Applications of Support Vector-Based Learning

Róbert Ormándi

The supervisors are
Prof. János Csirik and *Dr. Márk Jelasity*

Research Group on Artificial Intelligence of the University of Szeged and the Hungarian Academy of Sciences

PhD School in Computer Science
University of Szeged

A thesis submitted for the degree of
Doctor of Philosophy

Szeged
2013

Preface

An increasing amount of data is stored in electronic form. This phenomenon increases the need of the methods that help to access the useful pieces of information that are hidden in ocean of the collected data. That is, methods to perform automated data processing are needed. The role of machine learning is to provide methods that support this progress, i.e. it provides algorithms that can automatically discover those hidden and non-trivial patterns in data that are useful for us.

In the focus of this thesis stands a special family of machine learning algorithms referred to as *support vector-based learners*. These algorithms are all based on the so-called maximal margin heuristic, which helps to select the final model from the suitable ones preserving the generalization ability of the model. These methods are quite general, and they have a numerous amazing properties like generalization capability or robustness to noisy data. However, to apply them to a specific task it is needed to adapt them to that particular task. This is challenging since the inappropriate adaptation can result in a drastic decrease in prediction performance or in the generalization capability; or cause a computationally infeasible situation (e.g. huge computational complexity or untreatable network load in a distributed setting).

The goal of this thesis is to examine the suitability and adaptability of

various support vector-based learning methods in numerous learning tasks and environments. During this, novel techniques and ideas, which are less frequent or maybe surprising, are investigated and discussed. The considered tasks themselves touch a wide range problems like time series forecasting, opinion mining, collaborative filtering, and the common binary classification problem. An uncommon computational environment is also concerned, namely the fully distributed environment.

Róbert Ormándi, July 2013.

Acknowledgements

In the last few year, numerous people had a positive effect on me to finish this thesis. Probably I will not be able to thank all of them directly, but here I try to emphasize some of them.

First of all I would like to express my thanks to my supervisors, Prof. János Csirik and Dr. Márk Jelasity, for their guidance, continuous inspiration on my research and additionally for letting me work in the Research Group on Artificial Intelligence of the University of Szeged and the Hungarian Academy of Sciences. They showed me interesting, yet undiscovered fields and taught me how I can tackle with them applying scientific mentality and knowledge.

I also would like to thank my colleagues and friends for their continuous support. They definitely helped me discover new ideas and made my period of PhD studies really pleasurable. They are listed in alphabetical order: András Bánhalmi, Róbert Busa-Fekete, Kornél Csernai, Richárd Farkas, István Hegedűs and György Szarvas. Additionally, I would like to thank David P. Curley and Veronika Vincze for correcting this thesis from a linguistic point of view. Moreover, I would like to thank István Hegedűs and Tamás Vinkó for reviewing this thesis.

Last, but not least I would like to thank my wife, Ildikó, for her endless love, support, and inspiration. She kept the home fire burning while I was

working on this thesis. I would like to dedicate this thesis to her and our son, Márk, for expressing my appreciation.

List of Algorithms

List of Figures

List of Tables

Contents

CHAPTER 1

Introduction

More and more data is accumulated around us. The year 2012 was called the year of "big data" meaning that various tools become easily available for managing very large-scale data, while the storage of data is getting cheaper and cheaper. This phenomenon—although the problem of machine learning has long been considered fundamental—continuously increases the need for machine learning algorithms that work *properly* on specific tasks, and operate *efficiently* in unusual settings (like in distributed computational environments), since without these algorithms we simply cannot extract useful information from the data. That is, without the appropriate application of machine learning algorithms, we are not able to utilize that large amount of data. However, achieving these goals is still challenging, since the inappropriate adaptation of a learning algorithm to a specific task can yield models that are far from the optimal ones. On the other hand, the naive adaptation of the algorithms can result in unexpected effects within the system that applies them (like huge, unbalanced load in a distributed system).

The aim of this thesis is to present various approaches which help achieve these goals (appropriate adaptation of the algorithms in terms of both algorith-

mic and system aspects) related to a specific learning algorithm family, referred to as *support vector-based learners*[1]. That is, we investigate the *adaptivity* of support vector-based learners to various tasks and computational environments. The main "take-home message" of the thesis can be summarized as follows. One can achieve significant improvements through applying powerful basic ideas, like the maximal margin heuristic of support vector-based learners, as building-blocks, and careful system design.

The thesis starts with an introductory chapter (Chapter 2), which summarizes the necessary background information related to support vector machines and fully distributed systems.

The main part of the thesis can roughly be divided into two distinct parts. In the first part (Chapters 3-5), we investigate the *algorithmic adaptivity* of support vector machines. That is, we focus on how we can adapt the basic idea of support vector-based learners, the maximal margin heuristics, to develop efficient algorithms to a wide-range of applications, like time series forecasting (in Chapter 3), domain adaptation (in Chapter 4), and recommender systems (in Chapter 5). The general work-flow here is that we consider a task, investigate the special aspects of the given problem, propose an algorithm which utilizes the observed characteristics of the problem by applying the maximal margin heuristics, and finally evaluate the algorithm in thorough empirical experiments against the state-of-the-art, and often idealized baselines. During the algorithm design, we take into account the computational efficiency, and other practical aspects of the proposed algorithms to get a practical result.

Chapter 3 provides an extension for the Least Square Support Vector Machine, which is more suitable for learning time series. After providing the details of the algorithm, the evaluation of the method follows against several baselines on three widely used benchmark datasets.

In Chapter 4, a general domain adaptation mechanism is proposed with two distinct instantiations (one of them is based on support vector machines). Then, the evaluation points out that while both instantiations outperform the baseline methods, the support vector-based approach is a more suitable choice

[1]The main concepts and properties of the algorithm family are detailed in Chapter 2.

Table 1.1: The relation between the chapters of the thesis and the referred publications (• denotes the *basic* publications, while ◦ refers to related publications).

	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
ICDM 2008 [89]	•				
TSD 2010 [91]		•			
EUROPAR 2010 [92]			•		
WETICE 2010 [90]			•		
EUROPAR 2011 [93]				•	•
CCPE 2012 [94]					•
SASO 2012 [55]				◦	◦
SISY 2012 [53]				◦	◦
EUROPAR 2012 [54]				◦	◦
ICML 2013 [114]				◦	◦

from the instantiations.

The next chapter (Chapter 5) has a twofold contribution. First, it proposes an unusual use-case of the support vector machine to validate the learnability of a recommender database generated from implicit user feedback. Second, it investigates how we can adapt the centralized collaborative filtering approaches in a fully distributed setting. In this way, this chapter relates to both parts of the thesis.

In the second part of the thesis (Chapters 5-7), we turn to examine the *system model aspect of adaptivity*. In this part, our main question is how we can implement support vector-based learning algorithms in a specific system model that is radically different from the usual ones.

In Chapter 6, we carefully define the addressed system model, referred to as fully distributed system model. Then, we propose a gossip-based support vector implementation called P2PEGASOS. The basic idea of the protocol is that online support vector models take random walks in the network while update themselves using the training samples available in the nodes. The algorithm has some notable properties, like fast convergence speed, applicability of fully distributed data (that is, there is no need to move the data from the nodes of the network), simplicity, and extreme tolerance to various network errors (like message drop and delay).

In the following chapter (Chapter 7), we improve significantly the P2PEGASOS algorithm by introducing a general ensemble learning component which increases the convergence speed of the algorithm by an order of magnitude while keeps all the original advantages. Here we provide a proof of the convergence of the algorithm.

Each chapter of the thesis is based on at least one accepted publication. Tab. 1.1 shows the relation among the chapters of the thesis and the more important[1] publications.

[1]For a complete list of publications, please visit the corresponding section of my website: `http://www.inf.u-szeged.hu/~ormandi/papers`

CHAPTER 2

Background

This work involves some topics that are based on common results from the field of machine learning and distributed systems. These topics clearly have a crucial role in understanding our results. So, we dedicate this chapter to overview some of the most important concepts from these fields.

In the next sections, we briefly introduce the problem of supervised learning and show the classic way of how the Support Vector Machines [24, 31, 119] learning algorithm tackles this problem. This overview gives us the key insight to understand the basic concepts of support vector-based learning. Later in this chapter, we turn to discuss the basics of the fully distributed systems. Here, we briefly overview the concept of overlay networks and describe two particular unstructured overlay network management protocols, called NEWSCAST [64], and T-MAN [63], respectively, which are used intensively in the later chapters of the thesis.

We want to emphasize that the results described here have been achieved by others.

2.1 Supervised Learning

Let us assume that we are given a labeled database in the form of pairs of feature vectors and their correct labels, i.e. $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{L}$. The constant d is the dimension of the problem (the number of features) and $\mathcal{L}$ denotes the set of labels. We distinguish between supervised *regression* problem and *classification* problem. When the range of labels is continuous e.g. $\mathcal{L} = \mathbb{R}$, we talk about regression. However, when it is discrete and finite, i.e. $\mathcal{L} = \{C_1, \ldots, C_l\}$, we refer to the problem as classification where the number of classes is l. The special case of classification when the number of possible class labels equals to two is called *binary classification problem*. In this case we assume that—without the loss of generality—the set of labels is $\mathcal{L} = \{-1, +1\}$. Slightly different learning algorithms can be applied for regression and classification, but the goal is similar in the case of both problems.

The main goal of *supervised learning* (including classification and regression) can be defined as follows. We are looking for a *model* $f : \mathbb{R}^d \to \mathcal{L}$ that correctly predicts the labels of the available feature vectors, and that can also *generalize* well; that is, which can predict unseen examples too. The generalization property of the model is crucial since a trained model is mainly used for predicting of unseen examples. To be able to achieve generalization it is assumed that the unseen examples come from the same probability distribution than those were used for training. This is one of the main assumptions of machine learning [84].

During the training phase, a supervised learning algorithm seeks for a suitable model from a predefined (by the training algorithm itself) model space called *hypothesis space*. This phase can be thought of as an optimization process, where we want to maximize prediction performance, which can be measured via, e.g. the number of feature vectors that are classified correctly over the training set. The search space of this problem is the hypothesis space and each method also defines a specific search algorithm that eventually selects one model from this space keeping in mind the preservation of the generalization property of the model.

Training algorithms that process the training database as a continuous stream of training samples (i.e. stream of $(\mathbf{x}_i, y_i)$ pairs) and evolve a model by updating it for each individual training sample according to some update rule are called *online learning algorithms*. These algorithms are often used in large scale learning problems [78], there is no need for storing the whole training database in the memory.

In various machine learning tasks, the available training data is often partitioned into a training set and an evaluation set. In this case we assume that during the training phase the algorithm accesses only to the training set and the evaluation set is completely hidden.

2.1.1 Support Vector Machine

In the previous section, we introduced the main concepts of the supervised learning. Here we discuss a particular supervised learning algorithm family—namely the family of Support Vector Machines (SVMs) [24, 31, 119]—which plays a central role in this work.

The name SVM or more generally support vector-based learning refers to a supervised learning family instead of a particular algorithm [31]. The algorithms belonging to this family were designed around a central idea called *maximal margin heuristic*. The basic idea of support vector-based learning is pretty simple: find a good learning boundary while maximizing the margin (i.e. the distance between the closest learning samples that correspond to different classes). In this way, each algorithm that optimizes an objective function in which the maximal margin heuristic is encoded can be considered a variant of the SVMs. In the following, to show some interesting properties of SVMs, we briefly discuss a particular SVM classification algorithm (and its soft-margin and kernel-based extensions).

Let us assume that we are in a binary classification setting, i.e. we have a set $\mathcal{D}$ of training samples containing n pairs in the form $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$.

In its simplest form the SVM is expressed as a hyperplane parametrized by the normal vector (that is orthogonal to the hyperplane) of the plane ($\mathbf{w} \in \mathbb{R}^d$)

and a scalar ($b \in \mathbb{R}$). Given such a hyperplane the model of the SVM is written in the following form:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b). \tag{2.1}$$

The geometrical interpretation of this kind of model is pretty clear: the model assigns label -1 to the inputs that are below the hyperplane and $+1$ that are above.

During the training phase the SVM seeks for the appropriate parameters that separate the training samples and maximize the margin. Formally it is done by searching in the space of *canonical hyperplanes* [31] corresponding to the training set $\mathcal{D}$. It is easy to see that a hyperplane with parameters $(\mathbf{w}, b)$ is equally expressed by all pairs of $(\lambda\mathbf{w}, \lambda b)$ parameters where $0 < \lambda \in \mathbb{R}$. Let us define a canonical hyperplane to be such that separates the training database correctly with a distance of at least 1, i.e.:

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b \geq 1 \text{ when } y_i = +1, \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 \text{ when } y_i = -1 \text{ for each sample } 1 \leq i \leq n \end{aligned} \tag{2.2}$$

or in a more compact form:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for each sample } 1 \leq i \leq n. \tag{2.3}$$

Given a canonical hyperplane $(\mathbf{w}, b)$, we can obtain the *geometrical distance* from the plane to a data point $\mathbf{x}_i$ by normalizing the magnitude of $\mathbf{w}$:

$$dist\left((\mathbf{w}, b), \mathbf{x}_i\right) = \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|} \geq \frac{1}{\|\mathbf{w}\|}. \tag{2.4}$$

As one can see, the geometrical distance between the canonical hyperplane $(\mathbf{w}, b)$ and each training point $\mathbf{x}_i$ can be bounded by the reciprocal of the norm of $\mathbf{w}$.

The SVM training algorithm wants to maximize the margin, i.e. the geometrical distance between the closest training points from the opposite classes in the space of canonical hyperplanes [31]. Given a canonical hyperplane $(\mathbf{w}, b)$—based on the bound in Eq. 2.4—the margin is at least $\frac{2}{\|\mathbf{w}\|}$ i.e. to maximize it, the training algorithm has to minimize the quantity $\|\mathbf{w}\|$ or equally $\frac{1}{2}\|\mathbf{w}\|^2$ (to get a more handy objective function).

Based on the definition of the canonical hyperplanes and the above observed bound on the margin, we can easily formalize the optimization problem of the SVM:

$$\begin{aligned} \text{minimize}_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad (\text{for each } 1 \leq i \leq n). \end{aligned} \tag{2.5}$$

As one can easily notice that the above defined optimization problem is pretty restricted in the sense, it is only defined for separable training sets (because of the concept of canonical hyperplanes). Extending this optimization problem to the so-called *soft-margin* [31] optimization can overcome this problem. To achieve this we introduce a non-negative $\xi_i \in \mathbb{R}$ $(1 \leq i \leq n)$ slack variable for each condition of the above defined optimization problem. For a given condition i, this variable models how that condition violates the condition of being a canonical hyperplane, i.e. the error corresponds to be canonical hyperplane. By introducing the slack variables, we tolerate some error in those conditions. However, if we want this error to be as small as possible, we have to extend the objective function as well by adding the slack variables to it. Based on the introduction of slack variables, we get the soft-margin optimization form of SVM classification:

$$\begin{aligned} \text{minimize}_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \quad (\text{for each } 1 \leq i \leq n). \end{aligned} \tag{2.6}$$

Here the $C \in \mathbb{R}$ is a hyperparameter which could be considered as a trade-

off between the error minimization and margin maximization.

During the SVM training we have to solve the optimization defined in Eq. 2.5 or Eq. 2.6. Now let us focus on the more general soft-margin form. The original form of this optimization problem showed above is called *primal form*. Numerous methods exist that solve the optimization problem in this original form [26]. However in some scenarios (e.g. in centralized computation models), there are some advantages—most importantly, one can apply the kernel-based extension that introduces nonlinearity into the approach [24, 119]—of transforming this form to the so-called *dual form*.

The dual form of Eq. 2.6 is found by first taking the Lagrangian of the optimization problem:

$$\begin{aligned} \mathrm{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\nu}) =& \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ & - \sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{n}\nu_i\xi_i \\ & \alpha_i \geq 0 \quad (\text{for each } 1 \leq i \leq n) \\ & \nu_i \geq 0 \quad (\text{for each } 1 \leq i \leq n). \end{aligned} \tag{2.7}$$

Then by differentiating with respect to $\mathbf{w}$, b and each element of $\boldsymbol{\xi}$ (i.e. respect to the non-Lagrangian variables), imposing stationarity, resubstituting the obtained constraints into the primal, and applying the Karush-Kuhn-Tucker (KKT) complementary conditions [24, 119], we eventually get the following dual form:

$$\begin{aligned} \text{maximize}_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i \\ \text{subject to} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C \quad (\text{for each } 1 \leq i \leq n). \end{aligned} \tag{2.8}$$

This is a so-called *quadratic programming* (QP) problem in the variable $\boldsymbol{\alpha} \in \mathbb{R}^n$. It is a quite general optimization problem having a unique and globally optimal solution [18]. Numerous efficient techniques have been developed to

solve QP problems [18] and some of them scale well with the size of training database. Nevertheless, the size of this optimization problem is very large (since usually it is $n \gg d$), which often causes serious problems in practice.

Additionally, from the derivation of the primal equations, it can be seen that the optimal hyperplane can be written as:

$$\begin{aligned} \mathbf{w}^\star &= \sum_{i=1}^{n} \alpha_i^\star y_i \mathbf{x}_i \\ b^\star &= \frac{1}{n} \sum_{i=1}^{n} \frac{1 - y_i \mathbf{w}^{\star T} \mathbf{x}_i}{y_i} \end{aligned} \tag{2.9}$$

where $\boldsymbol{\alpha}^\star \in \mathbb{R}^n$ is the optimal solution of the dual problem. That is, the vector $\mathbf{w}^\star$ is the linear combination of the training examples [24, 119]. Moreover, it also can be shown (based on the KKT conditions [119]) that only the closest data points (that are called support vectors) contribute to $\mathbf{w}^\star$ (i.e. have a value $\alpha_i^\star > 0$). Subsequently, the non-support vectors have a coefficient $\alpha_i^\star = 0$ and can be skipped from the model. This leads to the *sparse solution* [119] of the model:

$$f(\mathbf{x}) = \text{sign}\Big(\sum_{(x_i, y_i) \in SV} \alpha_i^\star y_i \mathbf{x}_i^T \mathbf{x} + b^\star \Big) \tag{2.10}$$

where $SV \subseteq \mathcal{D}$ denotes the set of support vectors, where usually $|SV| \ll |\mathcal{D}|$.

The above defined dual representation learns a robust and sparse linear model. But what if the relation between the data and the labels is strictly non-linear? The *kernel representation* offers a solution to tackle with the nonlinearity by projecting the data into a higher dimensional *feature space* to increase the representation power of the linear learning machine. That is, we apply a $\phi : \mathbb{R}^d \to \mathbb{R}^h$ nonlinear transformation, where $h \gg d$, to change the representation of each data point $\mathbf{x} \in \mathbb{R}^d$.

Mapping the data onto a higher dimensional space each time when we need it would be computationally inefficient. But one can easily recognize that the only way in which the data appears in the training problem in Eq. 2.8, and

in the model in Eq. 2.10 is in the form of inner products ($\mathbf{x}_i^T \mathbf{x}_j$). Now suppose we first transformed the data to the higher dimensional features space, using a mapping ϕ defined above. Then the training algorithm would only depend on the data through dot products in that feature space, i.e. on functions of the form $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. By defining the *kernel function* $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ we can avoid the explicit computation of mapping ϕ. This procedure is called kernel-based learning and can increase the representation power of SVMs strictly without significantly increasing the computational complexity. This is why it is pretty often applied together with support vector-base learning algorithms.

Other advantage of the application of kernels is that it can be considered as an external component of the learning which can be modified without modifying the learning algorithm itself. Different kernels exist and the construction of the kernels is theoretically well-studied (see e.g. Mercel-theorem [31]). Kernels that are used very often are the polynomial kernel, and the RBF kernel [31], however, kernels for specific tasks exist as well like the alignment kernel (e.g. for time series forecasting), string kernels, and graph kernels [31, 33].

The above described variant of SVM deals with classification problems. This model applies a possible formalism of the central idea (margin maximization) which has extension to the soft-margin optimization to become a more general learning approach, and the kernel-based extension, which is applicable to be capable of handling nonlinear relations. However, the basic concept of maximal margin classifiers can be easily formalized in a different way [112] or can be extended to regression problems [31] as well.

2.2 Fully Distributed Systems

In the later part of the thesis, we will move our system model from the traditional *random access stored program machine* (RASPM) [52]—where you have a single computational unit (CPU), with a central memory—to the so-called *fully distributed* environments.

These systems are networked environments which contain usually a large number of units, called *nodes*, that are capable of local computation and can communicate with other nodes using the network. We assume that the sys-

tem consists of a potentially very large number of nodes, typically personal computing devices such as PCs or mobile devices. We do not assume any homogeneity between the nodes. They can be different in their computational power, operating system, or any other characteristics. But we expect that most of them know and run the same protocol (excepting for a few number of malicious users, which usually can be tolerated in this type of systems[1]).

Each node in the system has a unique network address. We assume that a particular node knows the addresses of a constant number of other nodes, called *neighbors*. The number of neighbors is much less than the size of the network (due to memory limitations), but the set of neighbors can vary in time. If a node A knows the address of node B, we say that node B is *connected to* node A (connectivity relation).

The communication is done by message passing between connected nodes without any central control (fully distributed aspect). That is, every node can send messages to every other node, provided the address of the target node is available. We assume that messages can have arbitrary delays, and messages can be lost as well (failure scenarios). In addition, nodes can join and leave at any time without warning (churn), thus leaving nodes and crashed nodes are treated identically. Leaving nodes can join again, and while offline, they may retain their state information.

The graph defined by the computational units as nodes, and the connectivity relations as edges is called *overlay network*. The communication is completely based on this graph, since it is a central component of each fully distributed system. The overlay networks are provided by middleware services, called *peer sampling services* [64]. The peer sampling service is itself a distributed protocol. In the following, we briefly overview two widely used unstructured peer sampling protocols that are used intensively in the later chapters.

[1]This perspective of the fully distributed systems is not investigated here.

Algorithm 2.1 Skeleton of NEWSCAST and T-MAN protocols

```
1: n ← initNeighbors()
2: loop
3:     wait(Δ)
4:     p ← selectPeer(n)
5:     send (n ∪ currNode()) to p
6: end loop
7: procedure onReceiveNodes(nodes)
8:     n ← updateNeighbors(n ∪ nodes)
9: end procedure
```

2.2.1 Overlay Management

In Alg. 2.1, one can see the basic skeleton of both the NEWSCAST [64] and T-MAN [63] protocols. The algorithm handles a set of *node descriptors*, stored in variable n, which contains the addresses (and other node related data) of the nodes known by the current node. The connectivity relations (and so the overlay itself) is defined by these node descriptor sets. Each node in the network runs the same protocol. Through the run of the protocol, these sets are exchanged between the nodes to obtain the required overlay network based on local communication.

The algorithm is divided into two distinguished sections: the active behavior (left hand side code snippet) and an event handler function (right hand side code snippet; onReceiveNodes function). The active behavior consists of an initialization part and an infinite loop defining periodic activities. The node descriptors are initialized (at line 1 of Alg. 2.1) by applying a node cache or other bootstrap service. The main part of the active behavior consists of a sequence of periodic activities inside a loop executed in each Δ time moment (at line 3 of Alg. 2.1). In each active execution, the protocol selects a node stored in its own node descriptor set (at line 4 of Alg. 2.1, based on the abstract function SELECTPEER). Then, it sends its node descriptor set, extending with its own descriptor (proposed by the CURRNODE function), to the selected node (at line 5 of Alg. 2.1).

When a node descriptor set is received, the mechanism defined in the event handle function, called onReceiveNodes, is executed. The function takes the received node set and updates its own set by applying the abstract function UPDATENEIGHBORS (at line 8 of Alg. 2.1).

The proposed skeleton is abstract in the sense that the additional informa-

Algorithm 2.2 The NEWSCAST protocol

```
1: procedure SELECTPEER(nodes)
2:     node ← uniRand(nodes)
3:     return node
4: end procedure

5: procedure UPDATENEIGHBORS(nodes)
6:     nodes ← nodes − {currNode()}
7:     s ← sort nodes by field ts
8:     return topK(s)
9: end procedure

10: procedure CURRNODE()
11:     d ←initDesc()
12:     d.ts ← current time
13:     return d
14: end procedure
```

tion stored in the node descriptors is not mentioned[1], and it contains three functions that are not yet defined (however, their semantic roles are clearly discussed above).

