Poly 2 SVM by Liblinear** (Chang et al., 2010) A fast solver for Polynomial degree 2 kernel SVM. The algorithm explicitly expresses the feature space as a set of attributes and then trains Liblinear on the transformed data. However, it is only scalable to very sparse or low dimensional data.

A summary of datasets[2] is shown in Table 2.

[2] url dataset (Ma et al., 2009) is available at *http://www.sysnet.ucsd.edu/projects/url/*; all the others are available at *http://www.csie.ntu.edu.tw/c̃jlin/libsvmtools/datasets/*

For Algorithm 1, instead of a random assignment, we initialized $\mathbf{z}^{(1)}$ by a single scan of data using Online AMM, which sequentially sets each element of $\mathbf{z}^{(1)}$ as in (4.10). Our preliminary results showed it worked better than the simple random initialization approach. We used the epoch-based stopping criteria[3] for SGD for each sub-problem. Since all studied data sets were large we used only 1 epoch before reassigning the weights. We stopped training after 5 passes through the data. We ran online AMM using a single pass of the data. In addition, we also explored performance of online AMM when 5 passes through data. This allowed us to compare online and batch AMM. The pruning step with $k = 10,000$ and $c = 10$ was used with Algorithms 1 & 2 in all the experiments.

The training error and training time of LibSVM, LaSVM, P-packSVM and Poly 2 SVM were taken from recently published results in Chang et al. (2010); Loosli et al. (2007); Zhu et al. (2009). For Pegasos and the proposed batch and online AMM algorithms, we selected the regularization parameter λ through cross-validation. The considered range was $\lambda = 10^{-2}, ..., 10^{-7}$. We repeated all the experiments five times, each with randomly shuffled training data. Mean and standard deviation of each set of experiments are reported. All the training examples were normalized such that all attributes were within range $[-1, +1]$. Both batch and online AMM algorithms were implemented in C++. Unless otherwise stated, all the experiments were run using a 3.0GHz Intel Xeon processor with 16G memory on Linux.

The generalization error and the training time of all the algorithms on large datasets are summarized in Table 3. Due to the large computational costs of Kernel SVM training on the largest datasets from our collection, no previous results are reported (to the best of our knowledge), so we use NA to mark these cases in the table. The original MM algorithm (Aiolli and Sperduti, 2005) is also not scalable to

[3] As suggested in Bordes et al. (2009). Shalev-Shwartz et al. (2007b) uses a predefined threshold on the objective value as the stopping criteria. However, this threshold can vary from case to case.

any of the studied large datasets, so it is not included in the comparison.

Batch vs. Online First, let us compare two versions of AMM. We can see that batch AMM achieved lower generalization error than its online sibling at the expense of significantly longer training time that occasionally was more than one order of magnitude longer. The difference in accuracy depends on the difficulty of the datasets. Considering that online AMM used only a single pass of the data, its slightly degraded generalization error is understandable.

AMM vs. Linear SVM Comparing the generalization error of AMM with Linear SVM, we can see that both AMM algorithms significantly outperformed Linear SVM on 5 out of 7 datasets. Considering training time, online AMM had comparable training time to Linear SVM and batch AMM was somewhat slower. Considering prediction time, Table 4 lists the number of weights in the AMM classifiers that dictate prediction time and memory needed to store the classification model. These numbers divided by the number of classes directly reflect the increased prediction time of AMM as compared to Linear SVM. We can see that AMM classifiers are around one order of magnitude slower than Linear SVM, which has $\mathcal{O}(MD)$ prediction time. This is very impressive result considering that AMM achieves significantly lower error rate than Linear SVM. This indicates that AMM is superior to linear SVM because it provides a better tradeoff between accuracy and speed in all but the most resource-constrained applications.

AMM vs. Kernel SVM Comparison against Poly2 SVM on three relatively small datasets (a9a, ijcnn, webspam) shows that AMM had similar error rates as Poly2 SVM, while achieving several orders of magnitude faster training. Also, the $\mathcal{O}(MD)$ prediction time of AMM is much more favorable than the $\mathcal{O}(MD^2)$ prediction time of Poly2 SVM. By comparing AMM with RBF SVM on 5 low-dimensional datasets (where RBF SVM's results were reported in the previous literature), we can see that AMM has somewhat lower $(0.1\% - 2.9\%)$ but still comparable error rate.

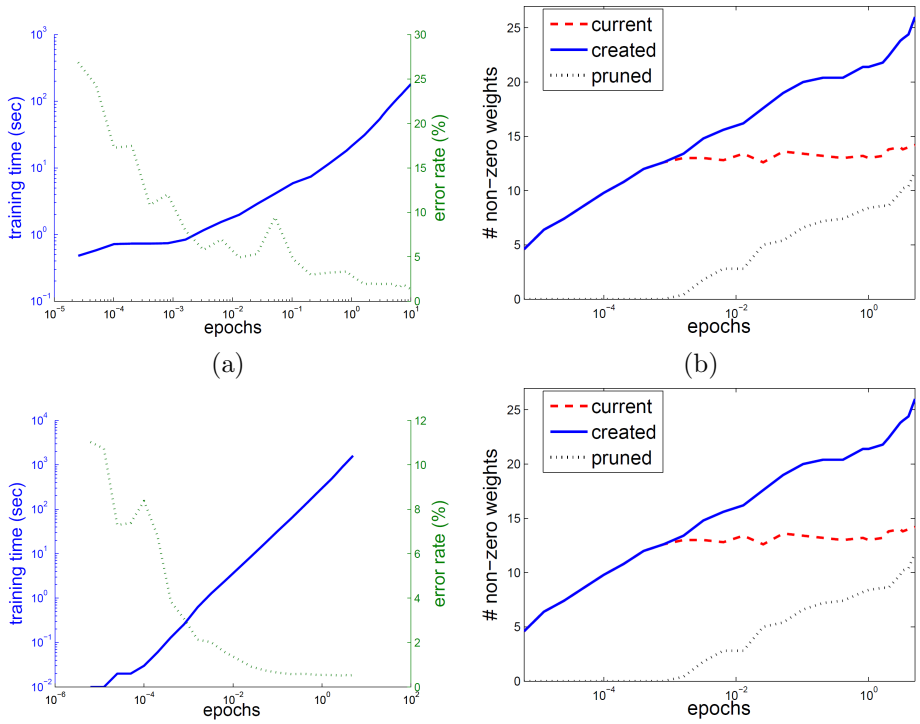


FIGURE 4.2: Detailed results on url data (upper two panels) and mnist8m_bin data (lower two panels)

Particularly, on the two largest datasets (mnist8m_bin and mnnist8m_mul), the error rate achieved by AMM after running on a single processor for a couple of hours is very competitive to P-packSVM's classifier trained in 2 days on 512 processors and LaSVM's classifier trained in 8 days on a single processor. These results show that the AMM algorithms are appealing alternative to kernel SVMs when learning from very large data.

Train Online AMM in Batch Mode Table 3 lists Online AMM results when it was run with a single pass through the data. We also performed experiments when it was allowed to make multiple passes through the data. The detailed results on the two largest datasets for the multi-pass Onine AMM are shown in Figure 2. The top left and bottom left panels in Figure 2 show the evolution of the error rate and the total training time as a function of the number of epochs (one epoch denotes a full

pass through the data). We can see that the error rate rapidly decreases during the first epoch and that it continues to improve slightly after the first epoch. The training time increases linearly, as expected. The results suggests that, if the training time is not of major concern and if data could be stored in memory, multiple accesses to the training data should be recommended. The top right and bottom right panels of Figure 2 show the number of AMM weights as training progresses. In addition to the current number of weights, the panels also show the total number of weights created and pruned at any stage of training process. It can be seen that the pruning has the desired effect of controlling the total number of model weights without negatively influencing the error rate.

4.5 Improving the representability of AMM algorithm

The empirical results from Wang et al. (2011) indicate that AMM is more accurate than linear SVM, but less accurate than non-linear SVMs. The question is whether the reduced performance is due to limited representability of the MM model, or due to limitations of MM learning algorithms. The following theorem answers this question by showing that MM can represent any concept.

Theorem 4. *Assume that we are given a training set* $\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}$, *in which there are no two examples with the same feature vector and different class labels. Then, there exists an MM classifier of the form (4.1) with* $g(i, \mathbf{x})$ *defined as in (4.5), that perfectly classifies the training set.*

Proof sketch. We give a proof by construction. Let us choose any strictly convex function $h : \mathbb{R}^D \to \mathbb{R}$. For each $\mathbf{x}_t$ find a tangent to $h(\mathbf{x})$ at $\mathbf{x}_t$, include the tangent into the weight matrix $\mathbf{W}$, and label it with y_t. Then, MM with weight matrix $\mathbf{W}$ perfectly classifies the training set, since, due to the strict convexity of $h(\cdot)$, among all other tangents the tangent at $\mathbf{x}_t$ has the maximal value for the t^{th} example and

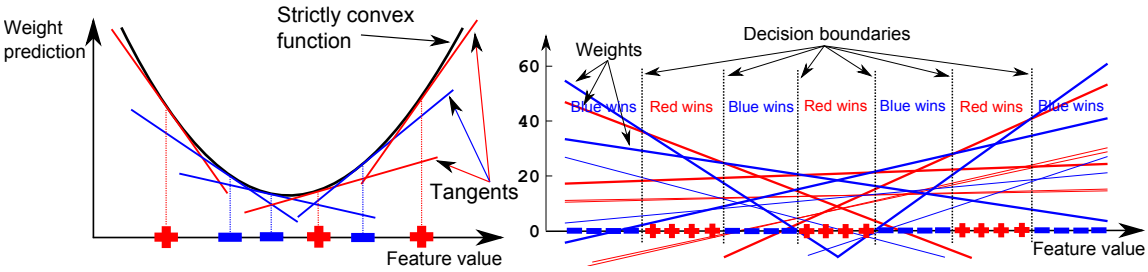


FIGURE 4.3: (a) Visualizing proof of Theorem 4; (b) GAMM on 1×7 XOR-like data

it returns the correct label y_t of the t^{th} example. □

Figure 4.3a illustrates how MM can be constructed given a training data set with 1-D examples. However, MM trained using the construction from the proof is not practical, as it would require T weights and would be prone to overfitting. Similar argument to Theorem 4 can be given to show that MM can represent an arbitrarily complex concept. For any partitioning of feature space in classes, we can split the space into a fine grid, find a tangent to a center of each grid cell, and assign the tangent a dominant class of its grid cell. The classification error could be made arbitrarily small by increasing the grid density. However, for high-dimensional feature spaces the size of the resulting MM model would be prohibitively large for practical purposes.

An interesting experimental observation is that MM models learned by the proposed GAMM algorithm resemble the constructive MM models from the proof of Theorem 4. To illustrate this, in Figure 4.3b we show the model trained on 1×7 XOR-like data by GAMM. As can be seen, the active hyperplanes indeed form a convex-shaped envelope. For completeness, we note that the resulting model also contains several inactive hyperplanes below the convex envelope.

4.5.1 Connection between MM and LVQ models

By considering Figure 4.3b and the proof of Theorem 4, we can observe that the weights of MM model define a partition of the input space into disjoint one-class

regions. This, along with prototype-based nature of MM, indicates a connection between MM models and the well-known Learning Vector Quantization (LVQ) algorithm (Kohonen, 2001) which finds a Voronoi tessellation of the input space. Interestingly, the connection between LVQ and large-margin classifiers has been established before in Crammer et al. (2002), where the authors showed that LVQ is a large-margin classifier that attempts to minimize loss function $\mathcal{L}^{(t)}_{LVQ}$ defined in terms of margin $\theta^{(t)}_{LVQ}$ for a current training example $\mathbf{x}_t$,

$$\mathcal{L}^{(t)}_{LVQ} = \max(0, 1 - \theta^{(t)}_{LVQ}), \quad \text{where } \theta^{(t)}_{LVQ} = \frac{1}{2}(\|\mathbf{x}_t - \boldsymbol{\mu}_-\|^2 - \|\mathbf{x}_t - \boldsymbol{\mu}_+\|^2), \tag{4.20}$$

and $\boldsymbol{\mu}_+$ and $\boldsymbol{\mu}_-$ are the closest LVQ prototypes to $\mathbf{x}_t$ belonging to correct and one of incorrect classes, respectively. Note that the loss function $\mathcal{L}^{(t)}_{LVQ}$ does not include regularization term, and, considering LVQ training scheme (Kohonen, 2001), it is not readily obvious how to regularize the LVQ model.