The NEWSCAST [64] protocol is yielded if one implements the abstract functions as defined in Alg. 2.2. Here it is assumed that the node descriptors are extended with a timestamp field, denoted by *ts* in the pseudo code, containing the time moment when the node descriptor is created by the function CURRNODE() (at line 12 of Alg. 2.2). In the case of the NEWSCAST protocol, the update method selects the top-*k* most freshest node descriptors from the set of received and owned node descriptors providing a continuous sampling of the probably online nodes. The function SELECTPEER supports this behavior by providing uniform random selection from the available node descriptors. Thus, the NEWSCAST protocol results a dynamic overlay.

The selection of the top-*k* elements is necessary to avoid the continuous growing of the node sets (at line 8 of Alg. 2.2). This results that the overhead of the protocols consists of sending one message of a constant size to a node periodically. Here *k* is the parameter of the protocol.

It can be shown that, through the set of node descriptors proposed by the NEWSCAST protocol, each node can request uniform random samples of the nodes in the network that are likely to be online at the time of the request [64]. This protocol has been extended to deal with uneven request rates at different

[1]We assumed earlier only that the address of the node is stored; however, to achieve concrete algorithms, other information could be needed as well.

Algorithm 2.3 The T-MAN protocol

```
1: procedure SELECTPEER(nodes)
2:     node ← argmax_{o∈nodes}(s(currNode(), o))
3:     return node
4: end procedure
5: procedure UPDATENEIGHBORS(nodes)
6:     (nodes ← nodes ∪ {r random nodes})
7:     nodes ← nodes − {currNode()}
8:     s ← sort nodes based on s (currNode(), .)
9:     return topK(s)
10: end procedure
11: procedure CURRNODE()
12:     d ←initDesc()
13:     fillCharacteristics(d)
14:     return d
15: end procedure
```

nodes, as well as uneven distributions of message drop probabilities [116].

For discussing the T-MAN [63] protocol, let us assume that the node descriptors contain the characteristics of the users[1] that use them, and a $s : \textit{node descriptor} \times \textit{node descriptor} \to \mathbb{R}$ similarity measure is given which can measure some kind of higher order similarity (e.g. similarity in taste or behavior of users) between two users based on the information that can be found in the node descriptors.

We can obtain the protocol T-MAN by defining the abstract functions of Alg. 2.1 in the way given in Alg. 2.3. The active behavior of the protocol selects the most similar node (at line 2 of Alg. 2.3) to the current node from the set of node descriptors available locally (through the SELECTPEER implementation). Through the update, the protocol retains the most similar k nodes to the current node by sorting the incoming nodes based on the function $s(\text{currNode}(), .)$ (at line 8 of Alg. 2.3) varying only in its second argument (as a partially called function, that is, function of one variable). The function CURRNODE is responsible for creating the node descriptor of the current node. Here the node related characteristics are loaded by the function call FILLCHARACTERISTICS at line 13 of Alg. 2.3. The details of this function are application dependent, hence we do not discuss it here.

Applying the above mentioned mechanism, this protocol converges to the

[1] We could say feature vectors of the users if we want to apply the terminology of supervised learning mentioned in Sec. 2.1.

similarity overlay of the given measure. That is, an overlay which contains edges to the most similar nodes based on the similarity function s. The speed of the convergence is pretty fast; it is $O(\log(n))$ per node [63]. Considering that the nodes of the network are individual computation units, i.e. this is an asynchronous parallel network, this is a very efficient solution.

The protocol is often extended with an optional modification shown at line 6 of Alg. 2.3. Here we add r random node descriptors to the original set making the protocol more explorative. The samples can be provided by e.g. a NEWSCAST protocol instance which runs on each node separately from the T-MAN protocol. Here r is another parameter of the protocol.

Numerous other overlay management protocols exist. Here we just highlighted two of them that are used intensively in the later chapters. For a more detailed description of the above discussed protocols, please read [63, 64].

CHAPTER 3

VMLS-SVM for Time Series Forecasting

In the automated data processing the *time dimension* appears very often. Basically each post that we share in the social network sites, each financial transaction that we execute during each payment, and any activities that are performed on the web are stored together with the timestamp when the data was produced. Later this huge amount of time related data is used by data-driven companies to optimize their future revenue by making better decisions related to their business model [45].

The processing of this type of data quite often involves the problem of time series forecasting. Here we want to build statistical models based on the currently available data which make accurate predictions on the future. The applications of these models are almost as diverse as the variety of the related datasources mentioned above. You can easily find applications in finance where the goal is to predict the changes of the markets, and entertainment where we want to know e.g. how our social network will look like in the near future [66].

However time series forecasting problems are quite common, the approaches that deal with it usually come from the field of regression. These models—however they perform very well in general regression tasks—often lack

of those design principles that make really accurate in these type of tasks.

For these reasons, in this chapter we propose a novel extension to the Least Squares Support Vector Machines (LS-SVM) [112, 113]. This extension, called Variance Minimization LS-SVM (VMLS-SVM) [89], refines the objective function of the original LS-SVM by introducing another parameter. The modification is based on the preliminary observation that underlines the importance of the variance term of the error function. We describe the theoretical details behind the extension, and through a series of empirical evaluation we show that it has a crucial role in solving time series forecasting tasks. We briefly discuss the effect of the new parameter, and show how you can fine-tune it to avoid overfitting.

The results of this chapter are based on our recent work published in [89].

3.1 Related Work and Background

Times series can be modelled by a large class of models which can be roughly divided into two main subclasses, namely linear models and nonlinear ones. The linear models (such as autoregressive (AR) models [19, 70], linear regression [29]) have been extensively investigated and are quite advanced, but they only work well when the data dependency is closely linear.

The neural network based regression models can overcome these weaknesses i.e. they can model nonlinear relations. Both the Artificial Neural Networks (ANN) [42] and the Radial Basis Function Neural Networks (RBF) [104] are commonly used in time series forecasting. Although they have some inherent drawbacks, such as the problem of multiple local minima, the choice of number of hidden units, and the danger of overfitting. These problems often make the use of neural network based approaches a bit difficult.

The application of SVMs in regression setting [31] (or SVRs as often referred) can avoid these problems by having unique and globally optimal solution [31] and only one hyperparameter ($C \in \mathbb{R}$, the margin maximization trade-off, for more details see Sec. 2.1.1, particularly e.g. Eq. 2.6). Additionally, they usually achieve a better generalization by applying the margin maximization technique than the traditional neural networks. Another advantage of the

SVM approach is the applicability of the *kernel-trick* which makes it more practicable. A slight drawback of training a classical formalized SVM is that it solves a linearly constrained QP problem in the size of training examples, n, which can be huge.

LS-SVM does not have this latter problem. This approach is a reformulation of the principles of SVM, which applies equality instead of inequality constraints. Based on this, its solution follows from a linear KKT [112] system instead of a computationally hard QP Problem [24, 119].

To give a little insight to the main concept of the LS-SVM formalism, suppose that we are in a supervised regression setting and given a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \mathbb{R}$. With LS-SVM, one considers the following optimization problem (primal form):

$$\begin{aligned} \text{minimize}_{\mathbf{w},b,\mathbf{e}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\frac{1}{2}\sum_{i=1}^{n} e_i^2 \\ \text{subject to} \quad & y_i = \mathbf{w}^T\phi(\mathbf{x}_i) + b + e_i \quad (\text{for each } 1 \leq i \leq n), \end{aligned} \tag{3.1}$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^h$ is the feature mapping, C is a trade-off parameter between generalization and the error minimization, and $e_i \in \mathbb{R}$ $(0 \leq i \leq n)$ are the error variables (their roles are similar to the slack variables in the case of classic SVM formalism detailed in Sec. 2.1.1).

In the case of LS-SVM the model is given by:

$$f(x) = \mathbf{w}^{\star T}\phi(\mathbf{x}) + b^\star, \tag{3.2}$$

where $\mathbf{w}^\star \in \mathbb{R}^h$ and $b^\star \in \mathbb{R}$ are the solutions of the optimization problem defined in Eq. 3.1.

The result of the optimization problem is obtained by taking the corresponding Lagrange function, differentiating it, and establishing stationarity constraints, which results the following linear equation system:

$$\left(\begin{array}{c|c} 0 & \mathbf{1}^T \\ \hline \mathbf{1} & \Omega + \frac{1}{C}\mathrm{I} \end{array}\right)\left(\begin{array}{c} b \\ \hline \boldsymbol{\alpha} \end{array}\right) = \left(\begin{array}{c} 0 \\ \hline \mathbf{y} \end{array}\right), \tag{3.3}$$

where $\mathbf{y} = (y_1, \ldots, y_n)^T$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^T$, $\mathbf{1}^T = (1, \ldots, 1) \in \mathbb{R}^{1\times n}$ and $\Omega \in$

$\mathbb{R}^{n\times n}$ where $\Omega_{i,j} = \phi(x_i)^T\phi(x_j) = K(x_i, x_j)$ (for each $1 \leq i, j \leq n$). From the result of the dual problem the optimal solution of the original problem can be computed by $\mathbf{w}^\star = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)$.

This formulation is quite simple and—as one can see above—the method has all of the advantages of SVM, like the applicability of the kernel-trick and it has a unique solution. Moreover, in the case of LS-SVM, the solution comes from solving a linear system of equations, not a quadratic one, which results in an increased efficiency in the computation of the model. But LS-SVM has also one slight drawback. While the classic SVM chooses some samples from the training data (the support vectors) to represent the model (that is, it has a sparse solution shown in Eq. 2.9), the LS-SVM uses all the training data to produce the result. Sparseness can also be introduced with LS-SVM by applying a pruning based method to select the most important examples [37, 74].

The model is quite general in the sense that it can also be shown to have a connection with regularization networks [41]. When no bias term is used in the LS-SVM formulation, as proposed in Kernel Ridge regression [107], the expression in the dual form corresponds to the Gaussian Processes [123].

3.2 VMLS-SVM

Essentially VMLS-SVM is an extension of LS-SVM, where the objective function of optimization is refined by adding a weighted variance minimization part. The motivation behind this modification is a preliminary observation which can be explained as follows: if two time series forecasting models are given with the same error, usually the better is the one that has a smaller variance, i.e. the one that can produce a smoother fitting. Applying the proposed modification, we can adjust the weight of the variance term of the error function as well with an additional hyperparameter.

Now let us describe this method in detail. Suppose that we have a training set $\mathcal{D}$, like that defined in the previous subsection. Next, let us express the

optimization problem of VMLS-SVM in the following way:

$$\begin{aligned} \text{minimize}_{\mathbf{w},b,\mathbf{e}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\frac{1}{2}\sum_{i=1}^{n} e_i^2 + D\frac{1}{2}\sum_{i=1}^{n}\left(e_i - \frac{1}{n}\sum_{j=1}^{n} e_j\right)^2 \\ \text{subject to} \quad & y_i = \mathbf{w}^T\phi(\mathbf{x}_i) + b + e_i \quad (\text{for each } 1 \le i \le n), \end{aligned} \tag{3.4}$$

where ϕ, C and e_i ($1 \le i \le n$) are the same as those of LS-SVM (see Eq. 3.1) and the D parameter is a trade-off between generalization and variance minimization. The regression function is the same as before, hence it is defined by Eq. 3.2.

As in the case of LS-SVM, the solution of Eq. 3.4 is obtained from the corresponding dual problem:

$$\begin{aligned} \mathbf{L}(\mathbf{w},b,\mathbf{e},\boldsymbol{\alpha}) = & \frac{1}{2}\|\mathbf{w}\|^2 + C\frac{1}{2}\sum_{i=1}^{n} e_i^2 + D\frac{1}{2}\sum_{i=1}^{n}\left(e_i - \frac{1}{n}\sum_{j=1}^{n} e_j\right)^2 - \\ & \sum_{i=1}^{n}\alpha_i\left(\mathbf{w}^T\phi(\mathbf{x}_i) + b + e_i - y_i\right). \end{aligned} \tag{3.5}$$

The optimal solution of this function can be obtained by demanding that the following conditions be satisfied:

$$\frac{\partial \mathbf{L}(\mathbf{w},b,\mathbf{e},\boldsymbol{\alpha})}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{i=1}^{n}\alpha_i\phi(\mathbf{x}_i), \tag{3.6}$$

$$\frac{\partial \mathbf{L}(\mathbf{w},b,\mathbf{e},\boldsymbol{\alpha})}{\partial b} = 0 \rightarrow \sum_{i=1}^{n}\alpha_i = 0, \tag{3.7}$$

$$\frac{\partial \mathbf{L}(\mathbf{w},b,\mathbf{e},\boldsymbol{\alpha})}{\partial e_i} = 0 \rightarrow \alpha_i = (C+D)\,e_i - \frac{D}{n}\sum_{j=1}^{n} e_j \quad \text{for each } 1 \le i \le n, \tag{3.8}$$

$$\frac{\partial \mathbf{L}(\mathbf{w},b,\mathbf{e},\boldsymbol{\alpha})}{\partial \alpha_j} = 0 \rightarrow \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_j) + b + e_j = y_j \quad \text{for each } 1 \le j \le n. \tag{3.9}$$

From Eq. 3.7 and 3.8 we get the following:

$$0 = \sum_{i=1}^{n}\alpha_i = (C+D)\sum_{i=1}^{n} e_i - D\sum_{j=1}^{n} e_j = C\sum_{i=1}^{n} e_i. \tag{3.10}$$

This constraint is encoded in the first equation of the final equation system presented in Eq. 3.13. The additional constraints can be observed by performing the following. For each $1 \leq j \leq n$, replacing $\mathbf{w}$ in Eq. 3.9 with the right hand side of Eq. 3.6, and then replacing α_i with the right hand side of Eq. 3.8. After this manipulation, we get:

$$\sum_{i=1}^{n} \left((C+D)\, e_i - \frac{D}{n} \sum_{k=1}^{n} e_k \right) K(x_i, x_j) + b + e_j = y_j. \tag{3.11}$$

With some additional algebraic manipulation, we arrive at the following:

$$\sum_{i=1}^{n} \underbrace{\left[(C+D)\, K(x_j, x_i) - \frac{D}{n} \sum_{k=1}^{n} K(x_k, x_j) \right]}_{\Omega_{j,i}} e_i + b + e_j = y_j. \tag{3.12}$$

This constraint is encoded in the jth equation (row) of the final equation system shown below in Eq. 3.13.

Using Eq. 3.10 and Eqs. 3.12 (for each $1 \leq j \leq n$) we get the following equation system (these equations do not contain the variables $\mathbf{w}$ and α since they were eliminated):

$$\left(\begin{array}{c|c} 0 & \mathbf{C}^{\mathrm{T}} \\ \hline \mathbf{1} & \Omega + \mathrm{I} \end{array} \right) \left(\frac{b}{\mathbf{e}} \right) = \left(\frac{0}{\mathbf{y}} \right), \tag{3.13}$$

where $\mathbf{1}$, $\mathbf{y}$ are the same as in Eq. 3.3, $\mathbf{C}^{\mathrm{T}} = (C, \ldots, C) \in \mathbb{R}^{1 \times n}$, and $\Omega \in \mathbb{R}^{n \times n}$ where $\Omega_{j,i}$ is defined in Eq. 3.12. By solving this equation system, we obtain the optimal solution of the original optimization problem defined in Eq. 3.4.

Now the regression function becomes

$$f(x) = \sum_{i=1}^{n} \left[(C+D)\, e_i^\star - \frac{D}{n} \sum_{j=1}^{n} e_j^\star \right] K(x_i, x) + b^\star, \tag{3.14}$$

where $\mathbf{e}^\star \in \mathbb{R}^n$, and $b^\star \in \mathbb{R}$ are the solutions of the equation system shown in Eq. 3.13.

3.2.1 How Our Method Is Applied to Time Series Forecasting

In the above subsection we introduced the details of VMLS-SVM as a regression method. Let us now turn to discuss how we applied this method for modelling time series. A time series is a sequence of vectors, $\mathbf{x}(t)$, $t = 0, 1, \ldots$ where t represents the elapsed time. For the sake of simplicity, we will consider only sequences of scalars here, although each technique can be readily generalised to vector series.

For us, time series forecasting means predicting the value of a variable from a series of observations of that variable up to that time. Usually we have to forecast the value of $x(t+k)$ from the following series of observations: $x(t), \ldots, x(t-N+1)$. Formally this can be stated as: find a function $f_k : \mathbb{R}^N \to \mathbb{R}$ to obtain an estimate of x at time $t+k$, from the N time steps back from time t. Hence for each time moment t

$$x(t+k) = f_k(x(t), \ldots, x(t-N+1)). \tag{3.15}$$

The observations $(x(t), \ldots, x(t-N+1))$ constitute a *window* and N is referred to as the *window size*. The technique which produces all the window-value pairs as training samples for a regression method is called the *sliding window technique*. This technique slides a window of length $N+1$ over the full time series and generates the training samples in the following form for every possible value of i:

$$(\mathbf{x}_i, y_i) = (x(t+i-N+1), \ldots, x(t+i), x(t+i+k)), \tag{3.16}$$

where $N - t - 1 \leq i$, $\mathbf{x}_i \in \mathbb{R}^N$, and $y_i \in \mathbb{R}$. In this way the size of the window (N) determines the dimension of the regression problem (d).

3.3 Experimental Results

We performed experimental evaluation on several benchmark databases in different settings. Before we turn to present our results, we summarize shortly how we performed these evaluations.

In order to perform time series forecasting using the VMLS-SVM method, we employ the sliding window technique described in the previous subsection which produces an initial training database. Afterwards, our system applies a normalization on the input databases.

In our tests, we experimented with three types of kernels for the LS-SVM and VMLS-SVM. These were the following: Polynomial kernel, RBF kernel, Alignment kernel for time series [33]. The first two kernels are general purpose, well-known kernels. The last one is especially designed to apply on sequential data like time series. It applies a Dynamic Time Warping based alignment of the samples.

3.3.1 Parameter Selection

As one can see, the VMLS-SVM has two hyperparameters, which makes it more complicated to fine-tune the method and avoid overfitting. Here we describe shortly how we set these parameters.

In our first tests on the first two benchmarks, we applied a simulated annealing based [71] optimization method, called parameter optimizer, which optimized the parameters of the underlying learning method. This parameter selection method can be viewed as a function minimizing method, where the input of objective function is the parameter of the underlying learner. The value of the function is the aggregated error of the underlying method on a fixed validation set. Of course, the underlying regression method was trained on a different, but also fixed training set using the input of the objective function as parameters. The applied error measure depends on the database evaluation metric, i.e. it is always the same as the error measure of the evaluation process. The stopping criteria of the parameter selection method was determined by visual inspection, i.e. when the error was quite small and it did not decrease, we stopped the process.

Using this optimization technique, we get a sequence of parameter sets, which was provided by the parameter optimization method. This revealed a trend of a correct parameter setup. Afterwards, we carried out some manually parameterized tests, using experiments from the automatic parameter selec-

tion. These tests were evaluated on the evaluation set, which is completely different from the training and validation sets.

3.3.2 Results on CATS Benchmark

The CATS benchmark [40] is a publicly available database for time series forecasting. It was made for a competition organized during the IJCNN'04 conference in Budapest. This artificial time series has 5,000 data points, among which 100 are missing. The missing values are divided into 5 blocks, elements between 981-1,000, 1,981-2,000, 2,981-3,000, 3,981-4,000, and 4,981- 5,000.

The goal of the competition was to predict these 100 missing values. Twenty-four teams participated and seventeen of them uploaded acceptable predictions. The evaluation metric was the mean squared error (MSE) metrics:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.17}$$

computed on the 100 missing values as evaluation set.

In order to make a comparison, we decided to use the LS-SVM and our VMLS-SVM in the same way. We defined twenty different prediction functions based on Eq. 3.15. In this setup the first prediction function (f_1) was used to predict the first missing value, the second prediction function (f_2) was used to predict the second missing value, and so on. Each prediction function used the same parameter setting, which was provided[1] by the parameter optimization method in the early tests. This optimization process used a validation set to test the goodness of different parameter setups. This validation set was generated in the following way: from the end of each training block we split the last 20 values. Afterwards, using the experiments of the automatic parameter optimization, we carried out some tests, using the fixed parameterized prediction method. In this phase, for each prediction function we trained its own LS-SVM or VMLS-SVM learner with the same parameter values on the full training set. The overall results are presented in Table 3.1.

[1]Here we changed just the values of D, C and the window size.

Table 3.1: MSEs of forecasting of various configurations of the LS-SVM and VMLS-SVM methods on the 100 missing values.

Method	Value of D	Value of C	Kernel	WS	MSE
LS-SVM	-	1,000	Poly, p=12	60	534.33
LS-SVM	-	1,000	RBF, γ=0.001	60	412.27
LS-SVM	-	1,000	Align, σ=12	60	432.66
VMLS-SVM	100	0.1	Poly, p=12	60	501.25
VMLS-SVM	100	0.1	RBF, γ=0.001	60	421.34
VMLS-SVM	100	0.1	Align, σ=12	60	437.45
VMLS-SVM	1,000	0.01	Poly, p=12	80	480.11
VMLS-SVM	1,000	0.01	RBF, γ=0.001	80	**287.36**
VMLS-SVM	1,000	0.01	Align, σ=12	80	**312.28**
VMLS-SVM	10,000	0.01	Poly, p=12	80	498.62
VMLS-SVM	10,000	0.01	RBF, γ=0.001	80	386.25
VMLS-SVM	10,000	0.01	Align, σ=12	80	408.17

In Table 3.1, "Poly" means Polynomial kernel and p denotes its parameter, the exponent. "RBF" means the RBF kernel and γ denotes its parameter, the gamma parameter. Similarly, "Align" means Alignment kernel and its parameter is denoted by σ. The notation WS means window size.

As can be seen in Table 3.1, the VMLS-SVM methods with the RBF kernel are the most effective for predicting the 100 missing values, and VMLS-SVM models are consequently more accurate than LS-SVM models with a similar kernel function.

The weighting of the variance minimizing term helps the VMLS-SVM to achieve a better performance on the evaluation set, but where the weight exceeded a threshold, the error on the evaluation set starts to increase. It means that the overweighting of the variance term caused overfitting. But in this case the error on the validation set starts to increase, while the training error decreases. Hence using the parameter selection technique defined above, we can choose a better generalization model, which is about $D = 1,000$, $C = 0.01$ and $\text{WS} = 80$.

Table 3.2 shows a comparison between our method (labeled as VMLS-SVM) and the best results reported for this series in the competition. Table 3.2 con-

Table 3.2: Comparison between the main methods on CATS benchmark.

Method	MSE
VMLS-SVM (RBF)	**287**
VMLS-SVM (Align)	**312**
Kalman Smoother	408
Recurrent Neural Networks	441
Competitive Associative Net	502
Weighted Bidirectional Multi-stream Extended Kalman Filter	530
SVCA Model	577
MultiGrid -Based Fuzzy Systems	644
Double Quantization Forecasting Method	653
Time -reversal Symmetry Method	660
BYY Harmony Learning Based Mixture of Experts Model	676
Ensemble Models	725
Chaotic Neural Networks	928
Evolvable Block-based Neural Networks	954
Time -line Hidden Markov Experts	103
Fuzzy Inductive Reasoning	1050
Business Forecasting Approach to Multilayer Perceptron Modelling	1156
A hierarchical Bayesian Learning Scheme for Autoregressive Neural Networks	1247
Hybrid Predictor	1425

tains just the best results of our models, using $D = 1,000$, $C = 0.01$ and WS $= 80$ parameter values.

3.3.3 Results on the Dataset of NN3 Neural Forecasting Competition

To further investigate the capabilities of VMLS-SVM, we made a comparison with ϵ-insensitive SVM regression method. Hence we applied our method to forecast the reduced subset of time series of the NN3 Artificial Neural Network and Computational Intelligence Forecasting Competition, which has reported results using an ϵ-insensitive SVM regression approach [32].

The NN3 dataset contains 11 time series. The averaged length of the time

Table 3.3: SMAPE of each time series using the VMLS-SVM and ϵ-insensitive SVM regression methods.

Time series	Length	SMAPE(VMLS-SVM)	SMAPE(ϵ-insensitive SVR)
1	126	0.5218	0.49
2	126	**1.7432**	2.5
3	126	**8.7458**	12.46
4	115	**4.6944**	5.4
5	126	**0.3091**	0.7
6	126	**1.8892**	3.38
7	126	**0.9871**	1.21
8	115	**8.1293**	9.35
9	123	**1.1947**	2.54
10	126	**8.4968**	12.7
11	126	**3.7217**	4.82

series is 124[1], and there is no domain knowledge about the time series. In [32], each the last 18 values of each time series were predicted. The evaluation metric was the symmetric mean absolute percent error (SMAPE) [8, 60]:

$$\text{SMAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{(y_i + \hat{y}_i)\,/2} \cdot 100, \tag{3.18}$$

where n is the number of prediction, $\hat{y}_i$ is the ith prediction and y_i is the ith expected value from the evaluation set.

For each time series, we defined 18 different prediction functions based on Eq. 3.15. Each prediction function used the same parameter setting, but each times series used different parameter setting, which were determined by parameter optimization. To carry out the parameter optimization, we made three subsets from each time series e.g. the training set, the validation set and the evaluation set. Each evaluation set contains the last 18 values from the corresponding time series, and each validation set is made up from the last 18 values of the corresponding time series without the evaluation set. For each set triplet, we made a hyper-parameter optimization for the VMLS-SVM based prediction functions, using the training and validation sets. We adjusted only

[1]The length of each time series is available in Table 3.3.

the D and C parameters during the optimization and used a fixed window size WS $= 12$, and RBF kernel with parameter $\gamma = 0.001$. These optimizations determined well-generalization parameter sets for each VMLS-SVM predictor. Afterwards, we used the correct parameter setups; we trained each VMLS-SVM prediction function on the training and validation sets and evaluated them on the evaluation set as test set. Table 3.3 shows a comparison between our method and the results reported for this series in [32].

The mean SMAPE of our method is 3.6757 and the same value of the ϵ-insensitive SVM regression method is 5.05. Hence we can assess that our method using VMLS-SVM methods and parameter optimization achieves a significantly higher mean performance. As can be seen, our method achieves lower SMAPE on each time series except the first one.

3.3.4 Results on the Santa Fe Data Set D

Here we present the results of our method on a widely-recognized benchmark, the D data series from the Santa Fe competition [121]. The data set consists of artificial data generated from a nine-dimensional periodically driven dissipative dynamical system with an asymmetrical four-well potential and a drift on the parameters.

This database has a relatively high dimension, it is large (100,000 data points) and highly dynamic. Moreover, we have no background information about it.

Our evaluation is measured by the Root Mean Squared Error (RMSE) metric:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{3.19}$$

where the notation is similar to the one was used in Eq. 3.17, and we predict 25 missing values.

This data set is quite large, which implies that it is a computationally intensive learning task. Hence we trained our system with only one configuration: We used only the last 20,000 training samples applying the RBF kernel with γ=0.001. The window size was 80, the value of parameter D was 1,000, and the value of parameter C was 0.01. These parameters were determined using

Table 3.4: Comparison between the methods on Santa Fe D data set.

Method	RMSE
VMLS-SVM (RBF)	**0.0628**
SVM (ϵ-insensitive)	0.0639
SVM (Huber)	0.0653
RBF network	0.0677
ZH [126]	0.0665

our experiments with the earlier tested benchmarks.

Table 3.4 shows the results of our method relative to the methods proposed by Müller et al. [87]. The presented results show that our methods can achieve a better performance than others in this task as well.

3.4 Conclusions

In this chapter we presented a novel extension for LS-SVM based on weighting the variance of error. After presenting the basic theory of the method, we evaluated it on publicly available benchmarks. These experiments show that the proposed method can indeed achieve a higher efficiency on the three different, widely-recognized benchmarks than the standard LS-SVM or other similar methods. The nice results we obtained can probably be attributed to the beneficial features of our novel method, since the data sets we used for experiments are typical and widely-used benchmarks for testing machine learning algorithms for time series prediction.

Moreover, we showed that with a carefully designed extension a basic SVM approach (LS-SVM) can be successfully adapted to a specific task (time series forecasting) resulting a much suitable approach. Additionally, one can see that formalizing a preliminary observation into the SVM-based framework is straightforward. We believe that the proposed method—and additionally how we obtained—confirms the robustness of the SVM-based learning.

This chapter is based on the results published previously in [89].

CHAPTER 4

Transformation-based Domain Adaptation

The generalization properties of most statistical machine learning approaches are based on the assumption that the samples of the training dataset come from the same underlying probability distribution than those that are used in the prediction phase of the model. Unfortunately—mainly in real-world applications—this assumption often fails. There are numerous Natural Language Processing (NLP) tasks where plentiful labeled training databases are available from a certain domain (source domain), but we have to solve the same task using data taken from a different domain (target domain) where we have only a small dataset. Manually labeling the data in the target domain is costly and inefficient. However, if an accurate statistical model from the source domain is presented, we can adapt it to the target domain [36]. This process is called *domain adaptation*.

Opinion mining aims at automatically extracting emotional cues from texts [72]. For instance, it can classify product reviews according to the customers positive or negative polarity. This is a typical problem where the requirement for domain adaptation is straightforward as there exists numerous slightly different domains (e.g. different products are different domains), and the con-

struction of manually labeled training data for each of them would be costly.

Here, we will define a general framework to directly capture the relations between domains. In order to experimentally evaluate our approach, SVM and Logistic Regression [15] were plugged into the framework and the approach was compared to a number of baseline algorithms and published results on opinion mining datasets. We will show that SVM is a more suitable approach to apply in the proposed framework.

The results presented here are mainly based on our earlier work [91].

4.1 Related Work

Numerous preliminary algorithms have been developed in the field of domain adaptation, which roughly can be categorized into two mainstreams.

One of these types of methods tries to *model the differences between the distributions* of the source and target domains empirically. In [28] the parameters of a maximum entropy model are learnt from the source domain, while an adapted Gaussian prior was used during training a new model on target data. A different technique proposed in [36] defines a general domain distribution that is shared between source and target domains. In this way, each source (target) example can be considered a mixture of source (target) and general distributions. Using these assumptions, their method was based on a maximum entropy model and used the EM algorithm for training. Another approach was proposed in [35], where a heuristic nonlinear mapping function was used to map the data into a higher dimensional feature space where a standard supervised learner could be applied in order to perform the domain adaptation.

Another generation of domain adaptation algorithms are based on *defining new features for capturing the correspondence* between source and target domains [12, 47]. In this way, the two domains appear to have very similar distributions, which enables effective domain adaptation. In [17], the authors proposed a method—called Structural Correspondence Learning (SCL) algorithm—which depends on the selection of pivot features that appear frequently in both the source and the target domains. Although it was experimentally shown that SCL can reduce the difference between domains based on a cer-

tain distance measure [12], this type of feature selection may be sensitive to different applications.

A more specific subtype of the above described algorithm family learns a *joint feature representation* for the source and the target domain where the corresponding marginal distributions are close to each other [96]. In [95], the authors proposed a new dimensionality reduction algorithm, the Maximum Mean Discrepancy Embedding (MMDE), for domain adaptation tasks. It aims at learning a shared latent space underlying the source and target domains where the distance between the distributions can be reduced. A feature space transformation based method is described in [96]. They learnt transformations for both the source and the target feature spaces that transform the data into a common Reproducing Kernel Hilbert Space (RKHS) using the key approach of MMDE.

Theoretical results on domain adaptation have been also proposed [79, 80, 81]. For instance in [80], the problem of multiple source domain adaptation is considered and theoretical results of the expected loss of a combined hypotheses space were presented on the target domain.

4.2 Transformation-based Domain Adaptation Approach

As we mentioned earlier, the principal goal of domain adaptation is finding out how we can best use a trained machine learning model in the learning of a different database. In this section, we will give a more precise formalism of the domain adaptation task, and we will describe our approach in detail.

4.2.1 Domain Adaptation Task

In the current context of domain adaptation, we will assume that there are two feature spaces given—$\mathcal{T}_S$ and $\mathcal{T}_T$[1]—the feature spaces of "source domain" and

[1]Here $\mathcal{T}$ refers to the task, while S and T refer to the term "source" and "target", respectively.

the "target domain", respectively. We have two sets of labeled training samples (supervised learning setting like the one introduced in Sec. 2.1), $\mathcal{D}_S \subseteq \mathcal{T}_S$ and $\mathcal{D}_T \subseteq \mathcal{T}_T$ ($|\mathcal{D}_T| \ll |\mathcal{D}_S|$). In addition, we will assume that both the source domain and the target domain use the same label set. We will call $\mathcal{D}_S$ the "training database of the source domain" and $\mathcal{D}_T$ the "training database of the target domain". The labels[2] in both domains come from the $\mathcal{L} = \{C_1, \ldots, C_l\}$ set and $y_{i,S}$ ($y_{i,T}$) denotes the *correct* class label of sample i from $\mathcal{D}_S$ ($\mathcal{D}_T$). The learning problem of the domain adaptation task is to find a $f_T : \mathcal{T}_T \to \mathcal{L}$ prediction function that achieves a high accuracy on the target domain. During the training, we are allowed to use a trained and accurate source model $f_S : \mathcal{T}_S \to \mathcal{L}$, and a small training database $\mathcal{D}_T$ from the target domain, which excludes the learning of an accurate statistical model applying purely this database from the target domain.

4.2.2 Transformation-based Approach

One of the main assumptions of the domain adaptation task is that there exists some kind of relation between the source domain and the target domain. Our idea is to try to model this relation, i.e. try to find a $\phi : \mathcal{T}_T \to \mathcal{T}_S$ transformation or target-source domain mapping. This transformation maps the samples from $\mathcal{T}_T$ into the feature space of $\mathcal{T}_S$. In the feature space of the source domain, we have a f_S prediction function, which achieves a high accuracy, so if we can find the ϕ, which transforms the samples from the target domain into the source domain in such a way that the prediction error of the transformed samples using the fixed f_S is small, we can solve the task of domain adaptation.

More precisely, we look for a $\phi : \mathcal{T}_T \to \mathcal{T}_S$ transformation which minimizes the prediction error of each transformed sample taken from the training database of the target domain. Our idea is to utilize the $f_S : \mathcal{T}_S \to \mathcal{L}$ model (a prediction function on the source domain with a high prediction accuracy) directly for this task. Hence the following optimization problem was formed:

[2]Our approach will focus on classification problems, but it can easily be extended to regression problems as well.

$$\text{minimize}_{\phi} \quad E_{\mathcal{D}_T, f_S}(\phi) + C \sum_{\mathbf{x} \in \mathcal{D}_T} \|\phi(\mathbf{x})\| \,. \tag{4.1}$$

Here the $\mathcal{D}_T$ and f_S are the same as those defined in the previous subsection (Sec. 4.2.1), ϕ is the object of the optimization, the target-source domain mapping. The $E_{\mathcal{D}_T, f_S}(\phi)$ is a general error function which just depends on ϕ and it will be defined later. We see that the optimization does not optimize the f_S model; it is assumed to be constant during the optimization. The first term of the above optimization ensures that we get a transformed sample set which has a small prediction error (i.e. a high accuracy) in the source domain using the f_S prediction function and transformation ϕ. The second term is a weighted regularization term, where C ($C \in \mathbb{R}$ and $0 \leq C$) is the weight of the regularization part.

If we can solve this optimization problem, we will get the prediction function of the target domain in the following form

$$f_T(\mathbf{x}) = f_S(\phi^{\star}(\mathbf{x})). \tag{4.2}$$

Here the $\phi^{\star} : \mathcal{T}_T \to \mathcal{T}_S$ mapping is the transformation which is the solution of the above-defined minimization task (Eq. 4.1) and $\mathbf{x} \in \mathcal{T}_T$ is an arbitrary sample from the target domain.

4.2.3 Transformation-based Approach Based on Linear Transformation

We need to apply some constraints on the ϕ target-source domain mapping when solving the defined optimization task. Our proposal is to use a mapping which is as simple as possible, but not trivial. Thus in this chapter, we will apply the following constraints on target-source domain mapping, and on the

two domains:

$$\begin{aligned} &\mathcal{T}_S = \mathbb{R}^n, \\ &\mathcal{T}_T = \mathbb{R}^m, \\ &\phi : \mathbb{R}^m \to \mathbb{R}^n, \\ &\mathbf{x} \mapsto W\mathbf{x} \in \mathbb{R}^n \text{ (for each } \mathbf{x} \in \mathbb{R}^m), \end{aligned} \tag{4.3}$$

where $W \in \mathrm{R}^{n \times m}$. With these constraints, we will get the following specialized optimization task:

$$\text{minimize}_{W,\, \|W\|=1} \quad E_{\mathcal{D}_T, f_S}(W). \tag{4.4}$$

Here the regularization term is not necessary since it is replaced by the $\|W\| = 1$ constraint on the transformation matrix. This modification can be interpreted as the regularization term in the original form without weighting.

To solve Eq. 4.4 optimization, we propose a simple gradient descent [110] based optimization algorithm. This algorithm template is shown in Alg. 4.4. To get a concrete algorithm, we only need to specify a $E_{\mathcal{D}_T, f_S}(W)$ error function, and we have to compute its gradient ($\nabla E_{\mathcal{D}_T, f_S}(W)$). In order to compute the gradient of the error function ($\nabla E_{\mathcal{D}_T, f_S}(W)$), we have to derive the f_S source model as well, which has not been defined until now.

In Alg. 4.4, we show the iterative method which was applied in our experiments (in Sec. 4.3). Within the algorithm, we implemented a simple mechanism which tries to avoid that the learner gets stuck in local optima. The evaluation, presented in Sec. 4.3, shows that this simple technique is usually enough to get a suitable adaptation result.

In the next two subsections, we present in detail two particular algorithm instances of Alg. 4.4 template. The first algorithm instance applies the well-studied SVM [24] as source model, and the second one uses the Logistic Regression [59] as source domain classifier.

4.2.4 Support Vector Machine as Source Model

First the widely used binary SVM [31] classification method is introduced as the base prediction method to instantiate Alg. 4.4, and the following error func-

Algorithm 4.4 Domain Mapping Learner (DML)

Input: $\mathcal{D}_T$, f_S
Initialize
$i = 0$,
$W^{(0)} = randomMatrix(n, m)$,
$r = 0$, $max\_r = 10$
repeat
$W^{(i+1)} = W^{(i)} - \eta_i \nabla E_{\mathcal{D}_T, f_S}\left(W^{(i)}\right)$
$i = i + 1$
$W^{(i)} = \frac{W^{(i)}}{\|W^{(i)}\|}$
if $E_{\mathcal{D}_T, f_S}\left(W^{(i)}\right)$ not changed for a while and $r < max\_r$ **then**
$r = r + 1$
$i = 0$
$W^{(0)} = randomMatrix(n, m)$
end if
until not converged

tion is applied:

$$E_{\mathcal{D}_T, f_S}(W) = \frac{1}{2} \sum_{(\mathbf{x}, y) \in \mathcal{D}_T} (y - f_S(W\mathbf{x}))^2. \tag{4.5}$$

The choice of binary SVM as source model determines that we will consider only binary classification problems here. We assume that both the source domain and the target domain are labeled with the following labels: $\mathcal{L} = \{-1, +1\}$ (binary classification as introduced in Sec. 2.1). In this case, the prediction function of the SVM classifier (similarly to the one was presented in Eq. 2.10) in our formalism is:

$$f_S(W\mathbf{x}) = \sum_{(\mathbf{x}_i, y_i) \in SV_S} \alpha_i^\star y_i K(\mathbf{x}_i, W\mathbf{x}) + b^\star. \tag{4.6}$$

Here $SV_S \subseteq \mathcal{D}_S$ denotes the set of support vectors in the source domain, $\alpha_i^\star \in \mathbb{R}$ is the learnt coefficient corresponding to the ith sample of $\mathcal{D}_S$ and $b^\star \in \mathbb{R}$ is the bias part of the source model. The notation $K : \mathcal{T}_S \times \mathcal{T}_S \to \mathbb{R}$ is a kernel function in the source domain (for more details, see: Sec. 2.1.1). The

argument of the prediction function is $W\mathbf{x}$, which is the product of the transformation matrix W and an arbitrary sample $\mathbf{x} \in \mathcal{T}_T$ from the target domain. Here multiplying with W means the target-source domain mapping.

The kernel function in Eq. 4.6 is a parameter of the prediction function and the gradient of the error function depends on it. Thus, to calculate the gradient of the error function we have to specify the kernel function. We decided to apply two commonly used kernel functions to compute the necessary gradient: the Polynomial kernel and the RBF kernel [24, 31].

In Eq. 4.7, we can see the gradient of the error function applying the polynomial kernel. The exact form of the kernel is shown in this equation as well. The degree of the polynomial is denoted by p.

$$\begin{aligned} K_p(\mathbf{x}_i, W\mathbf{x}) &= (\mathbf{x}_i^T W\mathbf{x})^p, \\ \nabla E_{\mathcal{D}_T, f_S, K_p}(W) &= -p \sum_{(\mathbf{x}_i, y_i) \in SV_S} \alpha_i y_i \sum_{(\mathbf{x}, y) \in \mathcal{D}_T} (y - f_S(W\mathbf{x})) \cdot \\ &\quad K_{p-1}(\mathbf{x}_i, W\mathbf{x}) \mathbf{x}_i \mathbf{x}^T \end{aligned} \tag{4.7}$$

Similarly, in Eq. 4.8 we show the RBF kernel, and the gradient of the error function using the RBF kernel. Here γ is a parameter of the RBF kernel.

$$\begin{aligned} K_\gamma(\mathbf{x}_i, W\mathbf{x}) &= \exp(-\gamma \|\mathbf{x}_i - W\mathbf{x}\|^2), \\ \nabla E_{\mathcal{D}_T, f_S, K_\gamma}(W) &= -2\gamma \sum_{(\mathbf{x}_i, y_i) \in SV_S} \alpha_i y_i \sum_{(\mathbf{x}, y) \in \mathcal{D}_T} (y - f_S(W\mathbf{x})) \cdot \\ &\quad K_\gamma(\mathbf{x}_i, W\mathbf{x})(\mathbf{x}_i - W\mathbf{x}) \mathbf{x}^T \end{aligned} \tag{4.8}$$

These gradients can be applied in the Alg. 4.4. The whole learning system is referred to as PolyDML (in the case of instantiation applying the Polynomial Kernel, and the gradient shown in Eq. 4.7) and RBFDML (in the algorithm instance based on the RBF Kernel, and its gradient presented in Eq. 4.8).

4.2.5 Logistic Regression as Source Model

Since the problem of domain adaptation usually comes from the field of NLP, we decided to present another algorithm instance of Alg. 4.4 which is based on the Logistic Regression [59] (LR) classification method—which is an often used, and widely studied learner in NLP—as source model. Here we chose a more specific error function (the Cross-Entropy error function) for the transformation learning method:

$$\begin{aligned} E_{\mathcal{D}_T, f_S}(W) &= -\sum_{(\mathbf{x},y)\in\mathcal{D}_T}\sum_{c\in\mathcal{L}} t_c(\mathbf{x}) \ln f_{S,c}(W\mathbf{x}), \\ \text{where } t_c(\mathbf{x}) &= \begin{cases} 1 & y = c \\ 0 & \text{otherwise,} \end{cases} \\ \text{and } f_{S,c}(W\mathbf{x}) &= P(c|W\mathbf{x}). \end{aligned} \tag{4.9}$$

The chosen LR classification method can handle multi class classification problems as well. Thus, we assume that $\mathcal{L} = \{C_1, \ldots, C_l\}$ (multi-class classification setting, see: Sec. 2.1). Using the notation given in Eq. 4.9, we can express the prediction function of the source model in the following form:

$$f_S(W\mathbf{x}) = \text{argmax}_{c\in\mathcal{L}}\, f_{S,c}(W\mathbf{x}) = \text{argmax}_{c\in\mathcal{L}}\, P\left(c|W\mathbf{x}\right). \tag{4.10}$$

In the case of the LR method, the posterior probability of class $c \in \mathcal{L}$ is expressed in the form [15]:

$$f_{S,c}(W\mathbf{x}) = \frac{\exp(\boldsymbol{\alpha}_c^{\star T} W\mathbf{x})}{\sum_{k\in\mathcal{L}} \exp(\boldsymbol{\alpha}_k^{\star T} W\mathbf{x}) + 1}, \tag{4.11}$$

where $\boldsymbol{\alpha}_j^\star \in \mathbb{R}^n$ is the vector of learnt coefficients corresponding to the class $j \in \mathcal{L}$. Like in the previous subsection, the argument of the function presented in Eq. 4.11 is $W\mathbf{x}$, which is the product of the transformation matrix W and an arbitrary sample $\mathbf{x} \in \mathcal{T}_T$. As earlier, multiplying with W means the transformation from the target domain to the source domain.

To get another particular algorithm from the gradient descent-based algo-

rithm template (Alg. 4.4), applying the chosen setting, we have to calculate the gradient of the selected error function. The necessary gradient, using Eq. 4.9 is:

$$E_{\mathcal{D}_T,f_S}(W) = -\sum_{(\mathbf{x},y)\in\mathcal{D}_T}\sum_{c\in\mathcal{L}} t_c(\mathbf{x})\left(\boldsymbol{\alpha}_c^T W\mathbf{x} - \ln\left(\sum_{k\in\mathcal{L}}\exp(\boldsymbol{\alpha}_k^T W\mathbf{x})+1\right)\right),$$
$$\nabla E_{\mathcal{D}_T,f_S}(W) = -\sum_{(\mathbf{x},y)\in\mathcal{D}_T}\sum_{c\in\mathcal{L}} t_c(\mathbf{x})\left(\boldsymbol{\alpha}_c - \frac{\sum_{k\in\mathcal{L}}\exp(\boldsymbol{\alpha}_k^T W\mathbf{x})\boldsymbol{\alpha}_k}{\sum_{k\in\mathcal{L}}\exp(\boldsymbol{\alpha}_k^T W\mathbf{x})+1}\right)\mathbf{x}^T. \tag{4.12}$$

After having created the necessary gradient presented, we get a new instance of Alg. 4.4, the Logistic Regression Domain Mapping Learner (LRDML) method.

4.3 Experimental Results

In this section, we will provide an overview of the basic concepts of the evaluation process. Afterwards, the experimental results achieved on a synthetic dataset and real-world opinion mining tasks will be presented.

4.3.1 Evaluation Methodology

In our evaluations, we mainly investigated whether it was better to apply our domain adaptation approach, or to use a direct machine learning approach. We hypothesized that domain adaptation is especially required when the target training dataset is small, thus experiments using target training data with various sizes were carried out. In the case of extremely small datasets one evaluation per target domain size could not be trusted, thus for each size of the target domain we performed multiple runs and computed the average value of the elementary accuracy scores along with their variances.

Direct method (baseline): In this case we did not treat our task as a domain adaptation task. We only used the databases of the target domain. On the training part of the dataset of target domain we trained a model, and evalu-

ated on the evaluation part simply. It is a typical baseline of several domain adaptation tasks [35].

Transformation-based method: In each test setting we used three different databases, one taken from the source domain, and two taken from the target domain. The database from the source domain was used as a training database for the underlying classification method (f_S). The two databases from the target domain were treated as a training and evaluation split for the appropriate version of Alg. 4.4. This evaluation split was completely unknown during the preprocessing and learning phases.

4.3.2 Synthetic Database

To gain insight into the behaviour of our transformation-based approach, we first investigated synthetically generated source and target domains. In order to visualize it, both domains were two dimensional. The positive samples of the source domain were generated based on the sum of two Gaussian distributions, and the negative ones similarly, but using just one Gaussian distribution. We generated 1,000 samples and used only the first 800 of them as the training database of the source domain. The training and evaluation sets of the target domain were generated from the previously generated 1,000 samples by rotating them by 90 degrees and the same train-test split was employed.