Let us introduce an alternative definition of margin as $\theta^{(t)}_{MM} = \boldsymbol{\mu}_+^{\mathrm{T}}\mathbf{x}_t - \boldsymbol{\mu}_-^{\mathrm{T}}\mathbf{x}_t$. Then, if we choose $\boldsymbol{\mu}_+$ to be a correct-class MM weight assigned to $\mathbf{x}_t$ according to $\mathbf{z}$, and $\boldsymbol{\mu}_-$ is an incorrect-class weight maximizing (4.5), we can rewrite (4.11) as

$$\mathcal{L}^{(t)}(\mathbf{W}|\mathbf{z}) = \max(0, 1 - \theta^{(t)}_{MM}) + \text{ regularization term.} \tag{4.21}$$

Clearly, the instantaneous loss functions $\mathcal{L}^{(t)}_{LVQ}$ from (4.20) and $\mathcal{L}^{(t)}(\mathbf{W}|\mathbf{z})$ from (4.21) are equivalent up to a definition of margin θ and the addition of MM regularization. However, if we expand $\theta^{(t)}_{LVQ}$,

$$\theta^{(t)}_{LVQ} = \frac{1}{2}(\|\mathbf{x}_t - \boldsymbol{\mu}_-\|^2 - \|\mathbf{x}_t - \boldsymbol{\mu}_+\|^2) = \Big(\boldsymbol{\mu}_+^{\mathrm{T}}\mathbf{x}_t - \boldsymbol{\mu}_-^{\mathrm{T}}\mathbf{x}_t\Big) + \frac{1}{2}\Big(\|\boldsymbol{\mu}_-\|^2 - \|\boldsymbol{\mu}_+\|^2\Big), \tag{4.22}$$

we can see that the LVQ margin $\theta^{(t)}_{LVQ}$ can be expressed as the sum of MM-type margin $\theta^{(t)}_{MM}$ and an additional term involving norms of the prototypes. In other

Algorithm 6 Training algorithm for GAMM

Inputs: Training set $\mathcal{D}$, regul. param. λ, duplication params p, β
Output: $\mathbf{W}^{(t)}$

1. **Initialize** $\mathbf{W}^{(0)} \leftarrow \mathbf{0}, t \leftarrow 1, r \leftarrow 0, i \leftarrow 1$, and randomly initialize $\mathbf{z}^{(0)}$
2. **repeat**
3. **repeat**
4. **if** $(l_{cvx}(\mathbf{W}^{(t)}; (\mathbf{x}_t, y_t); z_t^{(r)}) > 0)$ **then**
5. **if** $(rand() < p)$ **then**
6. duplicate weight $\mathbf{w}^{(t)}_{y_t, z_t^{(r)}}$;
7. decrease duplication probability as $p \leftarrow \beta p$;
8. compute $\mathbf{W}^{(t++)}$ using (4.12);
9. **until** (enough epochs)
10. compute $\mathbf{z}^{(++r)}$ using (4.10), and set $i \leftarrow 1$;
11. **until** $(\mathbf{z}^{(r+1)} = \mathbf{z}^{(r)}$ **or** enough epochs)

words, LVQ can be seen as an MM model without *explicit* regularization that maximizes the margin $\theta^{(t)}_{MM}$ at each training step, where the regularization is *implicit* through the additional margin term forcing the prototypes to have similar norms. Having emphasized these strong ties between MM and LVQ models, we note that there remains a difference in a way the distance measure between prototypes and example $\mathbf{x}_t$ is defined. Unlike MM that uses linear kernel, LVQ uses Euclidean distance, thus exhibiting limited performance in high-dimensional spaces due to the curse of dimensionality (Hastie et al., 1995; Verleysen et al., 2003).

4.5.2 Growing AMM (GAMM) algorithm

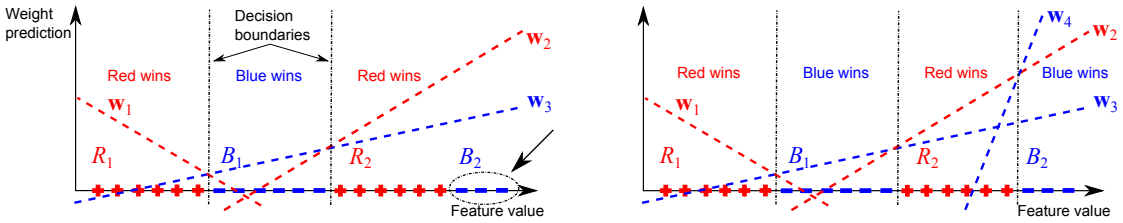
Both MM and LVQ can represent arbitrarily complex concepts, as shown in Theorem 4 and Hornik et al. (1989), respectively. However, in practice they might obtain limited results due to constraints of their respective training algorithms, as illustrated in Section 6.5. Significant efforts have been invested to address this issue for LVQ, resulting in a highly-influential work on Growing Self-Organizing Networks (Fritzke, 1994, 1995). These growing networks obtain significant performance improvements over the competing LVQ methods through insertion of prototypes into the model

that are a linear combination of existing prototypes. Considering the correspondence between LVQ and MM and inspired by the ideas from Fritzke (1994, 1995) shown to increase representability of LVQ, we present a novel MM method with significantly improved performance over the aforementioned AMM algorithm. Higher accuracy is achieved by allowing MM weights to be duplicated during the SGD training procedure. In particular, in addition to moving the true-class weight closer to the misclassified example, a duplicate of the weight is made with probability p.

The pseudocode of the proposed Growing AMM (GAMM) algorithm is shown in Algorithm 6. GAMM training closely follows the training procedure of AMM. However, unlike in the original AMM, when a convex classification loss is incurred at the t^{th} training iteration (line 6), with probability p we duplicate the true-class weight assigned to the observed example $(\mathbf{x}_t, y_t)$ (line 7). Since we expect that the weights will eventually stabilize as the training continues, and to prevent unnecessary generation of duplicate weights in later stages of training, the algorithm gradually decreases the probability p by multiplying with a positive constant $\beta < 1$ after every insertion (line 8; in the experiments we used $\beta = 0.99$). GAMM is a generalization of AMM algorithm, since by setting $p = 0$ we obtain the original AMM algorithm. Note that we can trade-off between exploration and exploitation strategies by manipulating the value of p. By using higher p we explore a larger hypothesis space, possibly leading to discovery of more powerful classifiers but also to overfitting.

Effect of weight duplication illustrated on toy example

Consider a simple two-class example shown in Figure 4.4. The two classes (with examples denoted by blue pluses and red minuses) are not linearly separable; they follow a complex XOR-like pattern, with each class consisting of two separate regions of 1-dimensional space ($\mathcal{R}_1$ and $\mathcal{R}_2$ regions for the red class, $\mathcal{B}_1$ and $\mathcal{B}_2$ regions for the blue class). Let us assume that Figure 4.4a represents AMM weights at some



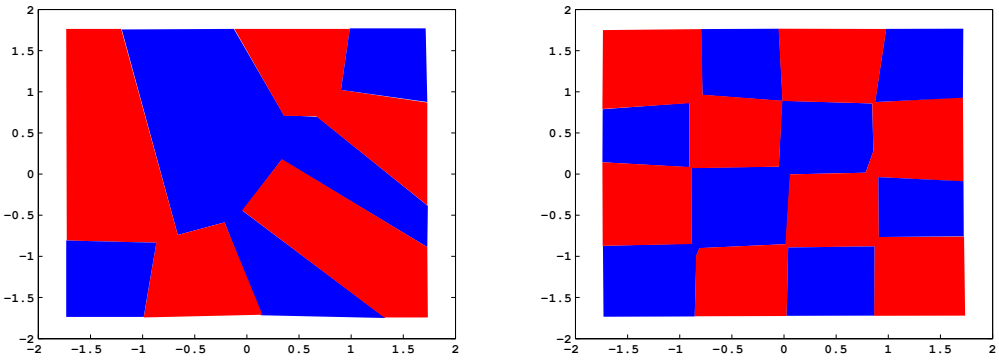
(a) Without duplication (misclassified region shown by the arrow)

(b) With duplication (succeeds to perfectly classify the data set)

FIGURE 4.4: Simple 1-dimensional XOR example where SGD training fails

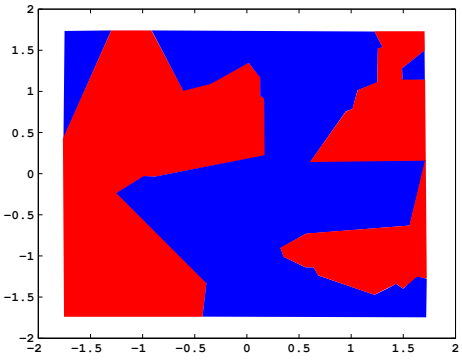
point during SGD procedure. The current MM model nearly perfectly separates the two classes (red-class weights denoted by $\mathbf{w}_1$ and $\mathbf{w}_2$, and blue-class weight by $\mathbf{w}_3$), except for $\mathcal{B}_2$ region shown by the arrow. As the SGD training continues from the point shown in Figure 4.4a, the examples from smaller $\mathcal{B}_2$ region would start moving weight $\mathbf{w}_3$ toward them, leading to misclassification of currently well-classified examples from $\mathcal{B}_1$ and $\mathcal{R}_2$ regions. In turn, the newly misclassified examples from $\mathcal{B}_1$ and $\mathcal{R}_2$ regions would move $\mathbf{w}_3$ back to its previous position shown in Figure 4.4a. The outcome would be that $\mathbf{w}_3$ goes back and forth as it attempts to correctly classify examples from both blue regions. We can see that, even though this simple example is absolutely noise-free, AMM that observes one example at a time would fail to solve the posed classification problem.

Now let us consider the case in which $\mathbf{w}_3$ would be *cloned* upon misclassification of an example from $\mathcal{B}_2$ region, thus creating a new blue-class weight $\mathbf{w}_4$. Instead of moving the well-positioned weight $\mathbf{w}_3$ responsible for correct classification of examples from $\mathcal{B}_1$ region, the new weight would be pushed towards $\mathcal{B}_2$ region, without affecting correctly classified points from $\mathcal{B}_1$ region. As $\mathbf{w}_4$ is pushed more and more towards $\mathcal{B}_2$ region, it will eventually cover the whole region, resulting in a perfect separation of blue and red classes illustrated in Figure 4.4b.

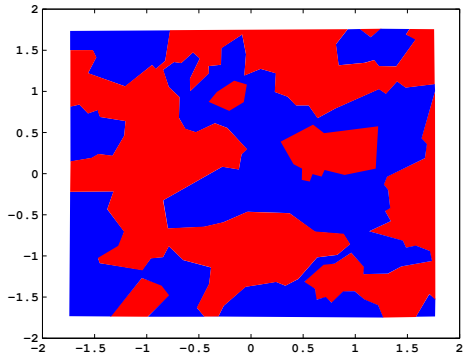


(a) AMM (27.95% err. r.)

(b) GAMM (7.38% err. r.)



(c) $n_w = 50$



(d) $n_w = 200$

FIGURE 4.5: (a) and (b) MM performance on 4×4 *checkerboard* data; (c) and (d) 2-D *weights* data set

4.5.3 Preliminary results

In this section we present preliminary results of GAMM algorithm on synthetic data sets. We considered 4×4 *checkerboard* data set, and compared performance of AMM and GAMM. The data set is not linearly separable, and provides an excellent benchmark to show the benefits of the proposed algorithm. We created a balanced, two-class training set with 15,000 examples and a test set with 5,000 examples. We set parameters to default values, $c = 10$ for AMM and $c = 50$ for GAMM (due to its frequent introduction of new weights), probability $p = 0.2$, and set λ through cross-validation. We trained for 15 epochs. As seen from Figures 4.5a and 4.5b, AMM

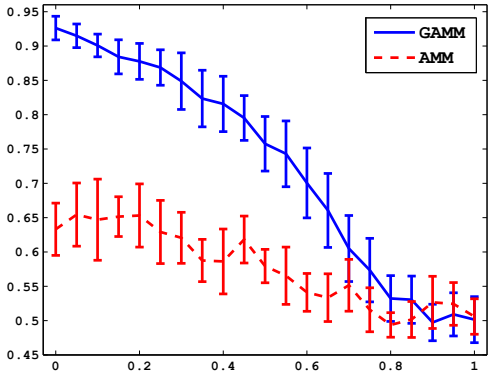


FIGURE 4.6: AMM and GAMM performance on noisy 4×4 *checkerboard* data set

could not solve this difficult non-linear problem, while GAMM achieved near-perfect class separation.