In Fig. 4.1 we can see a sample run of the POLYDML algorithm on the synthetic database. We applied the Polynomial kernel with $d = 1$ (i.e. the linear kernel), and set the value of the trade-off parameter C of SVM (for more detail, see: Sec. 2.1.1) to 1.0. The samples from both the source and the target domain are shown in the same figure. The figure shows six different states of the algorithms. In each state we can see the data samples of the source domain and the classification boundary, which are constants. The first state shows the position of the original training samples of the target domain based on the samples taken from the source domain. The second state called "Iteration 0" shows the position of samples of the target domain which were transformed by applying a $W^{(0)}$ random transformation proposed by the applied POLYDML learner. The next four states show the first four iterations of the algorithm. For each

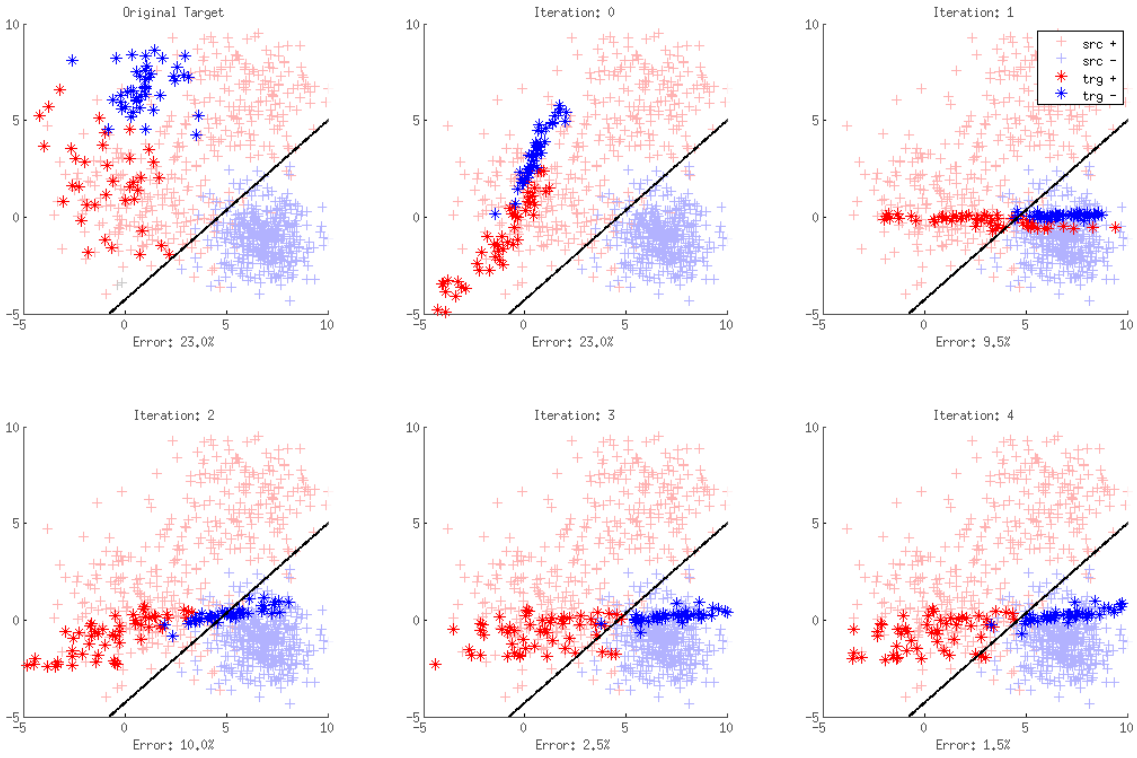

Figure 4.1: The first 4 iterations of POLYDML algorithm on Synthetic Database applying linear kernel.

state, we also included the error measured on the target training dataset. As one can see, in the initial states (i.e. in the first two states) the error rate is quite high, but in the first four iterations the error rate decreases fast and almost monotonically. We can see that when the monotony is not realized (i.e. from the iteration 1 to 2), the transformed samples of the target domain move to the "correct direction" according to the error minimization. The result shown in the figure suggests (empirically) that the proposed algorithm converges.

In the following evaluations, we compare the results of various Alg. 4.4 instances with direct methods (baseline). In Fig. 4.2, we can see the results of RBFDML (left hand side figure) and LRDML (right hand side figure) algorithms and the corresponding direct methods. Each point of the curve shows the averaged accuracy score and the corresponding variance of 100 runs of

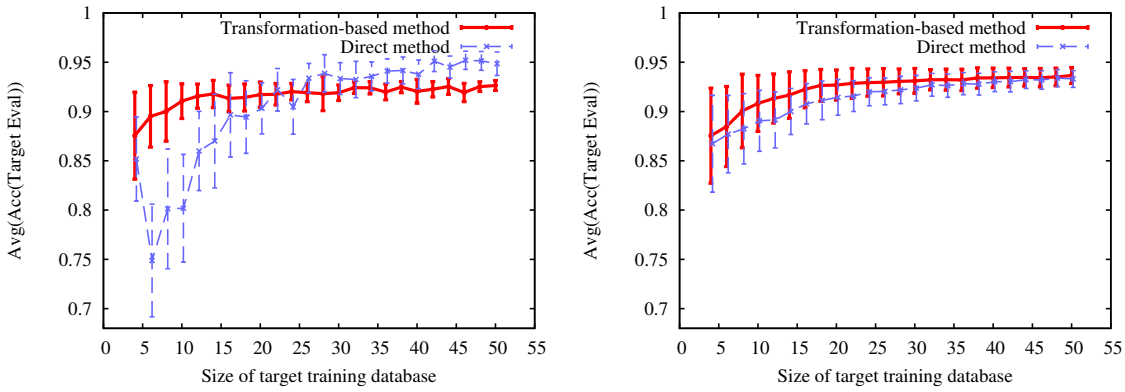

Figure 4.2: The average accuracies and variances of the RBFDML (left) and LRDML (right) algorithms using different sizes of subsets of the target domain of synthetic database.

transformation-based and 100 runs of direct methods. With SVM, we applied an RBF kernel with parameters $\gamma = 1$ and $C = 1$ (default values).

Our first observation is that both methods outperform the corresponding direct method when the size of the target domain is small. In the case of RBFDML, this improvement is more significant compared to the corresponding direct method than in the case of the LR method. However, it is more stable in the sense that the accuracy changes more smoothly. Based on this observation we can conclude that the SVM-based learning is more robust for the proposed domain adaptation approach since it results a more powerful improvement (in the small training sized regions). Furthermore, we can see that the variance of the accuracy decreases when we employ bigger datasets taken from the target domain, which suggests that we have more stable methods in these scenarios.

4.3.3 Results on the Multi-Domain Sentiment Dataset

The previous subsection empirically verified that the proposed transformation-based framework, and thus the proposed algorithms can achieve higher average accuracy on synthetically generated domains. Next, we will show that the proposed algorithms can achieve at least the same performance on real-world

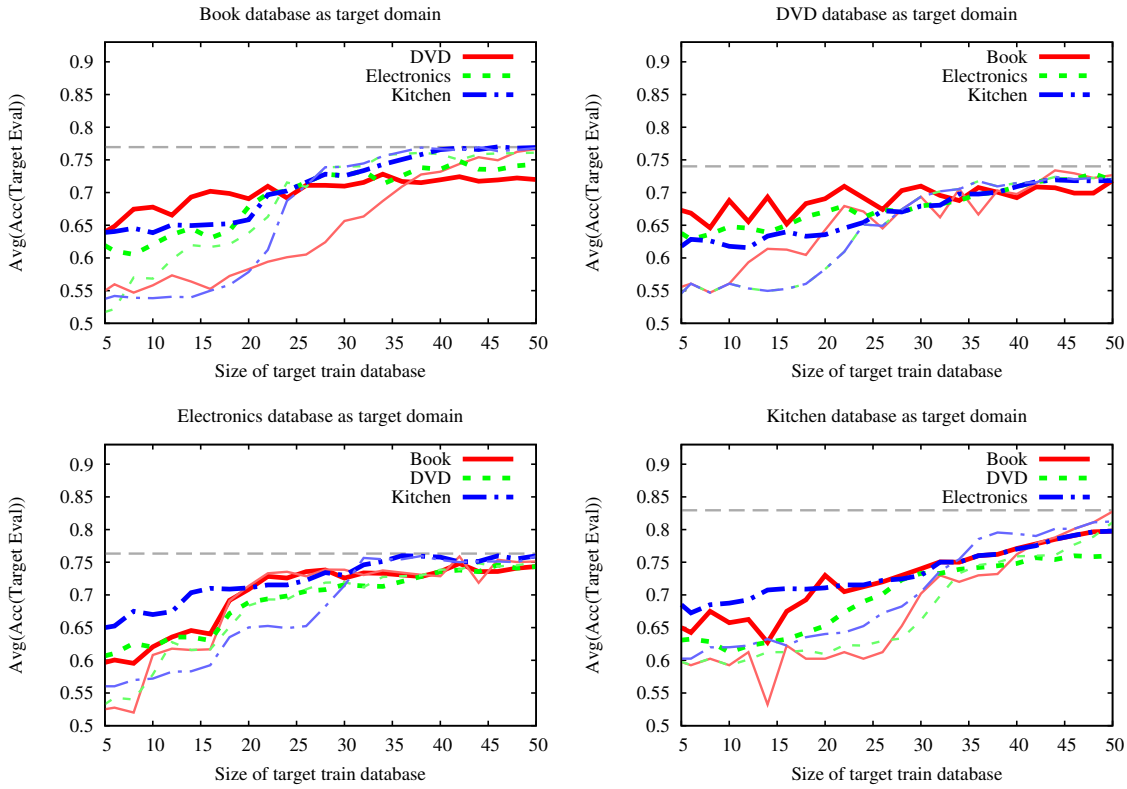

Figure 4.3: The average accuracies of RBFDML algorithm using different sizes of subsets of the target domains of Multi-Domain Sentiment Dataset.

datasets as on a synthetically generated one.

In order to do this, we experimented some variants of Alg. 4.4 on an opinion mining dataset [16] as well. This dataset contains product reviews taken from Amazon.com for four product types (domains), namely: books, DVDs, electronics and kitchen appliances. Originally the attributes of instances of each dataset were the term frequencies of the words of the corresponding review texts, and the labels of the instances were generated from the rating of the reviews. More precisely, reviews with rating ≥ 3 were considered as positive, while those with rating < 3 were labeled negative (binary classification problem, for more details see: Sec. 2.1). The datasets of each domain were balanced, all of them having 1,000 positive and 1,000 negative samples with a very high dimensionality (about 5,000 dimensions).

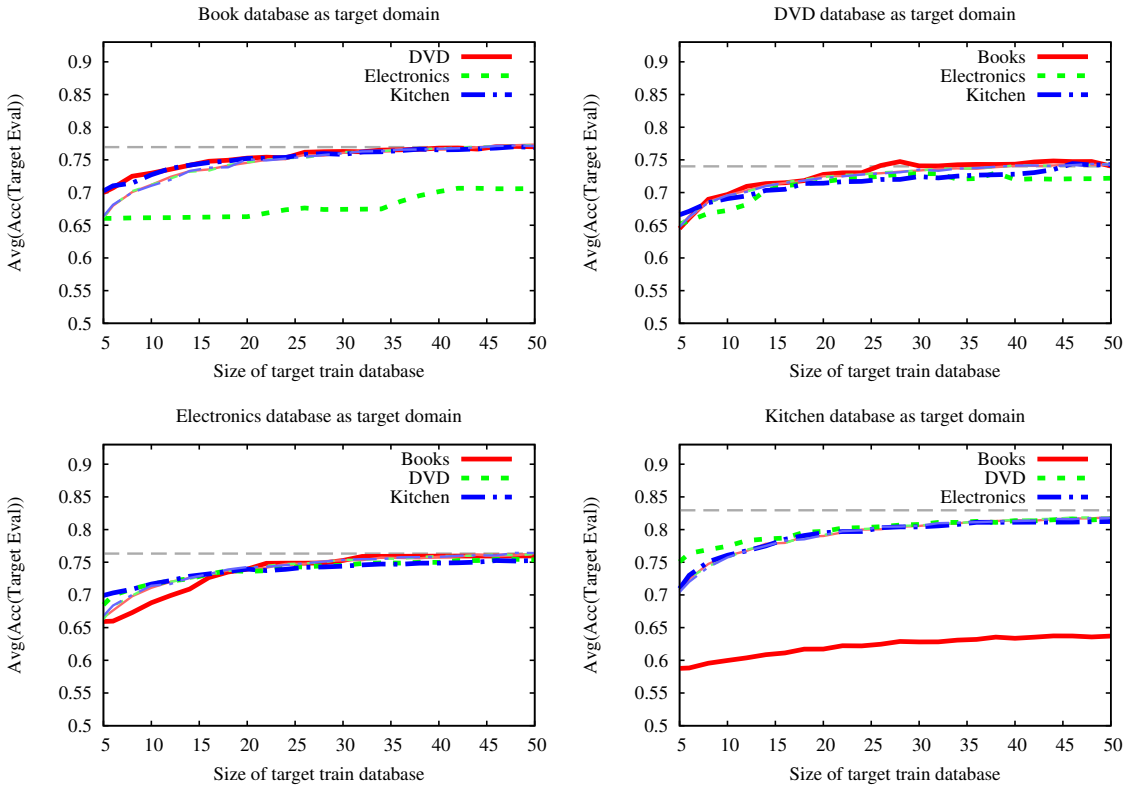

Figure 4.4: The average accuracies of LRDML algorithm using different sizes of subsets of the target domains of Multi-Domain Sentiment Dataset.

We carried out some dimensional reductions before the experiments. First, we split the datasets of each domain into two parts randomly (80% training set and 20% evaluation set). Then, we performed a feature selection step selecting the attributes where the so-called InfoGain score [15] was greater than 0 on the training set. Finally, we performed a Principle Component Analysis (PCA) [15] on each training database. We performed the same preprocessing on the evaluation set as well selecting exactly those features than in the case of the training part, and applying the transformation (PCA) obtained from the training database.

The scenario here was the same as that in the previous subsection except that databases were replaced by the datasets of Multi-Domain Sentiment Dataset. Since we had four different domains, we investigated all the possible 12

domain adaptation tasks applying both the algorithm RBFDML (Fig. 4.3) and algorithm LRDML (Fig. 4.4). In both cases the results are presented depending on the size of the database of the given target domain as earlier. The definition of the domain adaptation task (that is, what the target domain is) can be seen from the top labels of the elementary figures. In each subfigure the thinner lines correspond to the direct methods, while the horizontal lines show the result of the learner applying the full training database of the target domain. This is independent of the values of the x axis and can be viewed as the "limit values" of the corresponding results. At each point in the sub-figures we can see average accuracy scores of 10 independent evaluations.

As can be seen in Fig. 4.3 and Fig. 4.4, when we use limited-sized datasets from the target domain, in almost all of the cases, the proposed methods can achieve a higher accuracy than the baseline methods. However, there are two adaptation scenarios, namely the "Electronics to Books" and "Books to Kitchen", where the applied LRDML method got stuck in a local optima, and did not converge. This never happened to the SVM-based RBFDML methods. Moreover, these methods achieved a higher relative improvement compared to the corresponding direct methods. However—similar to the results observed on synthetic data—the LRDML usually provided a smoother error curve. In sum, these results implies that the RBFDML approach is more preferred against the LRDML approach.

Based on the simulations on synthetic and the real-world data, we can conclude that the approach presented in Alg. 4.4 is more suitable than the direct method. The reason for this phenomenon might be that the baseline could not made valid generalizations from the small number of samples—since the database of the target domain might not contain enough information to build a well-generalizing model—but the transformation-based approach using the well-generalizing source model which can achieve real generalizations which can be utilized by the transformation-based mechanism. Thus, in these cases the transformation-based method is the better choice.

The Structural Correspondence Learning (SCL), mentioned briefly in Sec. 4.1, is a related domain adaptation approach [16], which has published results on the opinion mining datasets we used. In comparison with its results, our best

approach (RBFDML) achieved better accuracy scores 10 times compared to the base SCL, and 7 times compared to its extended version (SCL-MI) out of the possible 12 scenarios.

4.4 Conclusions

In this chapter, we presented our novel, transformation-based approach for handling the task of domain adaption. We described two instances of our main algorithm, and experimentally showed that—applying them to a real world dataset in 12 different scenarios—our methods outperform the baseline approaches (direct methods), and published results of the same dataset.

Our experimental results proved that our general approach (Alg. 4.4) is capable of training models for the target domain which use a very limited number of labeled samples taken from the target domain. This is even true in the scenarios where there are enough samples, but baseline methods cannot generalize well using such samples. On the other hand, our approach has a key advantage against other domain adaptation procedures as it does not require access to the source data just to a trained source model, which can be crucial in specific scenarios (e.g. privacy issues).

Moreover—based on the sequence of thorough empirical evaluation—we concluded that the SVM-based RBFDML algorithm is the most robust instance of the Alg. 4.4. This result can be interpreted as another advantage of the SVM-based learning, and shows the strength and generality of the SVM approach.

The results described in this chapter are based on one of our previous results presented in [91].

CHAPTER 5

SVM Supported Distributed Recommendation

Offering useful recommendations to users of fully distributed systems is clearly a desirable function in many application domains. Some examples for larger efforts towards this goal are the Tribler platform [101], and more recently the Gossple project [69]. A fully distributed approach is also more preferable relative to centralized solutions, due to the increasing concerns over privacy.

However, the problem is also extremely challenging. Apart from the fact that centralized recommender systems—although working reasonably sometimes—are still far from perfect, offering good recommendations in fully distributed systems involves a number of special problems like efficiency, security and reliability, to name just a few.

In this chapter we focus on a class of recommender systems, the so-called user-based *collaborative filtering* algorithms that are fairly simple, yet provide a reasonable performance [4]. The key concept is a similarity metric over the users, and recommendations are made on the basis of information about similar users.

This idea also naturally lends itself to a distributed implementation, as it can be easily supported by similarity-based overlay networks (like the T-MAN

protocol, detailed in Sec. 2.2). Indeed, many distributed protocols from related work follow this path in some way or another.

In this chapter, we would like to shed light on the effects of the basic design choices in this domain with respect to recommendation performance, convergence time, and the balancing of the network load that the system generates during its operation. That is, we turn to investigate the problem of performing efficient machine learning in a radical different environment from the centralized one, referred to as fully distributed environment. Additionally, we present an interesting application of the centralized SVMs for the problem of user modelling to extract recommender datasets. However, this application is unusual, it clearly shows the strength of the SVM learners.

Our contribution is threefold. First, we propose a mechanism for generating recommender datasets from implicit user feedbacks, and present some interesting properties of these databases, which have direct effects on the performance of the applied fully distributed recommender algorithms as we will empirically validate in Sec. 5.5. Second, we draw attention to the potential load balancing problem in distributed systems that manage similarity-based overlays for any purpose including recommendation or search. Third, we propose novel algorithms for similarity-based overlay construction. Moreover, we perform extensive simulation experiments on large benchmark datasets and compare our set of algorithms with each other and with a number of baselines. We measure prediction performance, examine its convergence and dynamics, and we measure load balancing as well.

This work is mainly based on our two previous publications [90, 92].

5.1 Related Work

First, we overview relevant ideas in recommender systems in general, and subsequently we discuss related work in the fully distributed implementations of these ideas, as well as additional related work that are based on similar abstractions.

A recommender system can be viewed as a service which supports e-commerce activities by providing items of interest for the users [99]. These al-

gorithms are often centralized and Web-based operating on huge amounts of data—mainly on the previous ratings of the users. The algorithms which are based on the previous ratings of other *similar* users follow the so-called collaborative filtering (CF) approach. They are based on the simple heuristic that people who agreed (or disagreed) in the past will probably agree (or disagree) again. Thus, the predicted rate of an unseen item for a given user can be estimated on the basis of the rates of other users with *similar tastes*.

In the field of CF algorithms there exist numerous approaches. *User-based* approaches try to model the rating of a given item for a user by an aggregation of ratings of other users on the same item [4]. Although these approaches are very simple and intuitive, they provide a relatively good performance [57]. User-based CF algorithms are modular, hence they can be used with different aggregation methods and similarity metrics. One widely-used aggregation method is

$$\hat{r}_{u,i} = \frac{\sum_{v \in N_u} s_{u,v} \left(r_{v,i} - \bar{r}_v \right)}{\sum_{v \in N_u} |s_{u,v}|} + \bar{r}_u \tag{5.1}$$

defined in [103], where $r_{u,i}$ and $\hat{r}_{u,i}$ denote the known and the predicted rate of item i by user u, $\bar{r}_u$ and N_u denote the average rate and the neighbor set of user u, and $s_{u,v}$ measures the similarity between user u and v (e.g. cosine similarity [4] or Pearson similarity [4] can be employed).

Our preliminary experiments showed that (among several variants) the aggregation method shown in Eq. 5.1 combined with the cosine user similarity gives the best performance on our particular benchmarks. Since the focus of the present chapter is not recommendation performance per se, but the analysis of several distributed implementations of the basic idea of user-based CF, we fixed these methods in our experiments.

We should mention that there are numerous other approaches for recommendation such as the ones based on machine learning [14, 97], matrix factorization [115], generative models [76], clustering [88, 97], and dimension-reduction [14, 46].

Moving on to distributed methods, we emphasize that we focus on P2P recommendation, that is, the computational environment in which our algorithms work is identical with the one was mentioned in Sec. 2.2. Here, we do

not consider the parallel or cloud based implementations of centralized recommender techniques.

The largest group of methods define an overlay network based on some sort of similarity, and define a recommender algorithm on this network. For example, [99] and [25] follow this approach, although the overlay construction itself is not discussed or it is assumed to be done offline. The recommender algorithms then perform a search in this overlay up to a certain depth or up to a certain level of similarity, and aggregate the matching users with a standard method.

A slightly weaker approach is described in [117], where only a random network is assumed and the recommendation problem is treated as a search problem where a node needs to find similar users using a flooding based unstructured search.

A somewhat surprising result is described by Bakker at al. [10], where they argue that in fact it is enough to take a random sample of the network and use the closest elements of that sample to make recommendations. Our results are consistent with this observation, although we describe better and equally cheap alternatives.

A more sophisticated approach is described by Bickson et al. [13]. They define recommendation as a smoothing operation over a social network, which is expressed as a minimization problem using an objective function that expresses the requirements for the recommendation. The problem is solved by using an iterative method. Unfortunately no results are given on recommender system benchmarks due to the slightly different formulation of the basic problem.

It is of course possible to apply distributed hash tables [50]. Here, users are stored in a hash table and they are indexed by (item, rate) pairs as keys. Using this data structure, the users for a given item and rate are available from the distributed hash table (DHT) on demand. This method is not scalable if there are many recommendations to be made in the system, since the necessary information is not always available locally.

One of the most detailed studies on distributed recommender systems with performance evaluation can be found in [120]. In their approach the predicted

rate is defined as the expected value of the possible rates. Applying the Bayes' rule, three different so-called relevance models were derived: the user-based relevance model, the item-based relevance model and the unified relevance model which can be considered as a mixture of the previous two models. To estimate the necessary probabilities they applied kernel density estimation, which makes their method modular by giving opportunity for using different kernel functions. The proposed models were implemented on the basis of the BUDDYCAST [102] overlay management service, which is the main overlay management method of the Tribler file sharing protocol [101]. We used our own implementation of this model as a baseline method, since the original study [120] did not carry out load balancing measurements.

5.2 Inferring Ratings from Implicit User Feedbacks

Approaches for recommender systems take ratings as input. The ratings are often taken from a small numeric range, but in any case, one needs at least two different values, which stand for "dislike" and "like". These databases usually contain explicit ratings of items by the users. Since users have to make an effort to rate items, ratings are often scarce and of dubious quality. For this reason, it has long been observed that *inferring ratings from user behavior* is an important way of enhancing the ratings and thus the quality of recommendations. For example, [67] provides an overview of many such methods.

In this section, we propose a method for inferring binary ratings from BitTorrent networks. We will work with a large Filelist.org trace collected by the Technical University of Delft. Moreover, we will argue that the inferred ratings "make sense", through demonstrating that both the "like" and "dislike" classes of ratings can be captured by an SVM learner showing an unusual use-case of the learning algorithm.

5.2.1 Filelist.org Trace

In this section we will provide a brief overview of the dataset that is the basis of our benchmark. The data source we present originates from a BitTorrent-

based [30] P2P file-sharing community called FILELIST.ORG [2].

In a BitTorrent P2P network, each peer (user) downloads and uploads data simultaneously. The *torrent* file format describes one or more files that are to be shared. A *swarm* is the set of peers participating in downloading a common torrent. The peers that have completed downloading all the pieces of a torrent are called *seed*s, whereas the ones still trying to get some of them are called *leech*es.

The FILELIST.ORG is a private BitTorrent community[124, 125], meaning that members have to be invited by a senior member in order to be able to join the community website. Also, they have to comply with specific rules regarding their overall sharing ratio. We will base our benchmark dataset on the trace files gathered by Jelle Roozenburg at the Technical University of Delft [106] between 9th December, 2005 and 12th March, 2006. Over the course of 93 days, 91,745 peers (users) were observed in 3,068 swarms (sets of users participating in downloading the same file). For the files that were followed, 5,477 TB of data was reported to have been exchanged (2,979 TB up, 2,498 TB down[1]) among all the users combined. The churn rate, 9,574,290 joins and leaves were observed. For the measurements, the tracker website is periodically crawled to obtain a partial state of the P2P network. The original trace is rich in data as it consists of 691,319,475 events.

5.2.2 Inferring Ratings from the Trace

The original trace was first converted into a more convenient format, removing any unnecessary information, then the remaining table consisted of the following fields: user ID, item ID, timestamp, online/offline, completion.

These discrete points define a sequence of online and offline sessions for each user in each swarm. Using these sessions, we extrapolated file-ownership at any point in time as follows. We first filled the inner parts of each online session using the interpolation rules given in Tab. 5.1. Afterwards, we filled the offline sessions of each user-file pair by applying the rules given in Tab. 5.2. From these intervals we were able to generate a user-item database for any

[1]The difference between the two amounts come from measurement errors.

Table 5.1: Online session interpolation rules.

front	end	fill
0	0	0
0	1	1
1	0	0
1	1	1

Table 5.2: Offline session interpolation rules.

front	end	fill
0	0	0
0	1	0
1	0	0
1	1	1

given point in time.

A user-item database that corresponds to a point in time contains two kinds of entries: "has file" and "does not have file". That is, if a user has already completed downloading the given file, and has not removed the file, then he has the file. Note that if a user is downloading a file, but has not yet completed the downloads, then he does not have the file and is just like a user who has never attempted to download it.

Note that we could have retained information on swarm membership as well; that is, introduce the third label "downloading file". Though possible, we eventually decided not to do it so as to simplify the problem and avoid the semantic problems associated with this label. Also note that it is possible that a user has the file, but is not seeding it, so we still say "does not have file". However, it is impossible to decide whether a user removed the file because he does not like it or because he does not want to seed it. Overall, one has to be careful with interpreting these labels, but, as our evaluation will show, this labeling does result in useful results despite these problematic issues.

Sparsity is the ratio of the known and unknown ratings. This is one of the most important characteristics of recommender benchmarks [58]. If the data is very sparse (as in the case of the BookCrossing benchmark [127]), then numerous problems arise such as difficulties with measuring the similarity between users or building statistical models.

For this reason, we examined the dynamics of sparsity as a function of time, as shown in Fig. 5.1. In our case, we took the label "has file" to be the known rating; otherwise we considered the rating as unknown. We can observe that

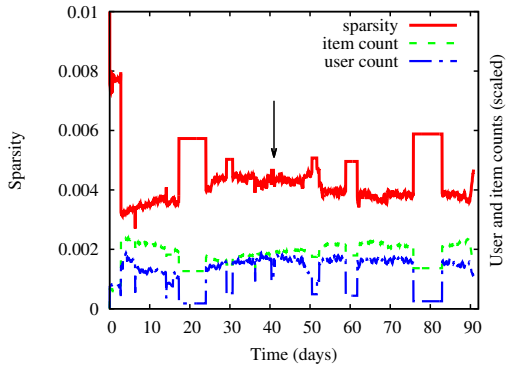

Figure 5.1: Sparsity changes during trace interval.

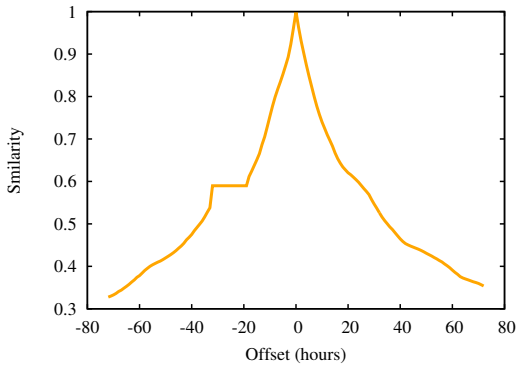

Figure 5.2: Similarity as a function of time.

sparsity is fairly stable, and has a small value; that is, the data is sparse. Based on this observation, we conclude that—at least in this property—the database that we infer is pretty similar to the BookCrossing benchmark dataset [127]. Fig. 5.1 shows the number of online users and the number of active files as well, as a function of time. The plateaus of the curves correspond to missing data.

In order to infer the preference of users, we first fixed a point in time, and took the corresponding user-item matrix. Originally, we took three different points in time, but our preliminary results indicated that there is no visible difference in performance over time, so we kept the time point indicated by an arrow in Figure 5.1.

Our baseline (or naive) dataset is given directly by the user-item matrix we selected earlier. From the point of view of ratings, we took "has file" to be a positive rating (indicated by the numeric value 1), otherwise we indicated *a lack of rating*. In other words, in the baseline user-item matrix we did not have any entries indicating a negative rating, since we have no basis to infer a negative rating from this data.

To infer negative ratings, and to make the positive ratings more precise, we used information that varied in time: we looked at file ownership before and after the timestamp of the baseline dataset. The amount of shift in time

Table 5.3: Rating conversion rules.

before	actual	after	inference
0	0	0	unspecified
0	0	1	unspecified
0	1	0	**0 (dislike)**
0	1	1	**1 (like)**
1	0	0	**0 (dislike)**
1	0	1	unspecified
1	1	0	unspecified
1	1	1	**1 (like)**

was the same in both directions. This way, for each user-item pair we got a triplet, which we converted to ratings, as indicated in Table 5.3. These rules are entirely heuristic and are based on common sense. This way we can create negative ratings (with a numeric value of 0).

It is interesting to observe the similarity of the user-item matrices (ratio of entries with an identical label) as a function of time shift. Figure 5.2 shows the difference between our baseline dataset and the user-item matrix with the given time shift. The figure indicated that there is a significant change in the user-item matrix, and thus the dynamics might indeed convey useful extra information.

5.2.3 Evaluation

Since we do not have any ground truth available for the actual like/dislike ratings of users, we need to apply an indirect approach to test the labels we assigned to user-item pairs. The method we propose is based on the *learnability* of the labels. That is, if a statistical learning algorithm is able to predict the labels of an independent validation set based on that it learns from a training set (where both the training and the validation sets are the disjoint subsets of the data we generated), then we can at least say that the labels do correspond to some regular property of the data. This regularity might come from an unintended source as well. To deal with this possibility, during the evaluation

we explicitly attempted to single out some trivial sources of regularity such as one label being more frequent than the other, etc. In the experiment we could not find any trivial or unintended reasons for the learnability of the labels.

For learning, we used the SMO [100] algorithm from the WEKA Java library [49]. The SMO algorithm solves the optimization problem which arises during the training of SVM defined in Eq. 2.8. To deal with the quadratic optimization problem in the size of the number of training examples, the SMO applies an iterative optimization process. Each elementary optimization step is performed in a subspace, referred to as working set, of the original feature space. The working sets vary during the steps of the iterative optimization process, but the dimensionality of each working set is always two. In these two dimensional working sets, the optimization step can be solved analytically, which makes the solver pretty accurate, however—especially in the case of large problems—it shows slow convergence due to the applied small subspace in which each particular optimization step is performed.

In order to generate features from the user-item ratings, we adopted the proposals described in [97]. These features are specifically designed for sparse datasets like ours.

For the baseline user-item matrix, the training set was generated by first selecting 90% of the user-item entries with the "has file" (1) label, and then we selected the same number of entries with the "does not have file" label. The test set was composed of the remaining 10% of the "has file" entries, and the same number of "does not have file" entries, disjunct from the training set. A similar method was used for the time-shift based datasets, for all the given time-shift values. The only difference is that there we had three labels: 1 (like), 0 (dislike) and null (don't know). Accordingly, to define a two-class learning problem, we created two pairs of training and test sets: one for the "like" class, and one for the "dislike" class.

The results are shown in Figure 5.3. The F-measure is the usual indicator for evaluating learning algorithms. It is defined as the harmonic mean of *precision* and *recall*. Precision is defined as the number of true positive predictions divided by all the positive predictions (how many of the positive predictions are indeed positive), while recall is given by the true positive predictions divided

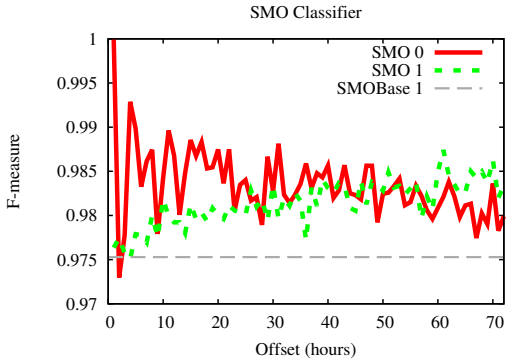

Figure 5.3: F-measure scores of the different time-shift datasets for positive (1) and negative (0) ratings against the result on the file ownership dataset.

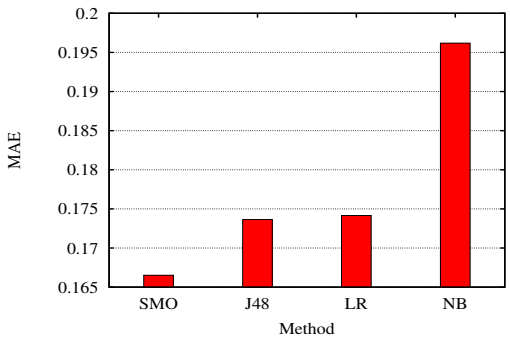

Figure 5.4: The mean absolute error (MAE) of the SMO, J48, LogReg and Naive Bayes methods (smaller values are better).

by the sum of the true positive and false negative predictions (how many of the positives do we catch).

It is evident that all the classes are learnable (randomly assigning labels to data results in an F-measure of 0.5). Besides, it is also evident that both classes achieve a higher F-measure score than the baseline does. That is, clearly, applying the time-shift preprocessing, we were able to achieve a better quality dataset.

The published time-shift based database is available for research purposes and can be downloaded from [1]. Currently, an inferred dataset from the $[-60; +60]$ hour interval is available in the following format: three numerical columns separated by a tabulator, which are the user ID, the item ID, and the rate (which is either 1 or 0, meaning the item was liked or disliked, respectively).

We conducted experiments with the above-mentioned methods on this database, whose results are shown in Fig. 5.4. In this evaluation case we split the database into train and test sets in the ratio 9:1. We chose the samples of sets at random from a uniform distribution, and we introduced different statistical learning methods (SMO, J48, LogReg and Naive Bayes; all of them are from the WEKA [49] machine learning library with default parametrization). We used

the mean absolute error (MAE) measure to demonstrate the performance of the baseline methods on the database. The results that we can observe in Fig. 5.4 are consistent with our previous results. However, it is interesting to notice that the SMO solver significantly outperforms the other learning methods on this task.

Based on the above results, we can conclude that the proposed inferring method—while it is far from precise, because of its inherent heuristics nature—can produce a dataset which definitely has an interesting inner structure which can be caught applying different statistical learning methods. While this is only implicit evidence (since we have no ground truth available), we believe that our approach significantly improves the rating data reflecting real user preferences. In the following sections, we replace this dataset with the similar (in characteristics) BookCrossing [127] database, since it contains the ratings of more users on more items (that is, more large scale), and it is more widely used (that is, more comparable with other baseline algorithms) benchmark dataset.

5.3 Interesting Properties of CF Datasets

From this section, we introduce additional benchmark datasets, that are commonly used for evaluating recommender approaches. Before we propose our main results, we take a closer look at these datasets and show some properties that raise interesting—and so far largely overlooked—problems in distributed environments. The mentioned benchmarks are the MovieLens [57] dataset, the Jester [46] dataset and the BookCrossing [127] dataset.

Table 5.4 summarizes some basic statistics of our datasets. In the case of MovieLens we used the official r_a partition so that its evaluation set contained 10 ratings per user. For Jester and BookCrossing we produced the evaluation set as proposed in [10]: we withheld 6 ratings from the training set where possible (if the user under consideration had at least 6 rated items). In this table '# items $\geq$' means the minimal number of items rated by some user. Sparsity denotes the ratio of existing and possible rates in the training sets. The value MAE(med) is a trivial baseline for prediction performance; it is defined as the mean absolute error (MAE) computed on the evaluation set using the

Table 5.4: Basic statistics of datasets.

	MovieLens	Jester	BookCrossing
# users	71,567	73,421	77,806
# items	10,681	100	185,974
size of train	9,301,274	3,695,834	397,011
sparsity	1.2168%	50.3376%	0.0027%
size of eval	698,780	440,526	36,660
eval/train	7.5127%	11.9195%	9.2340%
# items $\geq$	20	15	1
rate set	$1, \ldots, 5$	$-10, \ldots, 10$	$1, \ldots, 10$
MAE(med)	0.93948	4.52645	2.43277

median-rate of the training set as a prediction value. Clearly, a very significant difference can be found in properties related to sparsity. This will have significant implications on the performance of our algorithms, as we will show later.

As mentioned before, in distributed settings one suitable and popular approach is to build and manage an overlay that connects similar users. This overlay can be viewed as a graph where each node corresponds to a user and there is a directed edge between user A and B if and only if user B belongs to the most similar users of A.

This overlay plays an important role in a P2P recommender system. First, the performance of the recommendation depends on the structure of the overlay. Second, the costs and load balancing of the overlay management protocol depend on the topology of this similarity network.

To the best of our knowledge, the second role of the similarity overlay has not been addressed so far in the literature. Nevertheless, it is an important issue, since the load generated by the overlay management process might correlate with the number of nodes that link to a given node as one of its most similar nodes. More precisely, the load of a node might correlate with its in-degree in the overlay network. Thus, if the in-degree distribution of the overlay network is extremely unbalanced (e.g. if it has a power-law distribution), some of the nodes can experience a load that is orders of magnitude higher than the average. Thus, it is very important to consider the in-degree distribu-

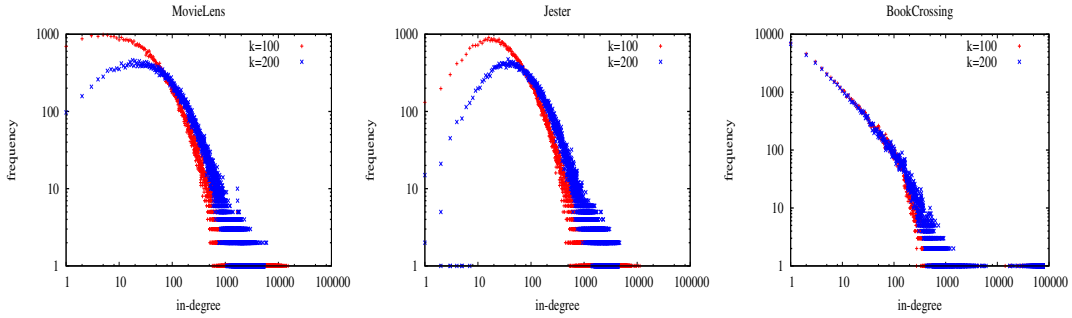

Figure 5.5: In-degree distribution of the benchmark datasets.

tion of the overlay when planning a P2P recommender system, and examine the incurred loads on the individual nodes as a function of this distribution.

Fig. 5.5 shows the in-degree distributions of the k nearest neighbor (kNN) overlay of each benchmark dataset. In this overlay each node has k directed outgoing edges to the k most similar nodes. As can be seen from the plots, the BookCrossing dataset has an almost power-law in-degree distribution, with many nodes having incoming links from almost every other node (note that the size of this dataset is around 77,806 users).

To see whether this might be a general property of high dimensional datasets, we need to consider some basic properties of high dimensional metric spaces. If we generate high dimensional uniform random datasets from the unit cube and construct their kNN graphs, we will find that most of the points lie on the convex hull of the dataset. These points are mostly situated at the same distance from each other. The nodes corresponding to these points have a mostly uniform and relatively small in-degree in the kNN graph. The very few points inside the convex hull are close to a huge number of points on the convex hull, and so have a high in-degree.

These observations indicate that we have to explicitly take into account load balancing when building a recommender system in a fully distributed manner.

5.4 Algorithms

The algorithms we examine all rely on building and managing a user-similarity overlay. In the top level of the protocol hierarchy, they apply the same user-based CF algorithm for making recommendations, strictly using locally available information (that is, information about the neighbors in the overlay).

Since we focus on overlay management, we fix the recommender algorithm and not discuss it any further. As it was mentioned in the previous sections, for this we need an aggregation method and a user similarity metric. We selected the aggregation shown in Eq. 5.1, proposed in [103]. Our similarity metric is cosine similarity, which achieved the best performance on our benchmarks. Note that the selected user similarity is of course known to the overlay management algorithm and is used to direct the overlay construction.

We also assume that the local views (or node related characteristics, as it was referred to in Sec. 2.2) of the nodes contain not only the addresses of the neighbors, but also a descriptor for each neighbor that contains ratings made by the corresponding user. This implies that computing recommendation scores does not load the network since all the necessary information is available locally. However, there is a drawback; namely the stored information is not up-to-date. As we will show later, this is not a serious problem since on the one hand, recommendation datasets are not extremely dynamic and, on the other hand, the descriptors are in fact refreshed rather frequently due to the management algorithms.

In sum, the task of overlay management is to build and maintain the best possible overlay for computing recommendation scores, by taking into account bandwidth usage at the nodes. We expect a minimal, uniform load from overlay management even when the in-degree distribution of the expected overlay graph is unbalanced.

5.4.1 BUDDYCAST based Recommendation

As we mentioned earlier we applied the BUDDYCAST overlay management protocol as a baseline method. Now we give a very brief overview of this algorithm and its numerous parameters; for details please see [102].

Algorithm 5.5 Random Nodes based Overlay Management

Require: k: the size of view; r: the number of randomly generated nodes

```
1: loop
2:     samples ← getRandomPeers(r)
3:     for i = 1 to r do
4:         peer ← get(samples, i)
5:         peerDescriptor ← descriptor(peer)
6:         insert(view, peerDescriptor)
7:     end for
8: end loop
```

The algorithm maintains a number of lists containing node descriptors. The taste buddy list contains the most similar users (peers), all those who communicated with the node before. The recommendation for a peer is calculated based on this list.

The BUDDYCAST algorithm contains a mechanism for load balancing: a block list. Communication with a peer on the block list is not allowed. If a node communicates with another peer, it is put on the block list for four hours.

Finally, a node also maintains a candidate list which contains close peers for potential communication, as well as a random list that contains random samples from the network. For overlay maintenance, each node periodically (in every 15 seconds by default) connects to the best node from the candidate list with probability α, and to a random list with probability $1 - \alpha$, and exchanges its buddy list with the selected peer.

5.4.2 kNN Graph from Random Samples

We assume that a node has a local view of size k that contains node descriptors. These will be used by the recommender algorithm.

In Algorithm 5.5 each node is initialized with k random samples from the network, and they iteratively approximate the kNN graph. The convergence is based on a random sampling process which generates r random nodes from the whole network in each iteration. These nodes are inserted into the view which is implemented as a bounded priority queue. The size of this queue is k and the priority is based on the similarity function provided by the recom-

mender module.

Applying a priority queue here on the basis of similarities means that nodes remember the most similar nodes from the past iterations. This means that since random samples are taken from the entire network, each node will converge to its kNN view with positive probability.

The method GETRANDOMPEERS can be implemented, for example, using the NEWSCAST protocol (for more details, please visit Sec. 2.2).

This algorithm does converge, as argued above, albeit very slowly. However, it is guaranteed to generate an almost completely uniform load since the only communication that takes place is performed by the underlying peer sampling implementation (NEWSCAST), which has this property.

5.4.3 kNN Graph by T-MAN

We can manage the overlay with the T-MAN algorithm as well (for a brief introduction to the main concepts of T-MAN, please see Sec. 2.2). This algorithm manages a view of size k, as in the random algorithm above. T-MAN periodically updates this view by first selecting a peer node to communicate with, then exchanging its view with the peer, and finally merging the two views and keeping the closest k descriptors. This is very similar to Algorithm 5.5, but instead of r random samples the update is performed using the k elements of the view of the selected peer.

In this chapter we examine the following methods for T-MAN which are employed as peer selection methods:

Global: This approach selects the node for communication from the whole network randomly. This can be done by using a NEWSCAST layer as it was described in the previous section. We expect this approach to distribute the load in the network uniformly since with this selection the incoming communication requests do not depend on the in-degree of the kNN graph at all.

View: In this approach the node for communication is selected from the view of the current node uniformly at random. The mechanism of this selection strategy is similar to the previous one, but the spectrum of the random selection is smaller since it is restricted to the view instead of the whole net-

work.

Proportional: This approach also selects a node for view exchange from the view of the current node, but here we define a different probability distribution. This distribution is different for each node and it is reversely proportional to the value of a selection counter, which measures the load of the node in the previous time interval. The exact definition of the selection probability for a neighbor j of node i is

$$p_{i,j} = \frac{\frac{1}{sel_j+1}}{\sum_{k \in View_i} \frac{1}{sel_k+1}}, \tag{5.2}$$

where sel_k is the value of the selection counter of the kth neighbor. This information is stored in the node descriptors. The motivation for this selection method is to reduce the load on the nodes that have a high in-degree in the kNN graph, while maintaining the favorable convergence speed of the T-MAN algorithm.

Best: The strategy that selects the most similar node for communication without any restriction. We expect that this strategy converges the most aggressively to the perfect kNN graph, but at the same time it results in the most unbalanced load.

5.4.4 Randomness Is Sometimes Better

Our experimental results (to be presented in Section 5.5) indicated that in certain cases it is actually *not* optimal to use the kNN view for recommendation. It appears to be the case that a more relaxed view can give a better recommendation performance.

To test this hypothesis, we designed a randomization technique that is compatible with any of the algorithms above. The basic idea is that we introduce an additional parameter, $n \leq k$. The nodes still have a view of size k, and we still use the same recommender algorithm based on these k neighbors. However, we apply any of the algorithms above to construct a (k-n)NN overlay graph (not a kNN graph), and we fill the remaining n elements in the following way: we take $r \geq n$ random samples (not necessarily independent in each cycle) and we take the closest n nodes from this list. With $n = k$ we get the algorithms

proposed in [10], and with $n = 0$ this modification has no effect, so we get the original algorithm for constructing the kNN graph.

5.5 Empirical Results

As our system model, we consider the fully distributed system model, which was introduced in Sec. 2.2. That is, a huge number of nodes communicate via messaging along an overlay network without any central control. Each node holds a partial view (node descriptors) about its direct neighbors.

Although the class of algorithms we discuss has been shown to tolerate unpredictable message delays and node failures well [63, 64], in this work we focus on load balancing and prediction performance, so we assume that messages are delivered reliably and without delay, and we assume that the nodes are stable.

We implemented our protocols and performed our experiments in PeerSim [85]. We performed a set of simulations of our algorithms with the following parameter value combinations: *view update* is random or T-MAN; *peer selection* for T-MAN is GLOBAL, VIEW, BEST or PROPORTIONAL; and the number of *random samples* is 20, 50, or 100 for random, and 0 or 100 for T-MAN.

The BUDDYCAST algorithm was implemented and executed with the following parameters: the size of the buddy list and the candidate list was 100, the size of the random list was 10, and α was 0.5. The size of the block list had to be restricted to be 100 as well, in order to be able to run our large scale simulations. The view size for the rest of the protocols was fixed at $k = 100$ in all experiments for practical reasons: this represents a tradeoff between a reasonably large k and the feasibility of large scale simulation.