To further explore robustness of AMM algorithms, we added label noise to the 4×4 *checkerboard* data. We increased the percentage of noisy examples (to which we uniformly at random assigned a class label) from 0 to 1 in increments of 0.05, repeating each experiment 10 times. The accuracy and confidence intervals showing two standard deviations on noise-free test data are shown in Figure 4.6. GAMM shows resiliency to noisy labels and continues to strongly outperform AMM until noise levels were very high.

To get a better insight into representational power and stability of AMM, we ran experiments on *checkerboard* data set of different complexity. We changed the number of rows and columns of the checkerboard grid from 1 to 5, and report mean and standard deviation of classification error rate after 10 repetitions. The results are shown in Table 4.5. We can see that AMM could not solve 2×4 and 3×3 checkerboards. On the other hand, GAMM achieved significantly higher accuracy and was more stable than AMM in all cases.

Table 4.5: Error rate as a function of *checkerboard* pattern

Pattern	AMM	GAMM
1×2	0.48 ± 0.33	0.42 ± 0.29
1×3	1.39 ± 0.72	1.33 ± 0.62
1×4	3.23 ± 0.70	2.45 ± 0.51
1×5	4.90 ± 0.97	3.73 ± 0.71
2×2	1.49 ± 0.57	1.08 ± 0.40
2×3	2.98 ± 0.85	2.20 ± 0.52
2×4	10.33 ± 6.77	3.77 ± 0.89
2×5	20.79 ± 7.42	5.90 ± 1.61
3×3	16.92 ± 9.72	4.17 ± 0.77
3×4	24.47 ± 7.42	5.69 ± 0.93
3×5	34.87 ± 4.84	7.97 ± 1.43
4×4	35.87 ± 3.39	7.38 ± 1.53
4×5	35.13 ± 5.26	13.11 ± 1.84
5×5	43.27 ± 3.71	22.15 ± 2.08

CHAPTER 5

BUDGETEDSVM: A TOOLBOX FOR SCALABLE SVM APPROXIMATIONS

5.1 Introduction

Support Vector Machines (SVMs) are among the most popular and best performing classification algorithms. Kernel SVMs deliver state-of-the-art accuracies in non-linear classification, but are characterized by linear growth in the number of support vectors with data size, which may prevent learning from truly large data. In contrast, linear SVMs cannot capture non-linear concepts, but are much more scalable and allow learning from large data with limited computing resources. Aimed at bridging the representability and scalability gap between linear and non-linear SVMs, recent advances in large-scale learning resulted in several powerful algorithms that enable accurate and scalable training of non-linear SVMs, such as Adaptive Multi-hyperplane Machines (AMM) (Wang et al., 2011), Low-rank Linearization SVM (Zhang et al., 2012), and Budgeted Stochastic Gradient Descent (BSGD) (Wang et al., 2012). With accuracies comparable to kernel SVM, these algorithms are scalable to millions of examples, having training and prediction times comparable to linear SVM and orders

of magnitude shorter than kernel SVM.

We present BudgetedSVM, an open-source C++ toolbox for scalable, non-linear, multi-class classification. The toolbox provides an Application Programming Interface (API) for efficient training and testing of the aforementioned non-linear classifiers, supported by data structures designed for handling large-scale, high-dimensional data which cannot fit in memory. BudgetedSVM can be seen as a missing link between the LibLinear and the LibSVM packages (Hsieh et al., 2008; Chang and Lin, 2011), combining the efficiency of linear SVM with the accuracy of kernel SVM models. We also provide command-line and Matlab interfaces to the toolbox, which, influenced by the simplicity of LibSVM and LibLinear, provide users with an efficient, easy-to-use tool for large-scale non-linear classification.

5.2 Non-linear Classifiers for Large-scale Data

Before taking a closer look at the implementation and usage details of the BudgetedSVM toolbox, in this section we give a brief description of the implemented algorithms.

5.2.1 Adaptive Multi-hyperplane Machines (AMM)

Wang et al. (2011) proposed a classifier that captures non-linearity by assigning a number of linear hyperplanes to each of C classes from a set $\mathcal{Y}$. Given a D-dimensional example $\mathbf{x}$, the AMM multi-class classifier has the following form,

$$f(\mathbf{x}) = \arg\max_{i \in \mathcal{Y}} g(i, \mathbf{x}), \quad \text{where } g(i, \mathbf{x}) = \max_{j=1,\ldots,b_i} \mathbf{w}_{ij}^{\mathrm{T}} \mathbf{x}, \tag{5.1}$$

where the i^{th} class is assigned b_i weight vectors with the total budget $B = \sum_i b_i$. AMM is learned via Stochastic Gradient Descent (SGD). The hyper-parameters include a regularization parameter λ, the number of training epochs e, the maximum number of non-zero weights per class B_{lim}, $b_i \leqslant B_{lim}$, and weight pruning parameters

k (pruning frequency) and c (pruning aggressiveness). As an initial guideline to the users, we experimentally found that for most data sets the values $e = 5$ (or $e = 1$ for very large data), $B_{lim} = 50$, $k = 10{,}000$, and $c = 10$ are appropriate, leaving only λ to be determined by cross-validation.

When $b_1, ..., b_C$ are fixed to 1, the AMM model reduces to linear multi-class SVM (Crammer and Singer, 2002), and the learning algorithm is equivalent to Pegasos, a popular linear SVM solver from Shalev-Shwartz et al. (2007b). As it is a widely-used linear SVM solver, we also provide the Pegasos algorithm directly as a shortcut in the BudgetedSVM toolbox.

5.2.2 Low-rank Linearization SVM (LLSVM)

Zhang et al. (2012) proposed to approximate kernel SVM optimization by a linear SVM using low-rank decomposition of the kernel matrix. The approximated optimization is solved via Dual Coordinate-Descent (DCD) (Hsieh et al., 2008). The binary classifier has the form

$$f(\mathbf{x}) = \text{sign}\big(\mathbf{w}^{\mathrm{T}}(\mathbf{M} \cdot g(\mathbf{x}))\big),$$

where $g(\mathbf{x}) = [k(\mathbf{x}, \mathbf{z}_1), \ldots, k(\mathbf{x}, \mathbf{z}_B)]^{\mathrm{T}}$, $\{\mathbf{z}_i\}_{i=1,\ldots,B}$ is a set of landmark points of size B, $k(\mathbf{x}, \mathbf{z}_i)$ is a kernel function, $\mathbf{w}$ defines a separating hyperplane in the linearized kernel space (found using the DCD method), and $\mathbf{M}$ is a $B \times B$ mapping matrix. The hyper-parameters include kernel parameters, regularization parameter λ, and the number of landmark points B. Parameter B controls a trade-off between speed and accuracy, while kernel parameters and λ are best determined by cross-validation.

5.2.3 Budgeted Stochastic Gradient Descent (BSGD)

Wang et al. (2012) proposed a budgeted algorithm which maintains a fixed number of support vectors in the model, and incrementally updates them during the SGD

Table 5.1: Time and space complexities of the classification algorithms

	Pegasos	AMM	LLSVM	BSGD	RBF-SVM
Training time	$\mathcal{O}(NCS)$	$\mathcal{O}(NSB)$	$\mathcal{O}(NSB^2 + NSB)$	$\mathcal{O}\big(N(C+S)B\big)$	$\mathcal{O}(INCS)$
Prediction time	$\mathcal{O}(CS)$	$\mathcal{O}(SB)$	$\mathcal{O}(SB^2 + SB)$	$\mathcal{O}\big((C+S)B\big)$	$\mathcal{O}(NCS)$
Model size	$\mathcal{O}(CD)$	$\mathcal{O}(DB)$	$\mathcal{O}(DB + B^2)$	$\mathcal{O}\big((C+D)B\big)$	$\mathcal{O}(NCS)$

training. The multi-class BSGD classifier has the same form as (5.1), but with $g(i, \mathbf{x})$ defined as

$$g(i, \mathbf{x}) = \sum_{j=1}^{B} \alpha_{ij} k(\mathbf{x}, \mathbf{z}_j),$$

where $\{\mathbf{z}_j\}_{j=1,\ldots,B}$ is the support vector set, and α_{ij} is a class-specific parameter associated with the j^{th} support vector. We implemented Pegasos-style training, where the budget is maintained through either merging (where RBF kernel is used) or random removal of support vectors. The hyper-parameters include the number of epochs e, kernel parameters, regularization parameter λ, and budget size B. Parameters B and e control a speed-accuracy trade-off, while kernel parameters and λ are best determined by cross-validation.

5.2.4 Time and space complexity

Time and space complexities of the algorithms are summarized in Table 5.1, where N is the number of training examples, C is the number of classes, D is the data dimensionality, data sparsity S is the average number of non-zero features, and B is the model size for AMM, BSGD, and LLSVM. Parameter I for SVM with RBF kernel (RBF-SVM) denotes a number of training iterations, empirically shown to be super-linear in N (Chang and Lin, 2011).

5.3 The Software Package

BudgetedSVM can be found at *http://dabi.temple.edu/budgetedSVM/*. The software package provides a C++ API, which comprises functions for training and testing

of non-linear models described in Section 5.2. Each model can be easily trained and tested by calling the corresponding *train/predict* function, defined in `mm_algs.h`, `bsgd.h`, and `llsvm.h` header files. The API also provides functionalities for handling large-scale, high-dimensional data sets, defined in `budgetedSVM.h` header file.

BudgetedSVM sequentially loads data chunks into memory to allow large-scale training, storing to memory only indices and values of non-zero features as a linked list. Furthermore, implementation of sparse vectors is optimized for high-dimensional data, allowing faster kernel computations and faster updates of hyperplanes and support vectors than linked list (e.g., as in LibSVM) or array implementation of vectors (e.g., as in MSVMpack by Lauer and Guermeur, 2011) used for regular-scale problems, where either time or memory costs can become prohibitively large during training in a large-scale setting. In particular, vectors are split into disjoint chunks where pointers to each chunk are stored in an array, and memory for a chunk is allocated only if one of its elements is non-zero. While significantly reducing time costs, we empirically found that this approach incurs very limited memory overhead even for data with millions of features. Consequently, BudgetedSVM vector reads and writes are performed memory-efficiently in constant time. Moreover, by storing and incrementally updating support vector ℓ_2-norms after each training step, time to compute popular kernels (e.g., linear, Gaussian, polynomial) scales only linearly with sparsity S. Further implementation details can be found in a comprehensive developer's guide.

We also provide command-line and Matlab interfaces for easier use of the toolbox, which follow the user-friendly format of LibSVM and LibLinear. For example, we type `budgetedsvm-train -A 1 a9a_train.txt a9a_model.txt` in the command prompt to train a classifier on the *adult9a* data set. The `-A 1` option specifies that we use the AMM algorithm, while the data is loaded from `a9a_train.txt` file and the trained model is stored to the `a9a_model.txt` file. Similarly, we type

Table 5.2: Error rates (in %) and training times[1] on benchmark data sets

	Pegasos		**AMM**			**LLSVM**			**BSGD**			**RBF-SVM**	
Data set	err.	time	err.	B	time	err.	B	time	err.	B	time	err.	time
webspam						3.46	$5e2$	2.5m	2.04	$5e2$	2.0m		
N = 280,000	7.94	0.5s	4.74	9	3s	2.60	$1e3$	6.1m	1.72	$1e3$	3.9m	0.77	4.0h
D = 254						1.99	$3e3$	0.5h	1.49	$3e3$	0.2h	(#SV: 26,447)	
rcv1						4.97	$5e2$	0.2h	3.33	$5e2$	0.8h		
N = 677,399	2.73	1.5s	2.39	19	9s	4.23	$1e3$	0.5h	2.92	$1e3$	1.5h	2.17	20.2h
D = 47,236						3.05	$3e3$	2.2h	2.53	$3e3$	4.4h	(#SV: 50,641)	
mnist8m-bin						6.84	$5e2$	1.6h	2.23	$5e2$	2.3h		
N = 8,000,000	22.71	1.1m	3.16	18	4m	4.59	$1e3$	3.8h	1.92	$1e3$	4.9h	0.43	N/A[2]
D = 784						2.59	$3e3$	15h	1.62	$3e3$	16.1h		

`budgetedsvm-predict a9a_test.txt a9a_model.txt a9a_output.txt` to evaluate the trained model, which loads the testing data from `a9a_test.txt` file, the model from `a9a_model.txt` file, and stores the predictions to `a9a_output.txt` file. We also provide a short tutorial which outlines the basic steps for using the BudgetedSVM interfaces.