In these simulations we observe the prediction performance in terms of the MAE measure and the distribution of the number of incoming messages per cycle at a node. Note that the number of outgoing messages is exactly one in each case.

Let us first discuss the effect of parameter r. This is a crucial parameter for random view update, while in the case of T-MAN the role of random samples is merely to help the algorithm to avoid local optima, and to guarantee con-

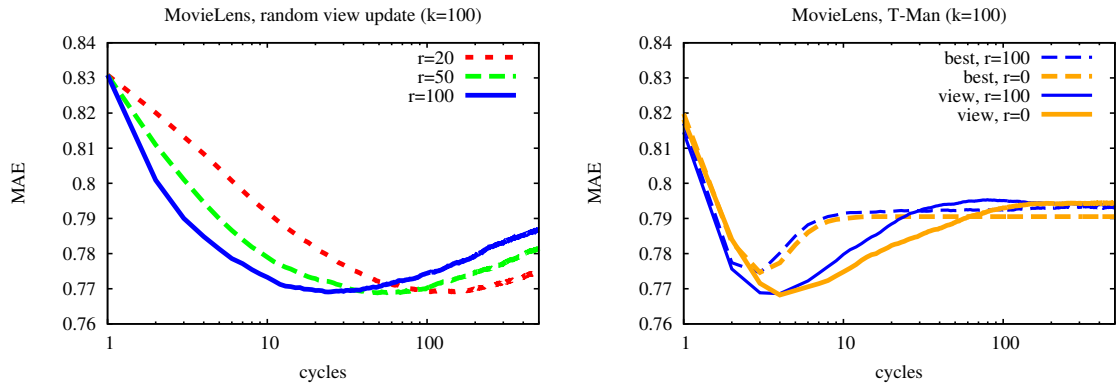

Figure 5.6: Effect of parameter r in a few settings.

vergence. Fig. 5.6 shows the effect of r in the case of the MovieLens database. The effect of r on the other databases and for other settings is similar.

We can observe that in the case of a random view update, r simply is a multiplicative factor that determines the speed of convergence: twice as many samples per cycle result in a halving of the necessary cycles to achieve the same value. In the case of T-MAN, the version with random samples converges faster, while the generated load remains the same (not shown). Accordingly, in the following we discuss T-MAN algorithms only with $r = 100$, and random view update algorithms only with $r = 100$.

In Fig. 5.7 we show the results of the experiments, where the MAE and the maximal load is illustrated. The maximal load is defined as the maximal number of incoming messages any node receives during the given cycle. The first interesting observation is that the load balancing property of the different algorithms shows a similar pattern over the three datasets, however, the convergence of the MAE is rather different (see also Table 5.4). In particular, in the case of the MovieLens and BookCrossing benchmarks the MAE reaches a minimum, after which it approaches the top-k based prediction from below, whereas we do not see this behavior in the much denser Jester database.

Indeed, the reason for this behavior lies in the fact that for the sparse datasets a larger k is a better choice, and our setting ($k = 100$) is actually far from opti-

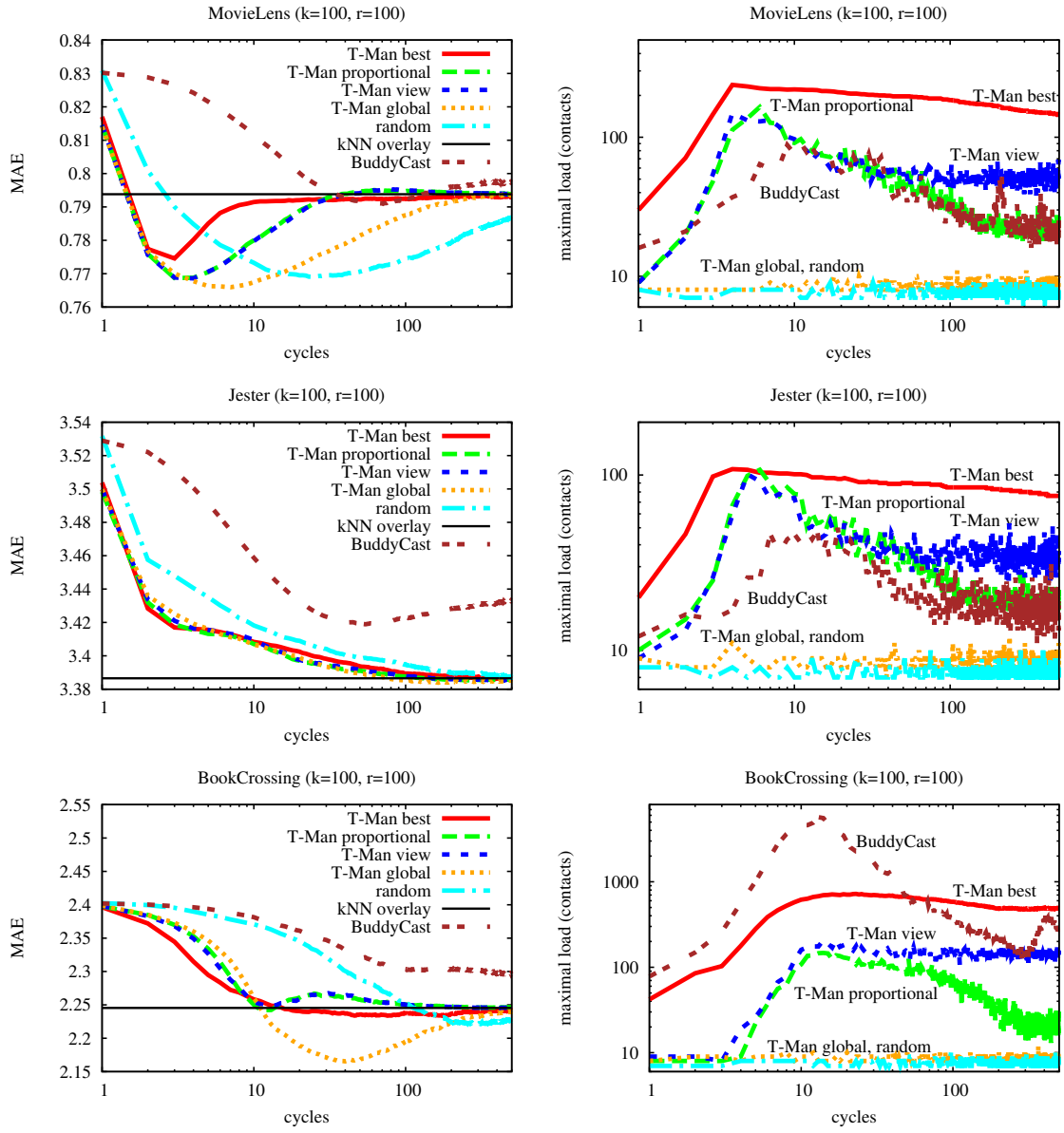

Figure 5.7: Experimental results. The scale of the plots on the right is logarithmic.

mal. In the initial cycles the view approximates a random sample from a larger k parameter. To verify this, we calculated the MAE of the predictions based on the algorithm described in Section 5.4.4. The results are shown in Figure 5.8 later on.

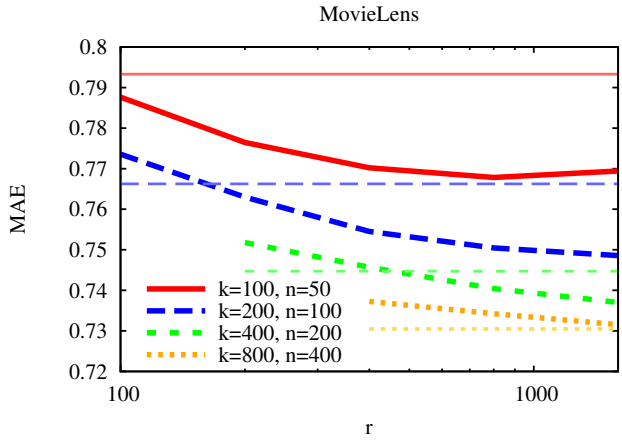

Figure 5.8: Effect of adding randomness to the view. Thin horizontal lines show the $n = 0$ case.

It is clear that for a small k it is actually better *not* to use the top k from the entire network; rather it is better to fill some of the views with the closest peers in a relatively small random sample from the network. Especially for the smallest k we examined ($k = 100$) this technique results in a significant improvement in the MAE compared to the recommendation based on the closest k peers in all datasets. This algorithm can easily be implemented, since we simply have to combine any of the convergent algorithms with an appropriate setting for k (such as $k = 50$) and use a peer sampling service to add to this list the best peers in a random sample of a given size.

As a closely related note, the random view update algorithms can be "frozen" in the state of minimal MAE easily, without any extra communication, provided we know in advance the location (that is, the cycle number) of the minimum. Let us assume it is in cycle c. Then we can use, for a prediction at any point in time, the best k peers out of the union of $c \cdot r$ random samples collected in the previous c cycles, which is very similar to the approach taken in [10].

Clearly, the fastest convergence is shown by the T-MAN variants, but these result in unbalanced load at the same time. The PROPORTIONAL variant discussed in Section 5.4.3 reduces the maximal load, however, only when the topology has already converged. During the convergence phase, PROPOR-

TIONAL behaves exactly like the variant VIEW.

Quite surprisingly, the best compromise between speed and load balancing seems to be GLOBAL, where the peer is selected completely at random by T-MAN. In many topologies, such as a 2-dimensional grid, a random peer possesses no useful information for another node that is far from it in the topology, so we can in fact expect to do worse than the random view update algorithm. However, in target graphs such as kNN graphs based on similarity metrics, a large proportion of the network shares useful information, namely the addresses of the nodes that are more central.

On such unbalanced graphs T-MAN GLOBAL is favorable because it offers a faster convergence than a pure random search (in fact, it converges almost as fast as the more aggressive T-MAN variants), however, the load it generates over the network is completely identical to that of random search, and therefore the maximal load is very small: the maximum of N samples from a Poisson distribution with a mean of 1 (where N is the network size). In addition, the node with the maximal load is different in each cycle.

Finally, we can observe that on the BookCrossing database some algorithms, especially BuddyCast and T-MAN with BEST peer selection, result in an extremely unbalanced degree distribution (note the logarithmic scale of the plot). This correlates with the fact that the BookCrossing database has the most unbalanced degree distribution (see Figure 5.5). Even though we have not optimized the parameters of BuddyCast, this result underlines our point that one has to pay attention to the in-degree distribution of the underlying kNN graph.

5.6 Conclusions

In this chapter we tackled the problem of the fully distributed recommendation including producing a dataset from explicit user feedbacks applying SVM as a validation method, and the construction of similarity-based overlay networks with user-based collaborative filtering as an application. We pointed out that one can generate meaningful (that is, learnable) recommender database without any ground truth, similarity-based overlays, built based on such a dataset, can have a very unbalanced degree distribution, and this fact might have a

severe impact on the load balancing of some overlay management protocols.

The main conclusion that we can draw is that in highly unbalanced overlays (that are rather frequent among similarity-based networks) the overlay construction converges reasonably fast even in the case of random updates; or, with T-MAN, uniform random peer selection from the network. At the same time, the traditional, aggressive peer selection strategies that have been proposed by other authors should be avoided because they result in a highly unbalanced load experienced by the nodes.

In sum, in this domain T-MAN with global selection is a good choice because it has a fully uniform load distribution combined with an acceptable convergence speed, which is better than that of the random view update. However, care should be taken because this conclusion holds only in these unbalanced domains, and in fact this algorithm is guaranteed to perform extremely badly in large-diameter topologies.

The results of the current chapter are mainly based on our earlier works [90, 92].

CHAPTER 6

P2PEGASOS—A Fully Distributed SVM

As we underlined many times in the previous chapters, data mining has numerous practical applications and these applications can be improved by supporting them with carefully designed, specific machine learning algorithms. In the previous chapters, we mainly focused on the algorithmic aspect of the design space. Here we turn to investigate the system model dimension deeply[1]. Particularly, we analyse how we can implement sophisticated machine learning algorithms in a specific fully distributed system *efficiently* and *robustly*.

In these types of fully distributed (peer-to-peer, P2P) systems, the problem of data aggregation has long been considered an important aspect. In the past decade, an extensive literature has accumulated on the subject. Research has mainly focused on very simple statistics over fully distributed databases, such as the average of a distributed set of numbers [62, 68], separable functions [86], or network size [82]. General SQL queries have also been implemented in this fashion [118]. The main attraction of the known fully distributed (mostly

[1]However, in the previous chapter we have already introduced the concept of learning in a P2P network, but there the learning itself was performed locally using data provided by the neighborhood.

gossip-based) algorithms for data aggregation is their impressive simplicity and efficiency, combined with robustness to benign failure.

Simple statistics or queries are very useful, but often more is needed. For example, for a P2P platform that offers rich functionality to its users including spam filtering, personalized search, and recommendation [9, 23, 101], or for P2P approaches for detecting distributed attack vectors [27], complex statistical learning models have to be built based on fully distributed, and often sensitive, data. At the same time, it would be highly desirable to build these models without sacrificing any of the nice properties of the aggregation algorithms mentioned above.

In sum, we need to find fully distributed, efficient, and lightweight data mining algorithms that make no or minimal assumptions about the synchrony and reliability of communication, work on fully distributed datasets without collecting the data to a central location, and make the learnt models available to all participating nodes.

In this chapter, we propose a SVM-based learning algorithm based on stochastic gradient search that meets the above mentioned requirements generated by the fully distributed environment. The key idea of our solution is that many models perform random walk over the network while being gradually adjusted to fit the data they encounter. Furthermore, we can even improve the performance of sequential stochastic gradient methods, exploiting the fact that there are many interacting models making random walks at the same time. These models can be combined applying local voting mechanisms.

The results presented in this chapter are based on those presented in our previous work [93].

6.1 System and Data Model

We assume that the system model of the fully distributed network that runs the proposed learning protocol is identical to that presented in Sec. 2.2. That is, a huge number of nodes communicate with each other without any central coordination along an overlay network. Both node and communication failures can occur.

Moreover, we assume that the overlay network is proposed by an instance of the NEWSCAST protocol [64]. That provides uniform random nodes for the learning protocol from the network, which are likely online at the time of the request. The API of the service consists of a local function GETRANDOMPEER(), which returns a random node address. For more detailed description of the system model and the NEWSCAT protocol, please see Sec. 2.2.

Related to the data model, we assume that each node stores exactly one supervised feature vector. This setting excludes the possibility of any local statistical processing. Moreover, we assume that these feature vectors never leave the node that stores them (i.e. considered as private for the nodes). This extreme data distribution model allows us to support applications that require extreme privacy, e.g. where the feature vectors are generated from profile information. Our goal is to build supervised classification models without collecting the data. For a more detailed description related to the data model and in general to the supervised learning, please read through Sec. 2.1.

6.2 Background

The proposed learning protocol is directly related to the Pegasos SVM algorithm [108]. Here we briefly overview the main characteristics of this learner and motivate why it is a suitable choice as a basis for our protocol.

Suppose that we are in a supervised binary classification setting and a $\mathcal{D} = \{(\mathbf{x}_1, y_i), \ldots, (\mathbf{x}_n, y_n)\} \in \mathbb{R}^d \times \{-1, +1\}$ training database is given. The Pegasos solver formalizes the SVM learning problem as follows [108] (primal form):

$$\text{minimize}_{\mathbf{w}} \quad \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i) \tag{6.1}$$

In this form of the SVM (unlike in those presented in Sec. 2.1.1), the b bias term is omitted, and the problem is represented as an unconstrained empirical loss minimization applying the Hinge loss and a penalty term for the norm of the classifier that is being learnt. The Pegasos learner solves the optimization

problem presented in Eq. 6.1 applying stochastic gradient method.

The stochastic gradient search based training is a common mechanism applied in the machine learning. Without going into too much detail, the basic idea is that we iterate over the training examples in a random order repeatedly, and for each training example, we calculate the gradient of the error function (which describes classification error), and modify the model along this gradient to reduce the error on this particular example. At the same time, the step size along the gradient is gradually reduced. In many instantiations of the method, it can be proven that the converged model minimizes the sum of the errors over the examples [39].

As we mentioned, the Pegasos solver directly optimizes the above mentioned objective function (shown in Eq. 6.1). In this primal form, the desired model $\mathbf{w}$ is *explicitly represented*, and is evaluated directly over the training examples. Contrary, the standard SVM algorithms usually solve the dual problem (in the form shown in Eq. 2.8) instead of the primal one [31]. Here the model $\mathbf{w}$ is *given indirectly*, as the sum of some training examples (the support vectors) weighted with the corresponding Lagrangian variables (shown in e.g. Eq. 2.9, the variables $\alpha_i^\star$). These Lagrangian variables specify how important the corresponding sample is from the point of view of the model.

However, the primal and dual formalizations are equivalent, both in terms of theoretical time complexity and the optimal solution[1]. A crucial difference is that, the methods that deal with the dual form require frequent access to the entire database to update the Lagrangian variables, which is unfeasible in our system model. Besides, the number of Lagrangian variables equals the number of training samples, which could be orders of magnitude larger than the dimension of the primal problem, d. Finally, there are indications that applying the primal form can achieve a better generalization on some databases [26].

6.3 Related Work

In this section, we consider the fully distributed data mining algorithms only, since some non-distributed SVM-based learning algorithms have been dis-

[1]However, solving the dual problem has some advantages as mentioned in Sec. 2.1.1; most importantly, one can take full advantage of the kernel-based extensions.

cussed in Sec. 2.1.1 and in the previous chapters. Moreover, we do not consider here the parallel and different cloud-based data mining algorithms either. These fields have a large literature, but the rather different underlying system model means it is of little relevance to us here.

In the area of P2P computing, a large number of fully distributed algorithms are known for calculating global functions over fully distributed data, generally referred to as aggregation algorithms. The literature of this field is vast, we mention only two examples: Astrolabe [118] and gossip-based averaging [62]. These algorithms are simple and robust, but are capable of calculating only simple functions such as the average. Nevertheless, these simple functions can serve as key components for more sophisticated methods, such as the EM algorithm [73], unsupervised learners [109], or the collaborative filtering based recommender algorithms [10, 51, 92, 117]. Usually, these approaches use other well-studied P2P services like some kind of overlay support, for example, T-MAN [61] (for more details related to the T-MAN protocol, please see Alg. 2.3 in Sec. 2.2).

In the past few years there has been an increasing number of proposals for P2P machine learning algorithms as well, like those in [5, 6, 7, 34, 56, 77, 109]. The usual assumption in these studies is that a peer has a subset of the training data on which a model can be learnt locally. After learning the local models, algorithms either aggregate the models to allow each peer to perform local prediction, or they assume that prediction is performed in a distributed way. Clearly, distributed prediction is a lot more expensive than local prediction; however, model aggregation is not needed, and there is more flexibility in the case of changing data. In our approach we adopt the fully distributed model, where each node holds only one data record. In this case we cannot talk about local learning: every aspect of the learning algorithm is inherently distributed. Since we assume that data cannot be moved, the models need to visit data instead. In a setting like this, the main problem we need to solve is to efficiently aggregate the various models that evolve slowly in the system so as to speed up the convergence of prediction performance.

As for SVM algorithms, we are aware of only one comparable P2P SVM implementation called Gadget SVM [56]. This implementation applies the above

Algorithm 6.6 P2P Stochastic Gradient Descent Algorithm

```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p
6: end loop
7: procedure ONRECEIVEMODEL(m)
8:     m ← updateModel(m)
9:     currentModel ← m
10:    modelCache.add(m)
11: end procedure
```

detailed mechanism (that is, it applies local learning) as well. Additionally, it requires round synchronization as well, which is clearly different from our asynchronous model. It applies the Push-Sum algorithm [68] as its main building block.

To the best of our knowledge there is no other learning approach designed to work in our fully asynchronous and unreliable message passing model, and which is capable of producing a large array of state-of-the-art models.

6.4 The Algorithm

The skeleton of the algorithm we propose is shown in Algorithm 6.6. This algorithm is run by every node in the network. When joining the network, each node initializes its model via INITMODEL(). After the initialization each node starts to periodically send its current model to a random neighbor that is selected using the peer sampling service (see Sec. 2.2 and Sec. 6.1). When receiving the model, the node updates it using a stochastic gradient descent step based on the training sample it stores, and subsequently it stores the model. The model queue can be used for voting, as we will explain later.

Recall that we assumed that each node stores exactly one training sample. This is a worst case scenario; if more samples are available locally, then we can use them all to update the model without any network communication, thus speeding up convergence.

In this skeleton, we do not specify what kind of models are used and what algorithms operate on them. For example, a model is a d dimensional hyperplane in the case of SVM, as described earlier, which can be characterized by a d dimensional real vector. In other learning paradigms other model types

Algorithm 6.7 P2Pegasos

```
1: procedure UPDATEMODEL(m)
2:     η ← 1/(λ · m.t)
3:     if y · m.w^T x < 1 then
4:         m.w ← (1 − ηλ)m.w + ηyx
5:     else
6:         m.w ← (1 − ηλ)m.w
7:     end if
8:     m.t ← m.t + 1
9:     return m
10: end procedure
11: procedure INITMODEL
12:     if stored modelCache exists then
13:         m ← modelCache.freshest()
14:     else
15:         m.t ← 0
16:         m.w ← (0, ..., 0)^T
17:     end if
18:     send model(m) to self
19: end procedure
```

are possible (not investigated here). To instantiate the skeleton, we need to implement INITMODEL() and UPDATEMODEL(). This can be done based on any learning algorithm that utilizes the stochastic gradient descent approach. In this chapter we will focus on the Pegasos algorithm [108], which implements the SVM method. The two procedures are shown in Algorithm 6.7.

We assume that the model m has two fields: $m.t \in \mathbb{N}$, which holds the number of times the model was updated, and $m.\mathbf{w} \in \mathbb{R}^d$ that holds the linear model. The parameter $\lambda \in \mathbb{R}$ is the regularization constant. In our experiments we used the setting $\lambda = 10^{-4}$. Vector $\mathbf{x} \in \mathbb{R}^d$ is the local feature vector at the node, and $y \in \{-1, 1\}$ is its correct classification. At line 4 the method updates the model executing the update mechanism. Here the local training sample is used. This mechanism gets executed if the local example $\mathbf{x}$ is misclassified by the received model $m.\mathbf{w}$ (see the condition at line 3). Otherwise the model is only deflated at line 6 due to the regularization constraint.

The effect of the algorithm will be that the models will perform a random walk in the network while being updated using the update rule of the Pegasos algorithm. In this sense, each model corresponds to an independent run of the sequential Pegasos, hence the theoretical results of the Pegasos algorithm are applicable. Accordingly, we know that all these models will converge to an optimal solution of the SVM primal optimization problem [108]. For the same reason, the algorithm does *not need any synchronization or coordination*. Although we do not give a formal discussion of asynchrony, it is clear that as long as each node can contact at least one new uniform random peer in a

Algorithm 6.8 P2Pegasos prediction procedures

```
1: procedure PREDICT(x)
2:     w ← currentModel.w
3:     return sign(w^T x)
4: end procedure
5: procedure VOTEDPREDICT(x)
6:     pRatio ← 0
7:     for m ∈ modelCache do
8:         if sign(m.w^T x) ≥ 0 then
9:             pRatio ← pRatio +1
10:        end if
11:    end for
12:    pRatio ← pRatio/modelCache.size()
13:    return sign(pRatio−0.5)
14: end procedure
```

bounded time after each successful contact, the protocol will converge to the optimal solution.

An important aspect of our protocol is that every node has at least one model available locally, and thus all the nodes can perform a prediction without performing communication. Moreover, since there are N models in the network (where N is the network size), we can apply additional techniques to achieve a higher predictive performance than that of an output model of a simple sequential implementation. Here we implement a simple voting mechanism, where nodes will use more than one model to make predictions. Algorithm 6.8 shows the procedures used for prediction in the original case, and in the case of voting.