5.3.1 Performance comparison

BudgetedSVM can learn an accurate model even for data with millions of data points and features, with training times orders of magnitude faster than RBF-SVM trained using LibSVM. For illustration, in Table 5.2 we give comparison of error rates and training times on binary classification tasks using several large-scale data sets (Wang et al., 2011). On *webspam* and *rcv1* it took LibSVM hours to train RBF-SVM, while BudgetedSVM algorithms with much smaller budgets achieved high accuracy within minutes, and even seconds in the case of AMM. Similarly, RBF-SVM training on large-scale *mnist8m-bin* could not be completed in a reasonable time on our test machine, while the implemented algorithms were trained within a few hours on extremely limited budgets to achieve low error rates. More detailed analysis of the BudgetedSVM algorithms can be found in their respective papers.

[1] We excluded data loading time; evaluated on Intel® E7400 with 2.80GHz processor, 4GB RAM.

[2] Listed accuracy obtained after 2 days of P-packSVM training on 512 processors (Zhu et al., 2009).

CHAPTER 6

DISTRIBUTED LEARNING OF CLASSIFICATION MODELS

6.1 Introduction

Recent advent of high-throughput applications which generate data sets that can easily reach terabytes in size, such as remote sensing, crowd-sourcing, high-energy physics, social networks, or computational advertising, has brought forward a clear need for computational approaches that can efficiently learn from Big Data problems (Bizer et al., 2012; Labrinidis and Jagadish, 2012; Lohr, 2012). Emerging conferences that specifically address the Big Data issues, as well as the number of recent publications related to large-scale tasks, underline the significance of the Big Data field. Moreover, recently introduced "Big Data Research and Development Initiative" by the United States government, aimed at providing support for these efforts, clearly indicates globally-recognized, strategic importance, as well as future potential and impact of Big Data-related research (Mervis, 2012).

With the emergence of extremely large-scale data sets, researchers in machine learning and data mining communities are faced with numerous challenges related

to the sheer size of the problems at hand, as many well-established classification and regression approaches were not designed and are not suitable for such memory- and time-intensive tasks. The inadequacy of standard machine learning tools in this new setting has led to investment of significant research efforts into the development of novel methods that can address such challenges. Classification tasks are of particular interest, as the problem of classifying input data examples into one of finite number of classes can be found in many areas of machine learning. However, state-of-the-art non-linear classification methods, such as Support Vector Machines (SVMs) (Cortes and Vapnik, 1995), are not applicable to truly big data due to very high time and memory overhead, which are in general super-linear and linear in the data size N, respectively, significantly limiting their use when solving large-scale problems. Several methods have been proposed to make SVMs more scalable, ranging from algorithmic speed-ups (Platt, 1998; Kivinen et al., 2002; Vishwanathan et al., 2003; Severyn and Moschitti, 2010; Tsang et al., 2005; Rai et al., 2009), to parallelization approaches (Graf et al., 2004; Chang et al., 2007; Zhu et al., 2009). However, scalability of SVM training is inherently limited as non-linear SVMs are characterized by linear growth of model size with training data size N (Steinwart, 2003a). This led to an increased interest in linear SVM models (Gentile, 2002; Li et al., 2002; Shalev-Shwartz et al., 2007a; Fan et al., 2008; Yu et al., 2012), which have constant memory and $\mathcal{O}(N)$ training time. These linear models provide a scalable alternative to non-linear SVMs, albeit with a certain drop in prediction accuracy.

Unfortunately, even linear time complexity may not be sufficiently efficient for modern data sets stored across petabytes of memory space, requiring researchers to develop and adopt new machine learning approaches in order to address extremely large-scale classification tasks. Significant research efforts culminated in several highly-influential frameworks for solving parallelizable problems that involve data sets which can not be loaded on a single machine. These frameworks for paral-

lel computations include MapReduce (Dean and Ghemawat, 2008, 2010), AllReduce (Agarwal et al., 2011), GraphLab (Low et al., 2010, 2012), Pregel (Malewicz et al., 2010), and others. MapReduce in particular has become very popular framework in industry, with major companies such as Yahoo!, Google, and Facebook spearheading its use in large-scale commercial systems (Borthakur et al., 2011; Shvachko et al., 2010).

Unlike other distributed frameworks that assume frequent communication and shared memory between the computation nodes (e.g., Agarwal et al., 2011; Low et al., 2010), MapReduce framework, and its open-source implementation called Hadoop, allows limited communication overhead between the nodes, which results in very strong fault-tolerance and guaranteed consistency. These favorable properties of MapReduce led to development of parallelizable variants of popular machine learning algorithms, such as k-means, perceptron, logistic regression, PCA, and others (Böse et al., 2010; Lin et al., 2011; Gesmundo and Tomeh, 2012; Chu et al., 2007). However, the proposed classification methods mostly rely on iterative training and two-way communication between the computation nodes (Gesmundo and Tomeh, 2012; Chu et al., 2007). This may impose significant costs during training as it does not closely follow the computational paradigm of MapReduce, which derives its reliability from the high level of autonomy of computation nodes.

In this paper we describe an efficient linear SVM learner with sub-linear training time, capable of fully employing the MapReduce framework to significantly speed up the training. The algorithm uses recently proposed Confidence-Weighted (CW) linear classifiers (Dredze et al., 2008; Crammer et al., 2009) to train a number of SVM models on each of the mappers. Following completion of the map step, the local CW models are sent to the reducer that optimally combines local classifiers to obtain a single model, more accurate than any of the individual ones. Compared to the CW algorithms, the proposed method, named AROW-MapReduce (AROW-

MR), allows significantly more efficient training of accurate SVMs on extremely large data sets due to the distributed training. We validate our approach on real-world, large-scale problem of Ad Latency prediction with nearly one billion data examples, where AROW-MR achieved higher accuracy and faster training than the baseline approaches.

6.2 Confidence-Weighted Classification

In this section we review the recently proposed confidence-weighted classifiers. We first detail the CW algorithm, proposed in Dredze et al. (2008), followed by the description of Adaptive Regularization of Weights (AROW) algorithm from Crammer et al. (2009), an improved CW method shown to significantly outperform the original CW algorithm.

First described in Dredze et al. (2008), the CW algorithm is a linear classifier that, in addition to the prediction margin for the new data example, also outputs probability of the correct classification. This is achieved by maintaining a multivariate Gaussian distribution over the separating hyperplanes, and during the training procedure both mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ of the distribution are learned. In this way a more expressive and informative model is found, giving us information about noise in each of the individual features, as well as about the relationship between features.

Let us assume that a trained CW model, with known mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, is given. Then, for an example $(\mathbf{x}, y)$ from data set $\mathcal{D}$, described by feature vector $\mathbf{x}$ and binary label $y \in \{-1, 1\}$, this induces a Gaussian distribution over the prediction margin $\hat{y}$ as follows,

$$\hat{y} \sim \mathcal{N}\big(y(\boldsymbol{\mu}^{\mathrm{T}}\mathbf{x}), \mathbf{x}^{\mathrm{T}}\Sigma\mathbf{x}\big). \tag{6.1}$$

Following (6.1), we can compute the probability of correct classification by using the

equation for the normal cumulative distribution function, to obtain the expression

$$\mathbb{P}(\text{sign}(\boldsymbol{\mu}^{\mathrm{T}}\mathbf{x}) = y) = \frac{1}{2}\big(1 + \text{erf}(\frac{y(\boldsymbol{\mu}^{\mathrm{T}}\mathbf{x})}{\sqrt{2\mathbf{x}^{\mathrm{T}}\Sigma\mathbf{x}}})\big). \tag{6.2}$$

The CW classifier is learned online, and the current model is updated each round after observing a training example. During training, our belief about the classifier before the t^{th} training iteration, expressed through the current mean $\boldsymbol{\mu}_t$ and the current covariance matrix $\boldsymbol{\Sigma}_t$, is updated so that the new example $(\mathbf{x}_t, y_t)$ is correctly classified with probability larger than some user-set parameter η. In addition, we impose an additional constraint that our new belief before iteration $t+1$ is not too far from our belief at the previous iteration t. More formally, the stated requirements yield the following optimization problem,

$$\begin{aligned} &(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) = \underset{\boldsymbol{\mu},\boldsymbol{\Sigma}}{\arg\min}\, D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)\big) \\ &\text{subject to} \;\; \mathbb{P}\big(y_t(\boldsymbol{\mu}^{\mathrm{T}}\mathbf{x}_t \geqslant 0)\big) \geqslant \eta, \end{aligned} \tag{6.3}$$

where D_{KL} is the Kullback-Leibler (KL) divergence. Since the problem (6.3) is non-convex, the authors of Dredze et al. (2008) solve an approximate convex problem, and derive closed-form updates for the parameters of the Gaussian distribution.

As formulated in (6.3), we seek such an update of the classification model so that the new training example is correctly classified with certain probability η. In Crammer et al. (2009), the authors point out that this may be suboptimal for noisy data sets. More specifically, once the learning algorithm observes a noisy example, the update would modify the current model so that the noise is correctly classified, which could have an adverse effect on the generalization performance of the classifier. To address this issue, a new CW formulation is proposed in Crammer et al. (2009), called Adaptive Regularization of Weights (AROW). In this approach, the following

problem is solved,

$$(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) = \arg\min_{\boldsymbol{\mu},\boldsymbol{\Sigma}} D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)\big) + \lambda_1 \big(\max(0, 1 - y_t \boldsymbol{\mu}^{\mathrm{T}} \mathbf{x}_t)\big)^2 + \lambda_2 (\mathbf{x}_t^{\mathrm{T}} \Sigma \mathbf{x}_t). \tag{6.4}$$

We can see that at each training step the old and the new belief are still constrained to be close, as measured by the KL-divergence. However, unlike in the CW algorithm which aggresively updates the model in order to accomodate new examples, in AROW formulation the aggresiveness of maximization of margin and minimization of uncertainty for the new example are controlled by the regularization parameters λ_1 and λ_2, respectively. As shown in Crammer et al. (2009), after finding the derivative of the objective function with respect to the parameters, update equations for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ in the case of misclassification of the t^{th} example (i.e., when $\text{sign}(\boldsymbol{\mu}_t^{\mathrm{T}} \mathbf{x}_t) \neq y_t$), can be written in closed-form,

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t + \alpha_t y_t \boldsymbol{\Sigma}_t \mathbf{x}_t, \\ \boldsymbol{\Sigma}_{t+1} &= \boldsymbol{\Sigma}_t - \beta_t \boldsymbol{\Sigma}_t \mathbf{x}_t \mathbf{x}_t^{\mathrm{T}} \boldsymbol{\Sigma}_t, \end{aligned} \tag{6.5}$$

where α_t and β_t are computed as

$$\begin{aligned} \alpha_t &= \beta_t \max(0, 1 - y_t \boldsymbol{\mu}^{\mathrm{T}} \mathbf{x}_t), \\ \beta_t &= (\mathbf{x}_t^{\mathrm{T}} \Sigma \mathbf{x}_t + r)^{-1}, \end{aligned} \tag{6.6}$$

and $r = 1/(2\lambda_1)$, for $\lambda_1 = \lambda_2$. Online AROW training is initiated with a zero-vector $\boldsymbol{\mu}_0$ and an identity matrix $\boldsymbol{\Sigma}_0$, and it further proceeds to observe training examples and to update the parameters using equations (6.5) and (6.6).

6.3 MapReduce framework

With the recent explosive growth of data set sizes, analysis and knowledge extraction from modern data sets using a single machine is becoming increasingly intractable.

In particular, training time of popular classification and regression methods (e.g., SVM, classification trees) is at best linear in training set size, which may be too expensive for problems with billions of examples. To address this pressing issue, a number of frameworks for distributed learning on clusters of computation nodes has been introduced, offering different levels of parallelization, node independence, and reliability (Dean and Ghemawat, 2008, 2010; Agarwal et al., 2011; Low et al., 2010, 2012; Malewicz et al., 2010). In this section, we describe one such framework which has become very popular in industry, called MapReduce. We also discuss AllReduce distributed framework, which is utilized in Vowpall Wabbit, a state-of-the-art distributed machine learning package.

MapReduce framework (Dean and Ghemawat, 2008, 2010), implemented as an open-source platform Hadoop[1], consists of two distinct phases called *map* and *reduce*, which constitute one MapReduce *job*. In the map phase, mappers read parts of the dataset (possibly stored on multiple computers) and perform some action (e.g., filtering, grouping, counting, sorting) with final results being sent to reducer in a form of ordered (*key*, *value*) pairs. In the reduce phase, reducer performs a summary operation on the data received from the mappers, where the received data is sorted by their key values. There may be multiple mappers and reducers, and the framework guarantees that all values associated with a specific key will appear in one and only one reducer. Note that the limited communication between computation nodes in MapReduce framework, which is allowed only from mappers to reducers, ensures high independence of mappers and significant fault-tolerance of the framework. Even in the case of mapper failure, the entire job is not significantly affected as remaining mappers are not aware of the failure, which can be fixed with a simple restart of the failed node.

We illustrate MapReduce abstraction on a simple example. Given a dataset $\mathcal{D}$

[1] http://hadoop.apache.org/, accessed November 2013

with D features, we may want to find how many times each feature has a non-zero value, which can be achieved using several mappers and a single reducer. Each mapper reads a part of the dataset, and for each example outputs $(k, 1)$ if the k^{th} feature was non-zero. When the mappers finish outputing (*key*, *value*) pairs, the reducer starts reading these pairs sorted by their key. Then, on the reducer side, we initialize the count variable to 0, and add all values associated with the same key as the ordered pairs are received. Once a key that is different from the one associated with the current count is read, reducer outputs the total count and resets the variable to compute the non-zero count for the following feature. In this way there is no need to store the individual counts, which lowers memory cost of the reducer.

There are several ways of utilizing MapReduce for distributed learning: 1) read the data using multiple mappers, and learn the model on a reducer in an online learning manner; 2) maintain a global model that is used by all mappers to compute partial updates, which are aggregated on reducer to update the global model (requires running multiple MapReduce jobs for convergence); and 3) learn several local models on mappers, and combine them into a global one on a reducer. For the first option, distributed learning takes the same amount of time as learning on a single machine, with the benefit that there is no need to store the data on a single disk. The second option is typically used for batch learning (Lin et al., 2011; Gesmundo and Tomeh, 2012; Chu et al., 2007), where each mapper computes partial gradient using the current model, while the reducer sums the partial gradients and updates the model. A new MapReduce job is then instantiated, with the updated model used by all mappers for the next round of gradient calculation. Thus, one job is analogous to one gradient descent step. Since learning may require several iterations to converge, multiple MapReduce jobs need to be ran one after another, which may be ineffective and time costly. In contrast, the third approach ensures more robust learning and small communication overhead as only a single job is run, and we utilize this approach

to propose an efficient and accurate classifier in Section 6.4.

6.3.1 AllReduce framework

MapReduce abstraction allows very limited interaction between the computation nodes to ensure high fault-tolerance. In the following we introduce significantly less-constrained framework called AllReduce (Agarwal et al., 2011), which is utilized by the popular Vowpal Wabbit (VW) software package[2] (Langford et al., 2009). Unlike MapReduce, AllReduce framework assumes communication between mappers as well, while the reducers are not used. In particular, when computing the update step for the current global model, partial update step computed on one mapper is communicated to all other mappers. Then, once every mapper receives a message from all other mappers, the aggregated update step is performed on all computation nodes simultaneously. A typical implementation of AllReduce is done by imposing a tree structure on the computation nodes, such that the partial messages are aggregated up the tree and then broadcasted down to all mappers.

Disadvantage of AllReduce framework is that the mappers need to run truly concurrently. However, it is common for large clusters to run many independent jobs requiring different amount of resources on the computation nodes, and for higher number of mappers there may be no guarantee that all of them will be available for concurrent execution. Furthermore, as we have observed in practice, due to the fact that all nodes are required to send their updates before the next learning iteration starts, AllReduce learning will stall if any individual mapper fails once the job has started.

[2] https://github.com/JohnLangford/vowpal_wabbit, accessed November 2013

6.4 Confidence-Weighted Classification using MapReduce Framework

In this section we present a distributed AROW algorithm, which can be used to efficiently train very accurate linear classifiers on large-scale data. Let us assume that we have M mappers, and on each mapper an AROW model is trained using only a subset of the whole data set $\mathcal{D}$, by running the algorithm described in Section 6.2. More specifically, on the m^{th} mapper an AROW model is trained using a data set $\mathcal{D}_m \subset \mathcal{D}$, such that $\bigcup_{m=1,\dots,M} \mathcal{D}_m = \mathcal{D}$ and $\mathcal{D}_i \cap \mathcal{D}_j = \varnothing, i \neq j$. We denote the trained AROW parameters for the m^{th} mapper as $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m$, which are sent to the reducer after the completion of the map stage. During the reduce stage, we learn the final, aggregated parameters $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$ such that the multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ is an optimal combination of M multivariate Gaussian distributions learned on mappers.

More formally, let us define objective function $\mathcal{L}$ to be minimized on the reducer,

$$\mathcal{L} = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}[D_{KL}^{S}\big(\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \| \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})\big)], \tag{6.7}$$

where the expectation is taken over the distributions over hyperplanes that separate the data set $\mathcal{D}$, and D_{KL}^{S} is the symmetric KL-divergence, defined as

$$D_{KL}^{S}(A \| B) = \frac{1}{2}\big(D_{KL}(A \| B) + D_{KL}(B \| A)\big). \tag{6.8}$$

As can be seen from (6.7), the reducer computes aggregated parameters $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$ such that the expected symmetric KL-divergence between the aggregated Gaussian distribution and hyperplane distributions, drawn from the probability distribution over separating hyperplane distributions for the data set $\mathcal{D}$, is minimized. We note that the proposed method can be viewed as a generalization of the averaging CW model used for large-scale data sets, briefly discussed in Dredze et al. (2008). Given the mapper-specific parameters $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m, m = 1, \dots, M$, empirical estimate of the

Algorithm 7 AROW-MapReduce (AROW-MR)

Inputs: Data set $\mathcal{D}$; number of mappers M
Output: Parameters of AROW-MR $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$

1. **Map:** Train the m^{th} AROW classifier on subset $\mathcal{D}_m$ of the data set $\mathcal{D}$ using equations (6.5) and (6.6), one AROW classifier for each mapper, to obtain $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m$, where $m = 1, \ldots, M$
2. **Reduce:** Combine the local AROW classifiers into an aggregated AROW classifier using equations (6.10), (6.11) and (6.12)
3. **Output** aggregated AROW parameters $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$

objective function $\mathcal{L}$ can be expressed as follows,

$$\mathcal{L} = \sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big)\ D_{KL}^{S}\big(\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \| \mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big), \tag{6.9}$$

where we define $\mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big)$, or probability of the m^{th} distribution over the separating hyperplanes, as the fraction of the training set used to train the m^{th} AROW classifier. We refer to the final CW classification model as AROW-MR.

6.4.1 Reducer optimization of AROW-MR

The optimization function (6.9) is convex, thus there exists a unique set of $(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ parameters that minimize $\mathcal{L}$. In this section we derive update equations for AROW-MR parameters, the mean and the covariance matrix of the aggregated Gaussian distribution over the separating hyperplanes.

In order to solve (6.9), we compute the first derivatives of the objecive function $\mathcal{L}$ with respect to the parameters of the aggregated Gaussian distribution. After finding the derivative of $\mathcal{L}$ with respect to $\boldsymbol{\mu}_*$ and equating the resulting expression

with 0, we obtain the following update rule for mean $\boldsymbol{\mu}_*$,

$$\boldsymbol{\mu}_* = \Big(\sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big) \ (\boldsymbol{\Sigma}_*^{-1} + \boldsymbol{\Sigma}_m^{-1}) \Big)^{-1} \\ \Big(\sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big) \ (\boldsymbol{\Sigma}_*^{-1} + \boldsymbol{\Sigma}_m^{-1}) \boldsymbol{\mu}_m \Big). \tag{6.10}$$

In order to compute the update rule for covariance matrix, we find the derivative of $\mathcal{L}$ with respect to $\boldsymbol{\Sigma}_*$ and equate the resulting equation with 0. After derivation, we obtain the following expression,

$$\boldsymbol{\Sigma}_* \Big(\sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big) \ \boldsymbol{\Sigma}_m^{-1} \Big) \boldsymbol{\Sigma}_* = \\ \sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big) \ \big(\boldsymbol{\Sigma}_m + (\boldsymbol{\mu}_* - \boldsymbol{\mu}_m)(\boldsymbol{\mu}_* - \boldsymbol{\mu}_m)^{\mathrm{T}}\big). \tag{6.11}$$

Equation (6.11) is a Riccati equation of the form $\mathbf{XAX} = \mathbf{B}$, solved with respect to matrix $\mathbf{X}$ with matrices $\mathbf{A}$ and $\mathbf{B}$ given. After finding the decomposition of matrix $\mathbf{A}$ as $\mathbf{A} = \mathbf{U}^{\mathrm{T}}\mathbf{U}$ (e.g., using the Cholesky decomposition), we can compute $\mathbf{X}$ in a closed-form using the following steps,

$$\begin{aligned} \mathbf{XAX} &= \mathbf{B} \\ \mathbf{XU}^{\mathrm{T}}\mathbf{UX} &= \mathbf{B} \\ \mathbf{UXU}^{\mathrm{T}}\mathbf{UXU}^{\mathrm{T}} &= \mathbf{UBU}^{\mathrm{T}} \\ (\mathbf{UXU}^{\mathrm{T}})^2 &= \mathbf{UBU}^{\mathrm{T}} \\ \mathbf{UXU}^{\mathrm{T}} &= \mathbf{U}^{0.5}\mathbf{B}^{0.5}(\mathbf{U}^{\mathrm{T}})^{0.5} \\ \mathbf{X} &= \mathbf{U}^{-0.5}\mathbf{B}^{0.5}(\mathbf{U}^{\mathrm{T}})^{-0.5}. \end{aligned} \tag{6.12}$$

By matching the elements of equation (6.12) with the elements of equation (6.11), we can find the closed-form solution for the covariance matrix $\boldsymbol{\Sigma}_*$. Then, in order

to find the optimal parameters $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$, equations (6.10) and (6.11) are solved iteratively until convergence (we empirically found that only a few iterations are sufficient for the optimization procedure to converge). The pseudocode given in Algorithm 7 summarizes the steps of the AROW-MR algorithm.

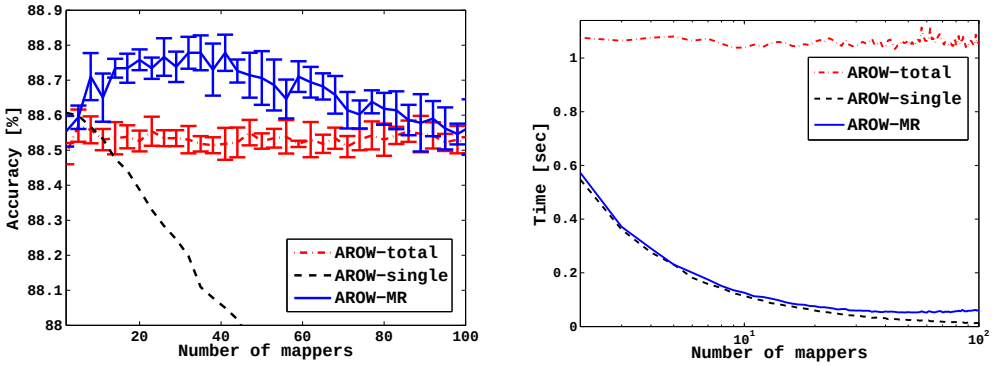
Let us discuss the time complexity of AROW and its distributed version AROW-MR. Given a D-dimensional data set $\mathcal{D}$ of size N, complexity of AROW training amounts to $\mathcal{O}(ND^2)$. On the other hand, complexity of AROW-MR is $\mathcal{O}(ND^2/M + MD^2 + D^3)$, where the first term is due to local AROW training on mappers, and the second and the third term are due to reducer optimization, which involves summation over M matrices of size $D \times D$ and Cholesky decomposition of the result, respectively. For large-scale data sets, for which it holds $N \gg M$, we can conclude that AROW-MR offers efficient training with significantly lower time overhead when compared to AROW algorithm.

6.5 Experiments

In this section we present the results of empirical evaluation of the AROW-MR algorithm. We first validated our method on a synthetic data set, then explored its performance on a real-world, large-scale task of Ad Latency prediction.