Here the vector $\mathbf{x}$ is the unseen example to be classified. In the case of linear models, the classification is simply the sign of the inner product with the model, which essentially describes on which side of the hyperplane the given point lies. We note that MODELCACHE is assumed to be of a bounded size. When storing a new model in it, an old one will be removed if the queue is full. In our experiments we used a queue implementation, where the queue holds the 10 latest added models.

6.5 Experimental Results

We investigated the proposed P2PEGASOS algorithm against several baseline approaches, in numerous scenarios, applying three different data sets. First, we describe the exact evaluation setting, then present and discuss our results.

Table 6.1: The main properties of the data sets, and the prediction error of the baseline sequential algorithms.

	Iris1	Iris2	Iris3	Reuters	SpamBase	Malicious10
Training set size	90	90	90	2,000	4,140	2,155,622
Test set size	10	10	10	600	461	240,508
Number of features	4	4	4	9,947	57	10
Classlabel ratio	1:1	1:1	1:1	1:1	1,813:2,788	792,145:1,603,985
Pegasos 20000 iter.	0	0	0	0.025	0.111	0.080 (0.081)
Pegasos 1000 iter.	0	0	0.4	0.057	0.137	0.095 (0.060)
SVMLight	0	0	0.1	0.027	0.074	0.056 (–)

6.5.1 Experimental Setup

Data Sets. We selected data sets of different types including small and large sets containing a small or large number of features. Our selection includes the commonly used Fiser's Iris data set [43]. The original data set contains three classes. Since the SVM method is designed for the binary (two-class) classification problem, we transformed this database into three two-class data sets by simply removing each of the classes once, leaving classes 1 and 2 (Iris1), classes 1 and 3 (Iris2), and classes 2 and 3 (Iris3) in the data set. In addition, we included the Reuters [48], the Spambase, and the Malicious URLs [78] data sets as well. All the data sets were obtained from the UCI database repository [44].

Table 6.1 shows the main properties of these data sets, as well as the prediction performance of the baseline algorithms. SVMLight [65] is an efficient SVM implementation. Note that the Pegasos algorithm can be shown to converge to the same value as SVMlight [108].

The original Malicious URLs data set has about 3,000,000 features, hence we first reduced the number of features so that we could carry out simulations. The message size in our algorithm depends on the number of features, therefore in a real application this step might also be useful in such extreme cases. We used a simple and well-known method, namely we calculated the correlation coefficient of each feature with the class label, and kept the ten features with the maximal absolute values. If necessary, this calculation can also be carried out in a gossip-based fashion [62], but we performed it offline.

The effect of this dramatic reduction on the prediction performance is shown in Table 6.1, where the results of Pegasos on the full feature set are shown in parentheses (SVMlight could not be run due to the large size of the database).

Scenarios. The experiments were carried out in the event based engine of the PeerSim simulator [85]. The peer sampling service was provided by the NEWSCAST protocol (see Sec. 2.2 for more details). The network size is the same as the database size; each node has exactly one sample. Each node starts running the protocol at the same time. The protocol does not require a synchronized startup, but we need it here to analyze convergence in a clearly defined setting.

In our experimental scenarios we modeled message drop, message delay, and churn. The drop probability of each message was 0.5. This can be considered an extremely large drop rate. Message delay was modeled as a uniform random delay from the interval $[\Delta, 10\Delta]$, where Δ is the gossip period, as shown in Algorithm 6.6. This is also an extreme delay, which is orders of magnitude higher than what can be expected in a realistic scenario.

We also modeled a realistic churn based on probabilistic models proposed in [111]. Accordingly, we approximated the online session length with a log-normal distribution, and we approximated the parameters of the distribution using a maximum likelihood estimate based on a trace from a private BitTorrent community called FileList.org, obtained from Delft University of Technology [106]. We set the offline session lengths so that at any moment in time 90% of the peers were online. In addition, we assumed that when a peer came back online, it retained its state that it had at the time of leaving the network. We now list the scenarios we experimented with: *No failure*: there is no message drop, no delay and no churn; *Drop only*: we simulate message drop as described, but no other types of failure; *Delay only*: we simulate message delay only; *Churn only*: we simulate node churn only; *All failures*: we apply message drop, delay and churn at the same time.

Metrics. The evaluation metric we focus on is prediction error. To measure prediction error, we need to split the datasets into training sets and test sets. The ratios of this splitting are shown in Table 6.1. At a given point in time, we

select 100 peers at random (or all the peers, if there are fewer than 100) and we calculate the average misclassification ratio of these 100 peers over the test set using the current models of the peers. The misclassification ratio of a model is simply the number of the misclassified test examples divided by the number of all test examples, which is the so called 0-1 error.

Moreover, we calculated the similarities between the models circulating in the network using the cosine similarity measure. This was done only for the Iris databases, where we calculated the similarity between all pairs of models, and calculated the average. This metric is useful for studying the speed at which the actual models converge. Note that under uniform sampling it is known that all models converge to an optimal model.

6.5.2 Results

Figure 6.1 shows the results over the Iris datasets for algorithm variants that do not apply voting for prediction. The plots show results as a function of cycles. One cycle is defined as a time interval of one gossip period Δ. Although the size of each data set is the same, the dynamics of the convergence is rather different. The reason is that the learning complexity of a database depends primarily on the inner structure of the patterns of the data, and not on the size of data set. In trivially learnable patterns a few examples are enough to construct a good model, while under complex patterns a large number of samples as well as many iterations might be required. Since Pegasos also has a similar convergence behavior, we can be sure that this is not an artifact of parallelization.

Let us now turn to the analysis of the individual effects of the different failures we modeled, comparing them to two baseline algorithms. The first baseline algorithm is SVMLight, a sequential efficient SVM solver [65] that optimizes the dual SVM problem given in Eq. 2.8. It is independent of the cycles, hence its performance is shown as a horizontal line. The second baseline algorithm is Pegasos. We ran Pegasos 100 times, and show the average error at each cycle. Note that for Pegasos each cycle means visiting another random teaching example.

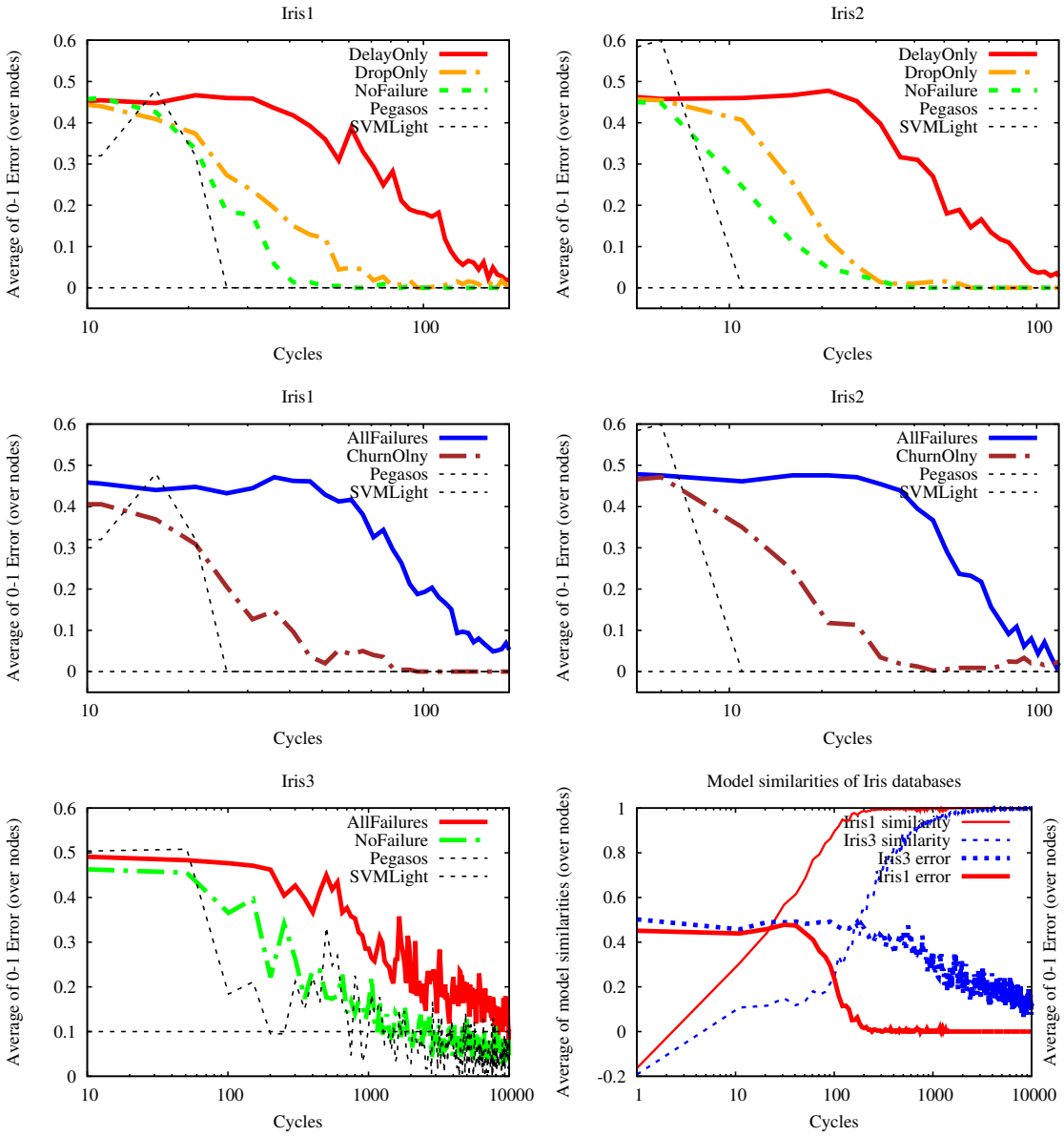

Figure 6.1: Experimental results over the Iris databases.

Clearly, the best performance is observed under no failure. This performance is very close to that of Pegasos, and converges to SVMlight (like Pegasos does). The second best performance is observed with churn only. Adding churn simply introduces an extra source of delay since models do not get for-

gotten as mentioned in 6.5.1. The situation would be different in an adaptive scenario, which we do not consider here. In the scenario with message drop only, the performance is still very close to the ideal case. Considering the extremely large drop rates, this result is notable. This extreme tolerance to message drop comes from the fact that the algorithm is fully asynchronous, and a 50% drop rate on average causes only at most a proportional slowdown of the convergence. Among the individual failure types, extreme message delay is the most significant factor. On average, each message takes as much as 5 cycles to reach its destination. The resulting slowdown is less than a factor of 5, since some messages do get through faster, which speeds up the convergence of the prediction error.

In Figure 6.1 we also present the convergence of the averaged cosine similarities over the nodes together with their prediction performance under no failures, without voting. We can see that in the case of each data set the models converge, so the observed learning performance is due to the good models as opposed to random influences.

Although, as mentioned above, in our case convergence speed depends mainly on data patterns, and not on the database size, to demonstrate scalability we performed large scale simulations as well with our large data sets. The results can be seen in Figure 6.2. Here we plotted just the two scenarios with no failures and with all the failures. The figure also shows results for the variants that use voting.

A general observation regarding the distinction between the P2PEGASOS variants with and without voting is that voting results in a better performance in all scenarios, after a small number of cycles. In the first few cycles, the version without voting outperforms voting because there is insufficient time for the queues to be filled with models that are mature enough. On some of the databases the improvement due to voting can be rather dramatic. We note that where the test data sets were larger (see Table 6.1) we obtained smoother convergence curves.

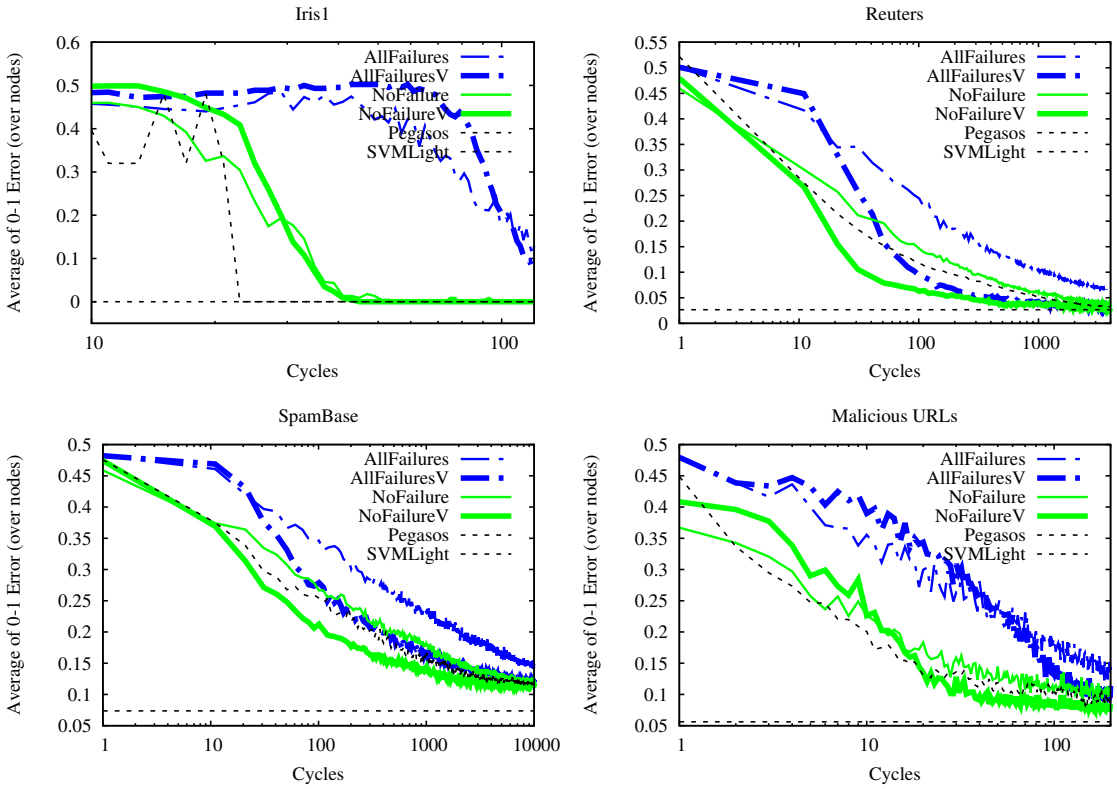

Figure 6.2: Experimental results over the large databases, and the Iris1 database. Labels marked with a 'V' are variants that use voting.

6.6 Conclusions

In this chapter we proposed a fully distributed SVM algorithm called P2PEGASOS. Nodes in the network gossip models that are continuously updated at each node along their random walk. Our main conclusion is that the approach is able to produce SVM models in a very hostile environment, with extreme message drop rates and delays, with very limited assumptions about the communication network. The only service that is needed is uniform peer sampling. The quality of the models are very similar to that of the centralized SVM algorithms.

Furthermore, we can also outperform the centralized Pegasos algorithm with the help of a voting technique that makes use of the fact that there are

many independent models in the network passing through each node. The models are available at each node, so all the nodes can perform predictions as well. At the same time, nodes never reveal their data, so this approach is a natural candidate for privacy preserving solutions.

The results shown in this chapter are mainly based on our previous study [93].

CHAPTER 7

Speeding Up the Convergence of P2PEGASOS

As we partially highlighted in the previous chapters, the main attraction of P2P technology for distributed applications and systems is acceptable scalability at a low cost (no central servers are needed) and a potential for privacy preserving solutions, where data never leaves the computer of a user in a raw form. Recently, there has been an increasing effort to develop machine learning algorithms—as we also proposed similar approaches in the previous chapters—that can be applied in P2P networks. This progress was motivated by the various potential applications such as spam filtering, user profile analysis, recommender systems and ranking.

In the previous chapter, we introduced an asynchronous learning protocol called P2PEGASOS. This protocol operates on the top of a fully distributed network performing an SVM-style supervised learning on the data available in the network. The learning approach we proposed there involves models that perform a random walk in the P2P network along the nodes proposed by the peer sampling service. Reaching each node, the models update them-

[1] In Chapter 6, we assumed that one training example is available per node. However, all the methods presented there can be extended to use more samples.

selves using the supervised training sample(s)[1] stored by the node. There are many models in the network which take random walks, so all nodes will experience a continuous stream of models passing through them. Apart from using these models for prediction directly, we introduced a simple ensemble technique which applies a voting mechanism on the recently observed models. There we showed that the protocol has accurate learning properties while it is extremely robust to different network failure scenarios. We observed pretty nice convergence properties as well.

Our goal—similarly to the goals of the previous chapter—is to develop algorithms for fully distributed learning, but here we want to *achieve approximately an order of magnitude higher convergence speed* than that presented in the case of original P2PEGASOS in Chapter 6. The design requirements—apart from the fast convergence speed—are the following. First, the algorithm has to be extremely *robust*. Even in extreme failure scenarios it should maintain a reasonable performance. Second, prediction should be possible at any time in a *local* manner; that is, all nodes should be able to perform high quality prediction immediately without any extra communication. Third, the algorithm has to have a *low communication complexity*; both in terms of the number of messages sent, and the size of these messages as well. Privacy preservation is also one of our main goals, although in this chapter we do not analyze this aspect explicitly.

The original P2PEGASOS algorithm presented in the previous chapter has two main cornerstones: the implementation of the random walk and an online learning algorithm. Here we extend these with a third component called ensemble component. That is, we consider model combination techniques which implement a distributed "virtual" ensemble learning method similar to bagging. In this we in effect calculate a weighted voting over an exponentially increasing number of linear models.

Our particular contributions include the following: (1) we introduce a novel, efficient distributed ensemble learning method for linear models which virtually combines an exponentially increasing number of linear models; and (2) we provide a theoretical and empirical analysis of the convergence properties of the method in various scenarios.

The results of this chapter are based on our previous work [94].

7.1 Fully Distributed Data

Our system and data model is identical with the one described in the previous chapter (for more details, please see Sec. 2.2 and Sec. 6.1). That is, we assume the network consists of a huge number of nodes that communicate with each other using messaging. Additionally, we expect that an instance of NEWSCAST protocol as peer sampling is available. Related to the data distribution, we assume that each node has a single feature vector that cannot be moved to a server or to other nodes. Since this model is not usual in the data mining community, we elaborate on the motivation and the implications of the model.

In the distributed computing literature the fully distributed data model is typical. In the past decade, several algorithms have been proposed to calculate distributed aggregation queries over fully distributed data, such as the average, the maximum, and the network size (e.g., [20, 62, 118]). Here, the assumption is that every node stores only a single record, for example, a sensor reading. The motivation for not collecting raw data but processing it in place is mainly to achieve robustness and adaptivity through not relying on any central servers. In some systems, like in sensor networks or mobile ad hoc networks, the physical constraints on communication also prevent the collection of the data.

An additional motivation for not moving data is *privacy preservation*, where local data is not revealed in its raw form, even if the computing infrastructure made it possible. This is especially important in smart phone applications [3, 75, 98] and in P2P social networking [38], where the key motivation is giving the user full control over personal data. In these applications it is also common for a user to contribute only a single record, for example, a personal profile, a search history, or a sensor reading by a smart phone.

Clearly, in P2P smart phone applications and P2P social networks, there is a need for more complex aggregation queries, and ultimately, for data models, to support features such as recommendations and spam filtering, and to make the system more robust with the help of, for example, distributed intruder de-

tection. In other fully distributed systems data models are also important for monitoring and control. Motivated by the emerging need for building complex data models over fully distributed data in different systems, we work with the abstraction of fully distributed data, and we aim at proposing generic algorithms that are applicable in all compatible systems.

In the fully distributed model, the requirements of an algorithm also differ from those of parallel data mining algorithms. Here, the decision factor is the cost of message passing. Besides, the number of messages each node is allowed to send in a given time window is limited, so computation that is performed locally has a cost that is typically negligible when compared to communication delays. For this reason, prediction performance has to be investigated *as a function of the number of messages sent*, as opposed to wall clock time. Since communication is crucially important, evaluating robustness to communication failures, such as message delay and message loss, also gets a large emphasis.

The approach we present here is applicable successfully also when each node stores many records (and not only one); but its advantages to known approaches to P2P data mining become less significant, since communication plays a smaller role when local data is already usable to build reasonably good models. In the following we focus on the fully distributed model.

7.2 Background and Related Work

The main aspects of the supervised learning (Sec. 2.1), the SVM-based learning (Sec. 2.1.1), and the fully distributed learning (Sec. 6.3) are mostly covered in previous chapters. Thus, we focus on the discussion of the ensemble learning techniques only.

In the applied data mining, it is often hard to find a good model that fits the data well, even if the model space contains such a model for the particular problem. *Ensemble learners* combine multiple supervised models to form a probably better one. That is, the ensemble learning provides techniques for combining many learners, usually referred to as *weak learners*, in an attempt to produce a final learner (*strong learner*). Several ensemble techniques are

Algorithm 7.9 Generalized Gossip Learning Scheme

```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     m ← modelCache.freshest()
6:     send m to p
7: end loop
8: procedure ONRECEIVEMODEL(m)
9:     cm ← createModel(m, lastModel)
10:    modelCache.add(cm)
11:    lastModel ← m
12: end procedure
```

known, like bagging, boosting, and the Bayesian Model Averaging. In practice, applying these techniques can improve the prediction performance pretty well, however some of them are theoretically investigated as well.

Turning to investigate the distributed setting, most distributed large scale algorithms apply some form of ensemble learning to combine models learnt over different samples of the training data. Rokach presents a survey of ensemble learning methods [105]. We apply a method for combining the models in the network that is related to both bagging [21] and "pasting small votes" [22]: when the models start their random walk, initially they are based on non-overlapping small subsets of the training data due to the large scale of the system (the key idea behind pasting small votes) and as time goes by, the sample sets grow, approaching the case of bagging (although the samples that belong to different models will not be completely independent in our case).

7.3 Gossip Learning: the Basic Idea

Alg. 7.9 provides a slightly generalized skeleton of the gossip learning approach (compared to the one proposed in the previous chapter, in Alg. 6.6). As before, the same algorithm runs at each node in the network. Similarly, the algorithm consists of two distinct parts: an active loop of periodic activity (between lines 1-7), and a method to handle incoming models (function ONRECEIVEMODEL starting at line 8).

The active part is mainly responsible for performing the periodic activities. First, at line 1, the implementation tries to load the value of the *modelCache* from the previous run meaning that the nodes hold their model while they are

offline. If it does not exist—that is, it is the first time when the protocol is used by the node—a new model is initialized by adding the single data point available on the node. This behavior is encoded within the function INITMODEL which is detailed previously in Alg. 6.7. Then, the main loop of the active part is followed. That is, in roughly each Δ time moment (line 3), the protocol selects a random neighbor (at line 4) using the peer sampling service, and sends a copy of its freshest model, stored in the *modelCache*, to that neighbor (at line 6). Basically, this is the way how the random walk of the models is implemented.

We make no assumptions about either the synchrony of the loops at the different nodes or the reliability of the messages. We do assume that the length of the period of the loop, Δ, is the same at all nodes. However, during the evaluations Δ was modeled as a random variable $\mathcal{N}(\mu, \sigma)$ where the parameters are $\mu = \Delta$ and $\sigma^2 = \Delta/10$. For simplicity, here we assume that the active loop is initiated at the same time at all the nodes, and we do not consider any stopping criteria, so the loop runs indefinitely. The assumption about the synchronized start allows us to focus on the convergence properties of the algorithm, but it is not a crucial requirement in practical applications.

The message handler function of the learning system, ONRECEIVEMODEL, runs when a node receives a model. In this function, the received model is passed through the abstract function CREATEMODEL together with the previously received, non-updated model stored in the variable *lastModel* (at line 9). The various instantiations of the function CREATEMODEL result different ensemble components that will be discussed later in this section. The resulted new model, which is based on the above mentioned two models (ensemble aspect) and the sample hold by the node (update functionality), is added to the *modelCache* (at line 10). That is, the freshest element of the model cache is a combination of the last two received models updated by the training sample available locally. The cache has a fixed size. When the cache is full, the model stored for the longest time is replaced by the newly added model. The cache provides a pool of recent models that can be used to implement, for example, voting based prediction. We discuss this possibility in Section 7.5. Finally the received model m is stored in variable *lastModel* for further processing (at line 11).