6.5.1 Validation on synthetic data

In order to better characterize the proposed distributed algorithm, in the first set of experiments we compared AROW and AROW-MR algorithms on synthetic data. We used the *waveform* data set generator, available from the UCI Repository, where we labeled the first and the second class as being positive and the third class as being negative. We generated 50,000 training examples and 5,000 test examples, and set $\lambda_1 = \lambda_2 = 0.1$ through cross-validation. We split the training set into M disjoint subsets of equal sizes and used one subset to train a local AROW on one mapper,



(a) Classification accuracy (b) Training time

FIGURE 6.1: Results on the synthetic *waveform* data set (with 50,000 training examples)

where we increased the number of mappers M from 2 to 100 to evaluate the effect of higher levels of parallelization. We report the results of the original AROW which used the entire training data set (denoted by AROW-total, which was not affected by the number of mappers), the results of AROW-MR, as well as the results of a local AROW model trained on a single mapper, denoted by AROW-single, for which the number of training examples decreased as the number of mappers was increased (i.e., number of training examples for each local model was $50{,}000/M$). We included AROW-single results to illustrate the performance of local AROW models that are eventually combined on the reducer to obtain AROW-MR model. Experiments on synthetic data were run in Matlab, on MacBook Pro with 2GHz Intel Core i7 with 8GB of DDR3 memory.

Mean accuracy and training time after 10 repetitions are shown in Figures 6.1a and 6.1b, respectively, where the error bars in Figure 6.1a represent intervals of two standard deviations (we omitted error bars for AROW-single as the standard deviation was around 0.5 and error bars would clutter the figure). We can see in Figure 6.1a that the accuracy of AROW-MR initially increased as the number of mappers increased, statistically significantly outperforming AROW-total. The accuracy of local

AROW models trained on each mapper (shown as AROW-single) dropped steadily with the increase of the number of mappers, which was expected as less training examples were used. Nevertheless, even though the accuracy of local models decreased, AROW-MR consistently outperformed AROW-total. Interestingly, as the number of mappers further increased, we can see that the accuracy of AROW-MR started decreasing when M surpassed 40, until it decreased to reach the accuracy of AROW for $M = 100$. This decrease is due to the fact that there are too few training examples on mappers for the local models to be close to convergence, which in turn affected accuracy of the aggregated model.

As illustrated in Figure 6.1b, distributed training also resulted in a significant speed-up in training time. We can see that AROW-MR training is order of magnitude faster than training of AROW-total, while at the same time achieving higher accuracy. Similarly to the accuracy results, the training time did not decrease further as we increased the number of mappers beyond a certain point. Although the mapper time continued to drop (shown by the dashed line), this was countered by longer time spent to solve the optimization problem (6.9) in the reduce phase due to larger M. This validates the known result that for certain problems "too much parallelization or distribution can have negative consequences" (Hughes and Hughes, 2004), and that the level of parallelization should be determined after deeper analysis of the problem being solved. Having said that, we can conclude that distributed training of AROW model resulted in significant accuracy and training time gains over the original AROW.

6.5.2 Ad Latency problem description

In the following section we compare the performance of AROW-MR and the baseline methods on large-scale, industrial task of Ad Latency prediction. However, before moving on to the discussion of empirical results, we first introduce this important

problem in online advertising, as well as the large-scale data set used in the experiments.

Over the previous decade, income generated by internet companies through online advertising has been growing steadily at amazing rates, with the total revenue reaching a record $36.6 billion in the US in 2012 alone[3]. This burgeoning, highly competitive market consists of several key players: 1) advertisers, companies that want to advertise their products; 2) publishers, websites that host advertisements; and 3) intermediate players that connect advertisers to publishers. In some cases such clear segmentation is not possible, and certain companies can be found in more than one role (e.g., Yahoo! or Google may provide both the products and the advertising space). Typically, advertiser designs an image of the advertisement, called a creative, specifying size and dimension requirements of the image to be shown on websites. This is then sent to the intermediate companies which have contracts with publishers, and which decide when and to whom the ads will be shown in order to maximize profits.

In order to retain existing and attract new users, publishers aim at improving user experience by minimizing page load times. In addition, equally important task for publishers is to ensure that the ads are delivered on time. Considering that ad latency time accounts for a significant percentage of the overall load time, improving ad load times would directly benefit both the user experience and the website revenue. Thus, correctly predicting ad latency time, and using this information to decide which ad should be shown to a user, is an extremely important problem in online advertising where an additional latency of several milliseconds could result in a significant loss of revenue. In this paper, we considered Ad Latency dataset consisting of nearly 1.3 billon ad impressions, for which 21 features were measured at serve time, along with ad latency given in milliseconds. Features can be divided into several groups:

[3] news.yahoo.com/us-internet-ad-revenue-grows-15-percent-2012-153752947, accessed Nov. 2013

- User-specific features include user's device type, operating system, and browser. We also used user's geographic location (i.e., state and city), user's physical distance from the colocation center serving the ad, user's connection speed (e.g., broadband, dial-up), as well as internet service provider used by the user.

- Advertiser-specific features include the advertiser's account ID, size of the advertisement (2-D dimensions of the creative, and its size in kilobytes), as well as the creative ID of a specific image used by the advertiser.

- Publisher's website can be partitioned into several regions, where each region has several spaces on which the ad can be shown. Further, ad can be placed at several different positions in the space (e.g., top, bottom). Thus, publisher-specific features include region ID, space ID, position, ad network used to serve the ad, serve type, hostname, as well as colocation center used to serve the ad.

- Lastly, we use time-stamp of ad impression, using time of day and day of the week.

In Table 6.1 we give the data set features, as well as the cardinalities for discrete features (we omitted sensitive information, which is marked with the '$\times$' symbol).

We can represent the problem as a binary classification by thresholding the value of ad latency. An ad is considered late (i.e., $y = 1$) if the time period from the moment when the webpage loads to the moment when the ad renders is longer than k milliseconds, and not late otherwise (i.e., $y = -1$). Value of k can be selected depending on a product or ad campaign requirements, and we omit the specific value used in the experiments as it represents a sensitive information.

6.5.3 Validation on Ad Latency data

In order to evaluate performance of the classification algorithms, we randomly split the Ad Latency data set into training set, consisting of 997,055,154 examples, and

Table 6.1: Features from the Ad Latency data set

Feature name	Cardinality
Device	15
Operating system	22
Browser	100
Connection speed	10
State	50
City	574
ISP	×
Distance to colocation center	continuous
Account ID	×
Creative ID	×
Ad size (dimensions)	33
Ad size (size in KB)	continuous
Region ID	×
Space ID (location on the page)	×
Ad position	28
Hostname	×
Ad network	×
Serve type	×
Colocation center	×
Hour of the day	24
Day of the week	7

non-overlapping testing set with 279,026,428 labeled examples. For the Ad Latency prediction task, the publishers prefer low False Positive Rate (FPR), ensuring that very few ads are wrongly classified as late, thus minimally hurting revenue. At the same time, the publisher prefer high True Positive Rate (TPR), which improves user experience and increases profit. As these two goals are often conflicting, the optimal strategy is to maximize the area under the Receiver Operating Characteristic (ROC) curve, referred to as the AUC (Fawcett, 2004). Thus, unlike in the experiments with the synthetic data, we report AUC as a measure of performance. Given a predicted margin for the n^{th} example, $\hat{y}_n \in \mathbb{R}$, a binary classification prediction is found as $\text{sign}(\hat{y}_n - \theta)$, where different values of threshold θ result in different predictions and different TPR and FPR values. We can obtain an ROC curve by sliding the threshold θ from $-\infty$ to ∞, and plotting the resulting TPR and FPR in a 2-D plot.

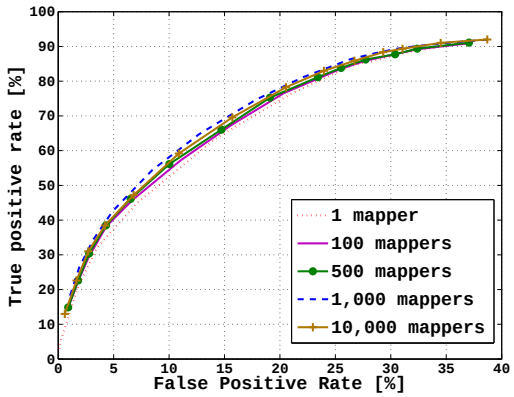


FIGURE 6.2: ROC curve for AROW-MR and AROW

Table 6.2: Performance comparison of AROW and AROW-MapReduce on Ad Latency task in terms of the AUC

# mappers	# reducers	Avg. map time	Reduce time	AUC
1	0	408h	n/a	0.8442
100	1	30.5h	1 min	0.8552
500	1	34 min	4 min	0.8577
1,000	1	17.5 min	7 min	0.8662
10,000	1	2 min	1h	0.8621

The experiments were conducted using MapReduce on Apache Hadoop, with AROW-MR mapper and reducer implemented in Perl. Performance of AROW-MR was compared to AROW algorithm which was run on a single machine, as well as to the logistic regression implemented in highly-scalable Vowpal Wabbit package, run both on a single machine and in a distributed manner using AllReduce on Hadoop. We ensured that Hadoop scheduled exactly M mappers by storing the data in M gzip-compressed part files.

We first compared the performance of AROW learned on a single machine with AROW-MR learned in a distributed manner. Similarly to experiments presented in Section 6.5.1, we increased the number of mappers to evaluate the effects of different levels of parallelization. The results are given in Table 6.2. We can see that the

Table 6.3: Performance of distributed logistic regression

# mappers	# reducers	Avg. map time	Reduce time	AUC
1	0	7h	n/a	0.8506
100	0	1h	n/a	0.8508
500	0	8 min	n/a	0.8501
1,000	0	6 min	n/a	0.8498

running time of AROW-MR drastically improved over the non-distributed AROW (trained using a single mapper). While it took 17 days for AROW to train, we were able to train accurate AROW-MR models in less than an hour. Expectedly, average mapper time decreased and reducer time increased as the number of mappers was increased, as each mapper was trained on smaller partition of the data and reducer was required to combine higher number of local models. Interestingly, we can also see that the results in Table 6.2 validate the results obtained on synthetic data regarding performance gains with increasing levels of parallelization. In particular, both the accuracy and training time improved until we reached M = 1,000, and dropped slightly for higher number of mappers. The detailed results are presented in Figure 6.2, where we plotted ROC curves of the confidence-weighted models trained using different number of mappers. We can see that the curve for non-distributed AROW, denoted by "1 mapper", results in the smallest AUC, while the level of parallelization achieved with 1,000 mappers represents the best choice for the Ad Latency prediction task.

Next, in Table 6.3 we show the performance of logistic regression (LR) model trained using VW package in both distributed mode (using AllReduce) and non-distributed manner (on a single machine). We can see that AROW-MR achieved higher accuracy than LR, which is a very popular approach using in large-scale classification. While AROW-MR obtained AUC of 0.8662, the best AUC achieved by LR was 0.8508; this increase in accuracy may result in significant increase of

revenue in computational advertising domain. Interestingly, the results also indicate that more distributed training of LR actually hurt its generalization performance, which slightly dropped as the number of mappers was increased. Further, we can see that LR was trained in 8 minutes, while it took 25 minutes to train AROW-MR model. However, although LR training is seemingly faster than training of AROW-MR, it is important to emphasize that VW package implements logistic regression in C language while, for technical reasons, AROW-MR was implemented in Perl. Considering that Perl is not an optimal choice for mathematical computations, we expect AROW-MR training time to improve significantly once implemented in C.

It is also worth noting that we were not able to run the LR experiment for more than one thousand mappers. The LR implementation in VW package uses AllReduce framework, which requires that all mappers run concurrently without any node failures, otherwise the training might fail. However, this is usually hard to guarantee in practice even for larger clusters, and it further exemplifies the advantage of the proposed algorithm over the competition. We can conclude that AROW-MR offers robust, highly efficient training of accurate classifiers, outperfoming the existing state-of-the-art for extremely large-scale, industrial-size problems.

CHAPTER 7

CONCLUSION

With the emergence of extremely large-scale data sets, researchers in machine learning and data mining communities are faced with numerous challenges related to the sheer size of the problems at hand. Many well-established supervised and unsupervised tools were not designed and are not suitable for such memory- and time-intensive tasks, warranting the development of novel, scalable methods capable of addressing the emerging big data problems. In this thesis, we recognized this problem and proposed novel tools suitable for handling of large-scale data. First, we considered unsupervised approaches to data analysis, and explored and described new visualization methods for fast knowledge extraction. We introduced data visualization problem using method based on efficient data reordering, and also described method for smaller-scale data based on object matching. Next, we considered supervised setting and proposed algorithms, as well as highly-optimized software, for fast training of accurate classification models on large data, capable of learning state-of-the-art classifiers on data sets with millions of examples and features within minutes. We have provided both theoretical and empirical evidence validating the benefits and usefulness of the proposed methods when working with big data.

For future work we plan to further extend these ideas, and also make them more accessible to non-expert users. Regarding the proposed unsupervised, visualization approaches, an interesting open question is how to extend the Convex Kernelized Sorting algorithm to larger scales and different-cardinality matching sets, as well as to semi-supervised setting. In addition, developing an easy-to-use EM-ordering software would help bring large-scale visualization outside academic setting, and allow for easy, intuitive, fast data exploration and knowledge extraction to the broader community. Furthermore, there are several avenues we want to explore for further development of the proposed supervised approaches. We plan to extend BudgetedSVM with existing methods, such as Tighter Perceptron (Wang and Vucetic, 2009) and BPA (Wang and Vucetic, 2010a), as well as state-of-the-art classification algorithms that are yet to be published, in order to make the software package a more inclusive toolbox of budgeted SVM approximations. We also plan to further explore a novel GAMM algorithm briefly discussed in Section 4.5, as the preliminary results strongly indicate the advantage over the competing approaches. However, more extensive experiments are required to better characterize this promising research direction. In addition, it is interesting to see how MapReduce, Pregel, GraphLab, and other distributed frameworks could be better utilized to develop faster, more efficient algorithms, as the results presented in Chapter 6 strongly suggest the benefits of the distributed approach when considering time and space complexity, as well as the accuracy of the learned models.

To conclude, size and complexity of the modern data sets is continuing to grow, and there are many challenges awaiting data mining and machine learning researchers in the years to come. The methods and the ideas discussed in this thesis help pave the road for addressing and solving these problems, bringing us a step closer to finally converting an overwhelming information overload into our competitive advantage.

BIBLIOGRAPHY

Agarwal, A., Chapelle, O., Dudík, M., and Langford, J. (2011), "A reliable effective terascale linear learning system," *arXiv preprint arXiv:1110.4198*.

Aiolli, F. and Sperduti, A. (2005), "Multi-Class Classification with Multi-Prototype Support Vector Machines," *Journal of Machine Learning Research*.

Applegate, D., Bixby, R. E., Chvátal, V., Cook, W., Espinoza, D. G., Goycoolea, M., and Helsgaun, K. (2009), "Certification of an optimal TSP tour through 85,900 cities," *Operations Research Letters*, 37, 11–15.

Artero, A. O., de Oliveira, M. C. F., and Levkowitz, H. (2004), "Uncovering clusters in crowded parallel coordinates visualizations," in *IEEE Symposium on Information Visualization*, pp. 81–88, IEEE.

Bacon, D. J. and Anderson, W. F. (1986), "Multiple sequence alignment," *Journal of Molecular Biology*, 191, 153–161.

Bar-Joseph, Z., Demaine, E. D., Gifford, D. K., Hamel, A. M., Jaakkola, T., and Srebro, N. (2002), "K-ary Clustering with Optimal Leaf Ordering for Gene Expression Data," WABI '02, pp. 506–520.

Belkin, M. and Niyogi, P. (2003), "Laplacian Eigenmaps for dimensionality reduction and data representation," *Neural Computation*, 15, 1373–1396.

Bertin, J. and Barbut, M. (1967), *Sémiologie graphique: les diagrammes, les réseaux, les cartes*, Mouton Paris.

Biedl, T., Brejova, B., Demaine, E. D., Hamel, A. M., and Vinar, T. (2001), "Optimal Arrangement of Leaves in the Tree Representing Hierarchical Clustering of Gene Expression Data," Tech. Rep. CS-2001-14.

Bizer, C., Boncz, P., Brodie, M. L., and Erling, O. (2012), "The meaningful use of big data: four perspectives–four challenges," *ACM SIGMOD Record*, 40, 56–60.

Blandford, D. and Blelloch, G. (2002), "Index Compression through Document Reordering," DCC '02.

Bordes, A., Ertekin, S., Weston, J., and Bottou, L. (2005), "Fast kernel classifiers for online and active learning," *Journal of Machine Learning Research*.

Bordes, A., Bottou, L., and Gallinari, P. (2009), "SGD-QN: careful quasi-newton stochastic gradient descent," *Journal of Machine Learning Research*.

Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., Ranganathan, K., Molkov, D., Menon, A., Rash, S., et al. (2011), "Apache Hadoop goes realtime at Facebook," in *International Conference on Management of Data*, pp. 1071–1080, ACM.

Böse, J.-H., Andrzejak, A., and Högqvist, M. (2010), "Beyond online aggregation: Parallel and incremental data mining with online map-reduce," in *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, p. 3, ACM.

Boyd, S. and Vandenberghe, L. (2004), *Convex Optimization*, Cambridge University Press, New York, NY, USA.

Chang, C.-C. and Lin, C.-J. (2011), "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, 2, 27:1–27:27, Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Chang, E., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., and Cui, H. (2007), "Psvm: Parallelizing support vector machines on distributed computers," *Advances in Neural Information Processing Systems*, 20, 16.

Chang, Y.-W., C.-J. Hsie and, K.-W. C., Ringgaard, M., and Lin, C.-J. (2010), "Training and testing low-degree polynomial data mappings via linear svm," *Journal of Machine Learning Research*.

Christofides, N. (1976), "Worst-case analysis of a new heuristic for the traveling salesman problem," *SIAM Journal on Computing*, 6, 563–563.

Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2007), "Map-reduce for machine learning on multicore," *Advances in Neural Information Processing Systems*, 19, 281.

Climer, S. and Zhang, W. (2004), "Take a walk and cluster genes: a TSP-based approach to optimal rearrangement clustering," in *International Conference on Machine Learning*, ACM.

Coleman, T. F. and Li, Y. (1996), "An Interior Trust Region Approach for Nonlinear Minimization Subject to Bounds," *SIAM Journal on Optimization*, 6, 418–445.

Collobert, R., Sinz, F., Weston, J., and Bottou, L. (2006), "Trading convexity for scalability," in *International Conference on Machine Learning*.

Cortes, C. and Vapnik, V. (1995), "Support-vector networks," *Machine Learning*, 20, 273–297.

Crammer, K. and Singer, Y. (2002), "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of Machine Learning Research*, 2, 265–292.

Crammer, K., Gilad-Bachrach, R., Navot, A., and Tishby, N. (2002), "Margin analysis of the LVQ algorithm," *Advances in Neural Information Processing Systems*, 15, 462–469.

Crammer, K., Kulesza, A., and Dredze, M. (2009), "Adaptive Regularization of Weight Vectors," *Advances in Neural Information Processing Systems*, 22, 414–422.

Czekanowski, J. (1909), "Zur differential Diagnose der Neandertalgruppe," in *Korrespondenz-blatt der Deutsche Gesellschaft für Anthropologie, Ethnologie und Urgeschichte*, vol. XL(6/7), pp. 44–47.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954), "Solution of a large-scale traveling-salesman problem," *Oper. Research*, 2, 393–410.

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

Dean, J. and Ghemawat, S. (2010), "MapReduce: a flexible data processing tool," *Communications of the ACM*, 53, 72–77.

Del Corso, G. M. and Manzini, G. (1999), "Finding exact solutions to the bandwidth minimization problem," *Computing*, 62, 189–203.

DiMaggio, P. A., McAllister, S. R., Floudas, C. A., Feng, X.-J., Rabinowitz, J. D., and Rabitz, H. A. (2008), "Biclustering via optimal re-ordering of data matrices in systems biology: rigorous methods and comparative studies," *BMC Bioinformatics*, 9, 458.

Ding, C. and He, X. (2004), "Linearized cluster assignment via spectral ordering," in *International Conference on Machine learning*, pp. 30–37, ACM.

Djuric, N. and Vucetic, S. (2013), "Efficient Visualization of Large-scale Data Tables through Reordering and Entropy Minimization," in *Proceedings of the IEEE International Conference on Data Mining*.

Djuric, N., Grbovic, M., and Vucetic, S. (2012), "Convex Kernelized Sorting," in *Proceedings of the AAAI Conference on Artificial Intelligence*.

Djuric, N., Lan, L., Vucetic, S., and Wang, Z. (2013a), "BudgetedSVM: A Toolbox for Scalable SVM Approximations," *Journal of Machine Learning Research*.

Djuric, N., Grbovic, M., and Vucetic, S. (2013b), "Distributed Confidence-Weighted Classification on MapReduce," in *Proceedings of the IEEE International Conference on Big Data*.

Dredze, M., Crammer, K., and Pereira, F. (2008), "Confidence-Weighted Linear Classification," in *Proceedings of the International Conference on Machine Learning*, pp. 264–271, ACM.

Eisen, M. B., Spellman, P. T., Brown, P. O., and Botstein, D. (1998), "Cluster analysis and display of genome-wide expression patterns," *PNAS USA*, 95, 14863–14868.

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

Fawcett, T. (2004), "ROC graphs: Notes and practical considerations for researchers," *Machine Learning*, 31, 1–38.

Friendly, M. (2002), "Corrgrams: Exploratory displays for correlation matrices," *The American Statistician*, 56, 316–324.

Friendly, M. (2006), "A Brief History of Data Visualization," in *Handbook of Computational Statistics: Data Visualization*, eds. C. Chen, W. Härdle, and A. Unwin, vol. III, pp. 15–56, Springer-Verlag.

Friendly, M. and Kwan, E. (2003), "Effect ordering for data displays," *Computational Statistics & Data Analysis*, 43, 509–539.

Fritzke, B. (1994), "Growing cell structures - A self-organizing network for unsupervised and supervised learning," *Neural Networks*, 7, 1441–1460.

Fritzke, B. (1995), "A growing neural gas network learns topologies," *Advances in Neural Information Processing Systems*, 7, 625–632.

Fua, Y.-H., Ward, M. O., and Rundensteiner, E. A. (1999), "Hierarchical parallel coordinates for exploration of large datasets," in *IEEE Conference on Visualization*, pp. 43–50, IEEE Computer Society Press.

Gentile, C. (2002), "A new approximate maximal margin classification algorithm," *Journal of Machine Learning Research*, 2, 213–242.

Gesmundo, A. and Tomeh, N. (2012), "HadoopPerceptron: a toolkit for distributed perceptron training and prediction with MapReduce," in *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 97–101, Association for Computational Linguistics.

Gomory, R. E. (1958), "Outline of an algorithm for integer solutions to linear programs," *Bulletin of the American Mathematical Society*, 64, 275–278.

Graf, H., Cosatto, E., Bottou, L., Dourdanovic, I., and Vapnik, V. (2004), "Parallel Support Vector Machines: The cascade SVM," *Advances in Neural Information Processing Systems*, 17, 521–528.

Guo, D. (2007), "Visual analytics of spatial interaction patterns for pandemic decision support," *International Journal of Geographical Information Science*, 21, 859–877.

Gutin, G. and Punnen, A. P. (2002), *The traveling salesman problem and its variations*, vol. 12, Springer.

Haghighi, A., Liang, P., Berg-Kirkpatrick, T., and Klein, D. (2008), "Learning Bilingual Lexicons from Monolingual Corpora," in *Proceedings of ACL-08: HLT*, pp. 771–779, Association for Computational Linguistics.

Hahsler, M., Hornik, K., and Buchta, C. (2007), "Getting Things in Order: An introduction to the R package seriation," .

Hastie, T., Simard, P., and Säckinger, E. (1995), "Learning prototype models for tangent distance," *Advances in Neural Information Processing Systems*, pp. 999–1006.

Helsgaun, K. (2000), "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic," *European Journal of Operational Research*, 126, 106–130.

Hinton, G. and Roweis, S. (2002), "Stochastic neighbor embedding," *Advances in Neural Information Processing Systems*, 15, 833–840.

Hornik, K., Stinchcombe, M., and White, H. (1989), "Multilayer feedforward networks are universal approximators," *Neural Networks*, 2, 359–366.

Hotelling, H. (1936), "Relation between two sets of variables," *Biometrica*, 28, 322–377.

Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., and Sundararajan, S. (2008), "A dual coordinate descent method for large-scale linear SVM," in *International Conference on Machine Learning*, pp. 408–415.

Hughes, C. and Hughes, T. (2004)*Parallel and distributed programming using C++*, pp. 31–32.

Inselberg, A. (1985), "The plane with parallel coordinates," *The Visual Computer*, 1, 69–91.

Inselberg, A. and Dimsdale, B. (1991), "Parallel Coordinates," in *Human-Machine Interactive Systems*, pp. 199–233, Springer.

Jagarlamudi, J., Juarez, S., and Daumé III, H. (2010), "Kernelized Sorting for Natural Language Processing," in *AAAI Conference on Artificial Intelligence*, pp. 1020–1025, AAAI Press.

Joachims, T. (2006), "Training linear svms in linear time," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Jonker, R. and Volgenant, A. (1987), "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, 38, 325–340.

Karp, R. M. (1972), "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, eds. R. E. Miller and J. W. Thatcher, pp. 85–103, Plenum Press.

Keim, D. A. (2001), "Visual exploration of large data sets," *Communications of the ACM*, 44, 38–44.

Keim, D. A. (2002), "Information visualization and visual data mining," *IEEE Trans. on Visualization and Computer Graphics*, 8, 1–8.

Keim, D. A., Mansmann, F., Schneidewind, J., and Ziegler, H. (2006), "Challenges in visual data analysis," in *International Conference on Information Visualization*, pp. 9–16, IEEE.

Kivinen, J., Smola, A. J., and Williamson, R. C. (2002), "Online learning with kernels," *IEEE Transactions on Signal Processing*, 52, 2165–2176.

Koehn, P. (2005), "Europarl: A Parallel Corpus for Statistical Machine Translation," in *Machine Translation Summit X*, pp. 79–86.

Kohonen, T. (2001), *Learning vector quantization*, Springer.

Kuhn, H. W. (1955), "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, 2, 83–97.

Labrinidis, A. and Jagadish, H. (2012), "Challenges and opportunities with big data," *Proceedings of the VLDB Endowment*, 5, 2032–2033.

Langford, J., Li, L., and Zhang, T. (2009), "Sparse online learning via truncated gradient," *The Journal of Machine Learning Research*, 10, 777–801.

Lauer, F. and Guermeur, Y. (2011), "MSVMpack: A Multi-Class Support Vector Machine Package," *Journal of Machine Learning Research*, 12, 2293–2296.

Li, Y., Zaragoza, H., Herbrich, R., Shawe-Taylor, J., and Kandola, J. (2002), "The perceptron algorithm with uneven margins," in *International Conference on Machine Learning*, pp. 379–386.

Liiv, I. (2010), "Seriation and matrix reordering methods: An historical overview," *Statistical Analysis and Data Mining*, 3, 70–91.

Lin, S. and Kernighan, B. W. (1973), "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, 21, 498–516.

Lin, Y., Lv, F., Zhu, S., Yang, M., Cour, T., Yu, K., Cao, L., and Huang, T. (2011), "Large-scale Image Classification: Fast Feature Extraction and SVM Training," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1689–1696, IEEE.

Lohr, S. (2012), "The age of big data," *New York Times*, 11.

Loosli, G., Canu, S., and Bottou, L. (2007), "Training invariant support vector machines using selective sampling," *Large Scale Kernel Machines, Cam-bridge, MA, MIT Press*.

Loua, T. (1873), *Atlas statistique de la population de Paris*, J. Dejey & Cie.

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010), "GraphLab: A new parallel framework for machine learning," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 340–349.

Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., and Hellerstein, J. M. (2012), "Distributed GraphLab: A framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, 5, 716–727.

Luo, B. and Hancock, E. R. (2001), "Structural Graph Matching Using the EM Algorithm and Singular Value Decomposition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23, 1120–1136.

Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. (2009), "Identifying Suspicious URLs: An Application of Large-Scale Online Learning," in *International Conference on Machine Learning*.

Mäkinen, E. and Siirtola, H. (2000), "Reordering the reorderable matrix as an algorithmic problem," in *Theory and Application of Diagrams*, pp. 453–468, Springer.

Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010), "Pregel: a system for large-scale graph processing," in *Proceedings of the ACM SIGMOD International Conference on Management of data*, pp. 135–146, ACM.

McCormick, W. T., Schweitzer, P. J., and White, T. W. (1972), "Problem decomposition and data reorganization by a clustering technique," *Operations Research*, 20, 993–1009.

Mervis, J. (2012), "Agencies Rally to Tackle Big Data," *Science*, 336, 22–22.

Parlett, B. N. and Landis, T. L. (1982), "Methods for Scaling to Doubly Stochastic Form," *Linear Algebra and its Applications*, 48, 53–79.

Petrie, W. M. F. (1899), *Sequences in prehistoric remains*, Reprint series in the social sciences, Bobbs-Merrill.

Pinar, A., Tao, T., and Ferhatosmanoglu, H. (2005), "Compressing Bitmap Indices by Data Reorganization," *ICDE05*, pp. 310–321.

Platt, J. (1998), "Fast training of Support Vector Machines using Sequential Minimal Optimization," *Advances in kernel methods - support vector learning, MIT Press*.

Platt, J., Cristianini, N., and Taylor, J. S. (2000), "Large margin DAGs for multiclass classification," in *Advance in Nueral Information Processing Systems*.

Quadrianto, N., Smola, A. J., Song, L., and Tuytelaars, T. (2010), "Kernelized Sorting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32, 1809–1821.

Rahimi, A. and Recht, B. (2007), "Random features for large-scale kernel machines," in *Advances in neural information processing systems*, pp. 1177–1184.

Rai, P., Daumé III, H., and Venkatasubramanian, S. (2009), "Streamed learning: one-pass SVMs," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pp. 1211–1216, Morgan Kaufmann Publishers Inc.

Rosenkrantz, D. J., Stearns, R. E., and Lewis II, P. M. (1977), "An Analysis of Several Heuristics for the Traveling Salesman Problem," *SIAM Journal on Computing*, 6, 563–581.

Roweis, S. (1998), "EM algorithms for PCA and SPCA," NIPS '97, pp. 626–632, MIT Press.

Roweis, S. T. and Saul, L. K. (2000), "Nonlinear Dimensionality Reduction by Locally Linear Embedding," *Science*, 290, 2323–2326.

Severyn, A. and Moschitti, A. (2010), "Large-scale support vector learning with structural kernels," in *Machine Learning and Knowledge Discovery in Databases*, pp. 229–244, Springer.

Shalev-Shwartz, S. and Singer, Y. (2007), "Logarithmic regret algorithms for strongly convex repeated games (Technical Report)," The Hebrew University.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007a), "Pegasos: Primal estimated sub-gradient solver for SVM," in *Proceedings of the 24th international conference on Machine learning*, pp. 807–814, ACM.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007b), "Pegasos: primal estimated sub-gradient solver for svm," in *International Conference on Machine Learning*, pp. 807–814.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010), "The hadoop distributed file system," in *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–10, IEEE.

Sinkhorn, R. and Knopp, P. (1967), "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, 21, 343–348.

Smola, A., Gretton, A., Song, L., and Schölkopf, B. (2007), "A Hilbert Space Embedding for Distributions," *Algorithmic Learning Theory*, 4754, 1–20.

Sonnenburg, S. and Franc, V. (2010), "COFFIN : a computational framework for linear svms," in *International Conference on Machine Learning*.

Steinwart, I. (2003a), "Sparseness of support vector machines," *Journal of Machine Learning Research*, 4, 1071–1105.

Steinwart, I. (2003b), "Sparseness of support vector machines," *Journal of Machine Learning Research*.

Tatu, A., Maass, F., Färber, I., Bertini, E., Schreck, T., Seidl, T., and Keim, D. A. (2012), "Subspace Search and Visualization to Make Sense of Alternative Clusterings in High-Dimensional Data," in *IEEE Symposium on Visual Analytics Science and Technology*, pp. 63–72, IEEE CS Press.

Tenenbaum, J. B., Silva, V. d., and Langford, J. C. (2000), "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science*, 290, 2319–2323.

Teo, C., Vishwanathan, S. V. N., Smola, A. J., and Le, Q. V. (2010), "Bundle Methods for Regularized Risk Minimization," *Journal of Machine Learning Research*.

Tripathi, A., Klami, A., Orešič, M., and Kaski, S. (2011), "Matching samples of multiple views," *Data Mining and Knowledge Discovery*, 23, 300–321.

Tsang, I. W., Kwok, J. T., and Cheung, P.-M. (2005), "Core vector machines: Fast SVM training on very large data sets," *Journal of Machine Learning Research*, 6, 363.

Vadapalli, S. and Karlapalem, K. (2009), "Heidi matrix: Nearest neighbor driven high dimensional data visualization," in *ACM SIGKDD Workshop on Visual Analytics and Knowledge Discovery*, pp. 83–92, ACM.

Van der Maaten, L. and Hinton, G. (2008), "Visualizing data using t-SNE," *Journal of Machine Learning Research*, 9, 2579–2605.

Vempala, S. S. (2012), "Modeling high-dimensional data: Technical perspective," *Comm. of the ACM*, 55, 112–112.

Verleysen, M., François, D., Simon, G., and Wertz, V. (2003), "On the effects of dimensionality on data analysis with neural networks," in *Artificial Neural Nets Problem solving methods*, pp. 105–112, Springer.

Vishwanathan, S. V. N., Smola, A. J., and Murty, M. N. (2003), "SimpleSVM," in *International Conference on Machine Learning*.

Walter, J., Ontrup, J., Wessling, D., and Ritter, H. (2003), "Interactive visualization and navigation in large data collections using the hyperbolic space," in *International Conference on Data Mining*, pp. 355–362, IEEE.

Wang, C. and Mahadevan, S. (2009), "Manifold Alignment without Correspondence," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pp. 1273–1278, Morgan Kaufmann Publishers Inc.

Wang, H.-Y. and Yang, Q. (2011), "Transfer Learning by Structural Analogy," in *AAAI Conference on Artificial Intelligence*, eds. W. Burgard and D. Roth, AAAI Press.

Wang, Z. and Vucetic, S. (2009), "Tighter Perceptron with Improved Dual Use of Cached Data for Model Representation and Validation," in *International Joint Conference on Neural Networks*, pp. 3297–3302.

Wang, Z. and Vucetic, S. (2010a), "Online passive-aggressive algorithms on a budget," in *International Conference on Artificial Intelligence and Statistics*, pp. 908–915.

Wang, Z. and Vucetic, S. (2010b), "Online training on a budget of support vector machines using twin prototypes," .

Wang, Z., Crammer, K., and Vucetic, S. (2010), "Multi-class pegasos on a budget," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 1143–1150.

Wang, Z., Djuric, N., Crammer, K., and Vucetic, S. (2011), "Trading representability for scalability: Adaptive multi-hyperplane machine for nonlinear classification," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Wang, Z., Crammer, K., and Vucetic, S. (2012), "Breaking the Curse of Kernelization: Budgeted Stochastic Gradient Descent for Large-Scale SVM Training," *Journal of Machine Learning Research*, 13, 3103–3131.

Wilkinson, L. and Friendly, M. (2009), "The history of the cluster heat map," *The American Statistician*, 63.

Williams, G. J., Christen, P., et al. (2008), "ReDSOM: relative density visualization of temporal changes in cluster structures using self-organizing maps," in *International Conference on Data Mining*, pp. 173–182, IEEE.

Yamada, M. and Sugiyama, M. (2011), "Cross-Domain Object Matching with Model Selection," *Journal of Machine Learning Research - Proceedings Track*, 15, 807–815.

Yeung, K. Y., Haynor, D. R., and Ruzzo, W. L. (2001), "Validating clustering for gene expression data," *Bioinformatics*, 17, 309–318.

Yu, H.-F., Hsieh, C.-J., Chang, K.-W., and Lin, C.-J. (2010), "Large linear classification when data cannot fit in memory," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Yu, H.-F., Hsieh, C.-J., Chang, K.-W., and Lin, C.-J. (2012), "Large linear classification when data cannot fit in memory," *ACM Transactions on Knowledge Discovery from Data*, 5, 23.

Zhang, K., Lan, L., Wang, Z., and Moerchen, F. (2012), "Scaling up Kernel SVM on Limited Resources: A Low-rank Linearization Approach," in *International Conference on Artificial Intelligence and Statistics*.

Zhang, T. (2004), "Solving large scale linear prediction problems using stochastic gradient descent," in *International Conference on Machine Learning*.

Zhu, Z. A., Chen, W., Wang, G., Zhu, C., and Chen, Z. (2009), "P-packSVM: Parallel primal gradient descent kernel SVM," in *IEEE International Conference on Data Mining*.