Algorithm 7.10 CREATEMODEL: three implementations

```
1: procedure CREATEMODELRW(m_1, m_2)
2:     return update(m_1)
3: end procedure
4: procedure CREATEMODELMU(m_1, m_2)
5:     return update(merge(m_1, m_2))
6: end procedure
7: procedure CREATEMODELUM(m_1, m_2)
8:     return merge(update(m_1),update(m_2))
9: end procedure
```

The algorithm contains abstract methods that can be implemented in different ways to obtain a concrete learning algorithm. The main placeholders are SELECTPEER and CREATEMODEL. As before, the method SELECTPEER is the interface for the peer sampling service, as described in the previous chapters (in Section 2.2). Again, we use the NEWSCAST algorithm [64] as a gossip-based implementation of peer sampling.

The core of the approach is the function CREATEMODEL. Its task is to create a new updated model based on locally available information—the two models received most recently, and the local single training data record—to be sent on to a random peer. Algorithm 7.10 lists three implementations that are still abstract. They represent those three possible ways of breaking down the task that we will study in this chapter.

The abstract method UPDATE can represent an arbitrary online learning mechanism—the second main component of our framework besides peer sampling—that updates the model based on one example (the locally available training example of the node). Here we focus on two online learning components only, the Adaline Perceptron model [122] as a toy example, and the PegasosSVM model [108], which is a sophisticated, SVM-based learning algorithm. The procedure CREATEMODELRW implements the case where models independently perform random walks over the network. This implementation is identical with the gossip-based SVM approach introduced in the previous chapter. Here we use this algorithm as a baseline.

The remaining two variants apply a method called MERGE, either before

the update (MU) or after it (UM). The method MERGE helps implement the third component: ensemble learning. A *completely impractical* example for an implementation of MERGE is the case where the model space consists of all the sets of basic models of a certain type. Then MERGE can simply merge the two input sets, UPDATE can update all the models in the set, and prediction can be implemented via, for example, majority voting (for classification) or averaging the predictions (for regression). With this implementation, all nodes would collect an exponentially increasing set of models, allowing for a much better prediction after a much shorter learning time in general than based on a single model [21, 22], although the learning history for the members of the set would not be completely independent.

This implementation is of course impractical because the size of the messages in each cycle of the main loop would increase exponentially. Our main contribution is to discuss and analyze a special case: linear models. For linear models we will propose an algorithm where the message size can be kept *constant*, while producing the same (or similar) behavior as the impractical implementation above. The subtle difference between the MU and UM versions will also be discussed.

Let us close this section with a brief analysis of the cost of the algorithm in terms of computation and communication. As of communication: each node in the network sends exactly one message in each Δ time unit. The size of the message depends on the selected hypothesis space; normally it contains the parameters of a single model. In addition, the message also contains a small constant number of network addresses as defined by the NEWSCAST protocol (typically around 20). The computational cost is one or two update steps in each Δ time units for the UM or the MU variants, respectively. The exact cost of this step depends on the selected online learner.

7.4 Merging Linear Models through Averaging

The key observation we make is that in a linear hypothesis space, in certain cases the voting-based prediction is equivalent to a single prediction by the *average* of the models that participate in the voting. Furthermore, updating a

set of linear models and then averaging them is sometimes equivalent to averaging the models first, and then updating the resulting single model. These observations are valid in a strict sense only in special circumstances. However, our intuition is that even if this key observation holds only in a heuristic sense, it still provides a valid heuristic explanation of the behavior of the resulting averaging-based merging approach.

In the following we first give an example of a case where there is a strict equivalence of averaging and voting to illustrate the concept, and subsequently we discuss and analyze a practical and competitive algorithm, where the correspondence of voting and averaging is only heuristic in nature.

7.4.1 The Adaline Perceptron

We consider here the Adaline perceptron [122], which arguably has one of the simplest update rules due to its linear activation function. Without loss of generality, we ignore the bias term. The error function to be optimized is defined as

$$E_{\mathbf{x}}(\mathbf{w}) = \frac{1}{2}(y - \mathbf{w}^T\mathbf{x})^2 \tag{7.1}$$

where $\mathbf{w}$ is the linear model, and $(\mathbf{x}, y)$ is a training example ($\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$, $y \in \{-1, 1\}$) (that is, it is a supervised binary classification problem, for more explanation, please see Sec. 2.1). The gradient at $\mathbf{w}$ for $\mathbf{x}$ is given by

$$\nabla_{\mathbf{w}} = \frac{\partial E_{\mathbf{x}}(\mathbf{w})}{\partial \mathbf{w}} = -(y - \mathbf{w}^T\mathbf{x})\mathbf{x} \tag{7.2}$$

that defines the learning rule for $(\mathbf{x}, y)$ by

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta(y - {\mathbf{w}^{(k)}}^T\mathbf{x})\mathbf{x}, \tag{7.3}$$

where η is the learning rate. In this case it is a constant.

Now, let us assume that we are given a set of models $\mathbf{w}_1, \ldots, \mathbf{w}_m$, and let us define $\bar{\mathbf{w}} = (\mathbf{w}_1 + \ldots + \mathbf{w}_m)/m$ for a given $m \in \mathbb{N}$. In the case of a regression problem, the prediction for a given point $\mathbf{x}$ and model $\mathbf{w}$ is $\mathbf{w}^T\mathbf{x}$. It is not hard

to see that

$$h(\mathbf{x}) = \bar{\mathbf{w}}^T\mathbf{x} = \frac{1}{m}\left(\sum_{i=1}^{m} w_i\right)^T \cdot \mathbf{x} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{w}_i^T\mathbf{x}, \tag{7.4}$$

which means that the voting-based prediction is equivalent to prediction based on the average model.

In the case of classification, the equivalence does not hold for all voting mechanisms. But it is easy to verify that in the case of a weighted voting approach, where vote weights are given by $|\mathbf{w}^T\mathbf{x}|$, and the votes themselves are given by $\operatorname{sgn}(\mathbf{w}^T\mathbf{x})$, the same equivalence holds:

$$\begin{aligned} h(\mathbf{x}) = \operatorname{sgn}(\bar{\mathbf{w}}^T\mathbf{x}) = \operatorname{sgn}\left(\left(\frac{1}{m}\sum_{i=1}^{m}\mathbf{w}_i\right)^T \cdot \mathbf{x}\right) = \\ = \operatorname{sgn}\left(\frac{1}{m}\sum_{i=1}^{m}\mathbf{w}_i^T \cdot \mathbf{x}\right) = \operatorname{sgn}\left(\frac{1}{m}\sum_{i=1}^{m}|\mathbf{w}_i^T\mathbf{x}|\operatorname{sgn}(\mathbf{w}_i^T\mathbf{x})\right). \end{aligned} \tag{7.5}$$

A similar approach to this weighted voting mechanism has been shown to improve the performance of simple vote counting [11]. Our preliminary experiments also support this.

In a very similar manner, it can be shown that updating $\bar{\mathbf{w}}$ using an example $(\mathbf{x}, y)$ is equivalent to updating all the individual models $\mathbf{w}_1, \ldots, \mathbf{w}_m$ and then taking the average:

$$\bar{\mathbf{w}} + \eta(y - \bar{\mathbf{w}}^T\mathbf{x})\mathbf{x} = \frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{w}_i + \eta(y - \mathbf{w}_i^T\mathbf{x})\mathbf{x}\right). \tag{7.6}$$

The above properties lead to a rather important observation. If we implement our gossip learning skeleton using Adaline, as shown in Algorithm 7.11, then the resulting algorithm behaves exactly as if all the models were simply stored and then forwarded, resulting in an exponentially increasing number of models contained in each message, as described in Section 7.3. That is, averaging effectively reduces the exponential message complexity to transmitting a *single* model in each cycle independently of time, yet we enjoy the benefits of the aggressive, but impractical approach of simply replicating all the models

Algorithm 7.11 Adaline update, and merging

```
1: procedure UPDATEADALINE(m)
2:     m.w ← m.w + η(y − ⟨m.w, x⟩)x
3:     return m
4: end procedure
5: procedure MERGE(m1,m2)
6:     m.t ← max(m1.t, m2.t)
7:     m.w ← (m1.w + m2.w)/2
8:     return m
9: end procedure
```

and using voting over them for prediction.

It should be mentioned that—even though the number of „virtual" models is growing exponentially fast—the algorithm is not equivalent to bagging over an exponential number of independent models. In each gossip cycle, there are only N independent updates occurring in the system overall (where N is the number of nodes), and the effect of these updates is being aggregated rather efficiently. In fact, as we will see in Section 7.5, bagging over N independent models actually outperforms the gossip learning algorithms.

7.4.2 Pegasos

Here we discuss the adaptation of Pegasos SVM [108] used for classification into our gossip-based learning system. The specific learning components required for the adaptation are detailed in the previous chapter and shown in Alg. 6.7, where the method UPDATEMODEL is taken from [108]. A brief explanation of the algorithm can be found in Sec. 6.4 as well. For a complete implementation of the framework, one also needs to select an implementation of CREATEMODEL from Alg 7.10. In the following, the three versions of a complete Pegasos–based implementation defined by these options will be referred to as P2PEGASOSRW, P2PEGASOSMU, and P2PEGASOSUM. Here P2PEGASOSRWis considered as a baseline (in which the ensemble aspect is ignored). This implementation is identical with that discussed in the previous chapter, and there referred to simply as P2PEGASOS.

The main difference between the Adaline perceptron and Pegasos is the *context dependent update rule* that is different for correctly and incorrectly classified examples. Due to this difference, there is no strict equivalence between averaging and voting, as in the case of the previous section. To see this, con-

sider two models, $\mathbf{w}_1$ and $\mathbf{w}_2$, and a training example $(\mathbf{x}, y)$, and let $\bar{\mathbf{w}} = (\mathbf{w}_1 + \mathbf{w}_2)/2$. In this case, updating $\mathbf{w}_1$ and $\mathbf{w}_2$ first, and then averaging them results in the same model as updating $\bar{\mathbf{w}}$ if and only if both $\mathbf{w}_1$ and $\mathbf{w}_2$ classify $\mathbf{x}$ in the same way (correctly or incorrectly). This is because when updating $\bar{\mathbf{w}}$, we virtually update both $\mathbf{w}_1$ and $\mathbf{w}_2$ in the same way, irrespective of how they classify $\mathbf{x}$ individually.

This seems to suggest that P2PEGASOSUM is a better choice. We will test this hypothesis experimentally in Section 7.5, where we will show that, surprisingly, it is not always true. The reason could be that P2PEGASOSMU and P2PEGASOSUM are in fact very similar when we consider the entire history of the distributed computation, as opposed to a single update step. The histories of the models define a directed acyclic graph (DAG), where the nodes are merging operations, and the edges correspond to the transfer of a model from one node to another. In both cases, there is one update corresponding to each edge: the only difference is whether the update occurs on the source node of the edge or on the target. Apart from this, the edges of the DAG are the same for both methods. Hence we see that P2PEGASOSMU has the favorable property that the updates that correspond to the incoming edges of a merge operation are done using independent samples, while for P2PEGASOSUM they are performed with the same example. Thus, P2PEGASOSMU guarantees a greater independence of the models.

In the following, we present our theoretical results for both P2PEGASOSMU and P2PEGASOSUM. We note that these results do not assume any coordination or synchronization; they are based on a fully asynchronous communication model. First, let us formally define the optimization problem at hand, and let us introduce some notation.

Let $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq n, x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}\}$ be a distributed training set (which defines a supervised binary classification problem, for more details, please review Sec. 2.1) with one data point at each network node. Let $f : \mathbb{R}^d \to \mathbb{R}$ denote here the objective function of the Pegasos SVM shown in Eq. 6.1 (applying the Hinge loss in the general form of the SVM optimization problem shown in Eq. 2.6). Note that f is strongly convex with a parameter λ [108]. Let $\mathbf{w}^\star$ denote the global optimum of f. For a fixed data point $(\mathbf{x}_i, y_i)$

we define

$$f_i(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i), \tag{7.7}$$

which is used to derive the update rule for the Pegasos algorithm. Obviously, f_i is λ–strongly convex as well, since it has the same form as f with $m = 1$.

The update history of a model can be represented as a binary tree, where the nodes are models, and the edges are defined by the direct ancestor relation. Let us denote the direct ancestors of $\mathbf{w}^{(i+1)}$ as $\mathbf{w}_1^{(i)}$ and $\mathbf{w}_2^{(i)}$. These ancestors are averaged and then updated to obtain $\mathbf{w}^{(i+1)}$ (assuming the MU variant is applied). Let the sequence $\mathbf{w}^{(0)}, \ldots, \mathbf{w}^{(t)}$ be defined as the path in this history tree, for which

$$\mathbf{w}^{(i)} = \text{argmax}_{\mathbf{w} \in \{\mathbf{w}_1^{(i)}, \mathbf{w}_2^{(i)}\}} \|\mathbf{w} - \mathbf{w}^\star\|, \;\; i = 0, \ldots, t-1. \tag{7.8}$$

This sequence is well defined. Let $(\mathbf{x}_i, y_i)$ denote the training example, that was used in the update step that resulted in $\mathbf{w}^{(i)}$ in the series defined above.

Theorem 1 (P2PEGASOSMU convergence)**.** We assume that (1) each node receives an incoming message after any point in time within a finite time period (eventual update assumption), (2) there is a subgradient ∇ of the objective function such that $\|\nabla_{\mathbf{w}}\| \leq G$ for every $\mathbf{w}$. Then,

$$\frac{1}{t}\sum_{i=1}^{t} f_i(\bar{\mathbf{w}}^{(i)}) - f_i(\mathbf{w}^\star) \leq \frac{G^2(\log(t)+1)}{2\lambda t} \tag{7.9}$$

where $\bar{\mathbf{w}}^{(i)} = (\mathbf{w}_1^{(i)} + \mathbf{w}_2^{(i)})/2$.

Proof. During the running of the algorithm, let us pick any node on which at least one subgradient update has been already performed. There is such a node eventually, due to the eventual update assumption. Let the model currently stored at this node be $\mathbf{w}^{(t+1)}$.

We know that $\mathbf{w}^{(t+1)} = \bar{\mathbf{w}}^{(t)} - \nabla^{(t)}/(\lambda t)$, where $\bar{\mathbf{w}}^{(t)} = (\mathbf{w}_1^{(t)} + \mathbf{w}_2^{(t)})/2$ and where $\nabla^{(t)}$ is the subgradient of f_t. From the λ-convexity of f_t it follows that

$$f_t(\bar{\mathbf{w}}^{(t)}) - f_t(\mathbf{w}^\star) + \frac{\lambda}{2}\|\bar{\mathbf{w}}^{(t)} - \mathbf{w}^\star\|^2 \leq (\bar{\mathbf{w}}^{(t)} - \mathbf{w}^\star)^T \cdot \nabla^{(t)}. \tag{7.10}$$

On the other hand, the following inequality is also true, following from the

definition of $\bar{\mathbf{w}}^{(t+1)}$, G and some algebraic rearrangements:

$$(\bar{\mathbf{w}}^{(t)} - \mathbf{w}^\star)^T \cdot \nabla^{(t)} \leq \frac{\lambda t}{2}\|\bar{\mathbf{w}}^{(t)} - \mathbf{w}^\star\|^2 - \frac{\lambda t}{2}\|\mathbf{w}^{(t+1)} - \mathbf{w}^\star\|^2 + \frac{G^2}{2\lambda t}. \quad (7.11)$$

Moreover, we can bound the distance of $\bar{\mathbf{w}}^{(t)}$ from $\mathbf{w}^\star$ with the distance of the ancestor of $\bar{\mathbf{w}}^{(t)}$ that is further away from $\mathbf{w}^\star$ with the help of the Cauchy–Bunyakovsky–Schwarz inequality:

$$\|\bar{\mathbf{w}}^{(t)} - \mathbf{w}^\star\|^2 = \left\| \frac{\mathbf{w}_1^{(t)} - \mathbf{w}^\star}{2} + \frac{\mathbf{w}_2^{(t)} - \mathbf{w}^\star}{2} \right\|^2 \leq \|\mathbf{w}^{(t)} - \mathbf{w}^\star\|^2. \quad (7.12)$$

From Eqs. 7.10, 7.11, 7.12 and the bound on the subgradients, we derive

$$f_t(\bar{\mathbf{w}}^{(t)}) - f_t(\mathbf{w}^\star) \leq \frac{\lambda(t-1)}{2}\|\mathbf{w}^{(t)} - \mathbf{w}^\star\|^2 - \frac{\lambda t}{2}\|\mathbf{w}^{(t+1)} - \mathbf{w}^\star\|^2 + \frac{G^2}{2\lambda t}. \quad (7.13)$$

Note that this bound also holds for $\mathbf{w}^{(i)}$, $1 \leq i \leq t$. Summing up both sides of these t inequalities, we get the following bound:

$$\sum_{i=1}^{t} f_i(\bar{\mathbf{w}}^{(i)}) - f_i(\mathbf{w}^\star) \leq -\frac{\lambda t}{2}\|\mathbf{w}^{(t+1)} - \mathbf{w}^\star\|^2 + \frac{G^2}{2\lambda}\sum_{i=1}^{t}\frac{1}{i} \leq \frac{G^2(\log(t)+1)}{2\lambda}, \quad (7.14)$$

from which the theorem follows after division by t. □

The bound in Eq. 7.14 is analogous to the bound presented in [108] in the analysis of the PEGASOS algorithm. It basically means that the average error tends to zero. To be able to show that the limit of the process is the optimum of f, it is necessary that the samples involved in the series are uniform random samples [108]. Investigating the distribution of the samples is left to future work; but we believe that the distribution closely approximates uniformity for a large t, given the uniform random peer sampling that is applied.

For P2PEGASOSUM, an almost identical derivation leads us to a similar result (omitted due to the similarity).

7.5 Experimental Results

Here we experiment with two algorithms: P2PEGASOSUM and P2PEGASOSMU. In addition, to shed light on the behavior of these algorithms, we include a number of baseline methods as well.

7.5.1 Experimental Setup

Data Sets. We used three different data sets: Reuters [48], Spambase, and the Malicious URLs [78] data sets, which were obtained from the UCI database repository [44]. These data sets are of different types including small and large sets containing a small or large number of features. We used exactly these data sets in the previous chapter; there, in Sec. 6.5.1, you can find a detailed description of how we preprocessed this data. We just emphasize here that all these preprocessing steps can be done in a gossip-based manner, however, we performed it offline. One can check Table 6.1, which summarizes the main properties of the applied data sets.

Modeling failure. Just like in the previous chapter, we use the event based engine of the PeerSim [85] and, as we mentioned previously, we apply the NEWSCAST protocol (for further description, please see Sec. 2.2) as peer sampling service implementation.

In a set of experiments, referred to as *failure scenarios*, we model extreme message drop, message delay, and churn. The exact setting of the failure scenarios are identical with those applied in the experiments of the previous chapter. In this way, we get comparable results with those. Here we just enumerate through the settings of the failure scenarios quickly. For more details on how these parameter values were generated (especially related to the churn model), what their exact meaning is, please visit Sec. 6.5.1. In these scenarios the drop probability is set to be 0.5. Message delay is modeled as a uniform random variable from the interval $[\Delta, 10\Delta]$, where Δ is the waiting period of our learning protocol shown in Alg. 7.9. We model churn based on the probabilistic models proposed in [111].

Evaluation metric. Similar to the previous chapter, the main evaluation metric here we focus on is prediction error measure on an independent (that is, it is unknown to the training algorithm) set of examples. To measure prediction error, we split the datasets into training sets and validation sets before the training process was started. The splits we applied are exactly the same as those performed in the previous chapter, which are shown in Table 6.1. In our experiments with P2PEGASOSMU and P2PEGASOSUM, we track the misclassification ratio over the validation set of 100 randomly selected peers. The misclassification ratio of a model is calculated by dividing the number of the misclassified validation examples by the number of all test examples. This is the so-called 0-1 error.

For the baseline algorithms we used all the available models for calculating the error rate, which equals the number of training samples. From the Malicious URLs database we used only 10,000 examples selected at random, to make the evaluation computationally feasible. Note that we found that increasing the number of examples beyond 10,000 does not result in a noticeable difference in the observed behavior.

Here we calculated the similarities between the models circulating in the network just like in the previous chapter. The metric we used was the cosine similarity measure. First, we calculated the similarity between all pairs of models, and calculated the average.

Baseline Algorithms. The first baseline we use is P2PEGASOSRW[1], which is equivalent to the algorithm P2PEGASOS proposed in the previous chapter. If there is no message drop or message delay, then this is equivalent to the Pegasos algorithm, since in cycle t all peers will have models that are the result of Pegasos learning on t random examples. In the case of message delay and message drop failures, the number of samples will be less than t, as a function of the drop probability and the delay.

We also examine two variants of *weighted bagging*. The first variant (WB1) is

[1]We changed the name of the algorithm to make it clear which ensemble component was applied.

defined as

$$h_{WB1}(\mathbf{x}, t) = \text{sgn}\left(\sum_{i=1}^{N} {\mathbf{w}_i^{(t)}}^T \mathbf{x}\right), \tag{7.15}$$

where N is the number of nodes in the network, and the linear models $\mathbf{w}_i^{(t)}$ are learnt with Pegasos over an independent sample of size t of the training data. This baseline algorithm can be thought of as the ideal utilization of the N independent updates performed in parallel by the N nodes in the network in each cycle. The gossip framework introduces dependencies among the models, so its performance can be expected to be worse.

In addition, in the gossip framework a node has influence from only 2^t models on average in cycle t. To account for this handicap, we also use a second version of weighted bagging (WB2):

$$h_{WB2}(\mathbf{x}, t) = \text{sgn}\left(\sum_{i=1}^{\min(2^t, N)} {\mathbf{w}_i^{(t)}}^T \mathbf{x}\right). \tag{7.16}$$

Note that initially this version can be expected to perform worse than WB1, but still better than the gossip framework, because WB2 uses at least as much updates as the gossip framework, but in a more independent fashion. After $\log N$ cycles the two versions of weighted bagging become identical.

The weighted bagging variants described above are not practical alternatives, these algorithms serve as a baseline only. The reason is that an actual implementation would require N independent models for prediction. This could be achieved by P2PEGASOSRW with a distributed prediction, which would impose a large cost and delay for every prediction. This could also be achieved by all nodes running up to $O(N)$ instances of P2PEGASOSRW, and using the $O(N)$ local models for prediction; this is not feasible either. In sum, the point that we want to make is that our gossip algorithm approximates WB2 quite well using only a single message per node in each cycle, due to the technique of merging models.

The last baseline algorithm we experiment with is PERFECT MATCHING. In this algorithm we replace the peer sampling component of the gossip framework: instead of all nodes picking random neighbors in each cycle, we create a

random perfect matching among the peers so that every peer receives exactly one message. Our hypothesis was that—since this variant increases the efficiency of mixing—it will maintain a higher diversity of models, and so a better performance can be expected due to the "virtual bagging" effect we explained previously. Note that this algorithm is not intended to be practical either.

Using the local models for prediction. An important aspect of our protocol is that every node has at least one model available locally, and thus all the nodes can perform a prediction. Moreover, since the nodes can remember the models that pass through them at no communication cost, we cheaply implement a simple voting mechanism, where nodes will use more than one model to make predictions. Algorithm 6.8 shows the procedures used for prediction in the original case, and in the case of voting.

Here the vector x is the unseen example to be classified. In the case of linear models, the classification is simply the sign of the inner product with the model, which essentially describes on which side of the hyperplane the given point lies. In our experiments we used a cache of size 10.

7.5.2 Results and Discussion

The experimental results for prediction without local voting are shown in Figures 7.1 and 7.2.

Note that all variants can be mathematically proven to converge to the same result, so the difference is in convergence speed only. Bagging can temporarily outperform a single instance of Pegasos, but after enough training samples, all models become almost identical, so the advantage of voting disappears.

In Figure 7.1 we can see that our hypothesis about the relationship of the performance of the gossip algorithms and the baselines is validated: the standalone Pegasos algorithm is the slowest, while the two variants of weighted bagging are the fastest. P2PEGASOSMU approximates WB2 quite well, with some delay, so we can use WB2 as a heuristic model of the behavior of the algorithm. Note that the convergence is several orders of magnitude faster than that of Pegasos (the plots have a logarithmic scale).

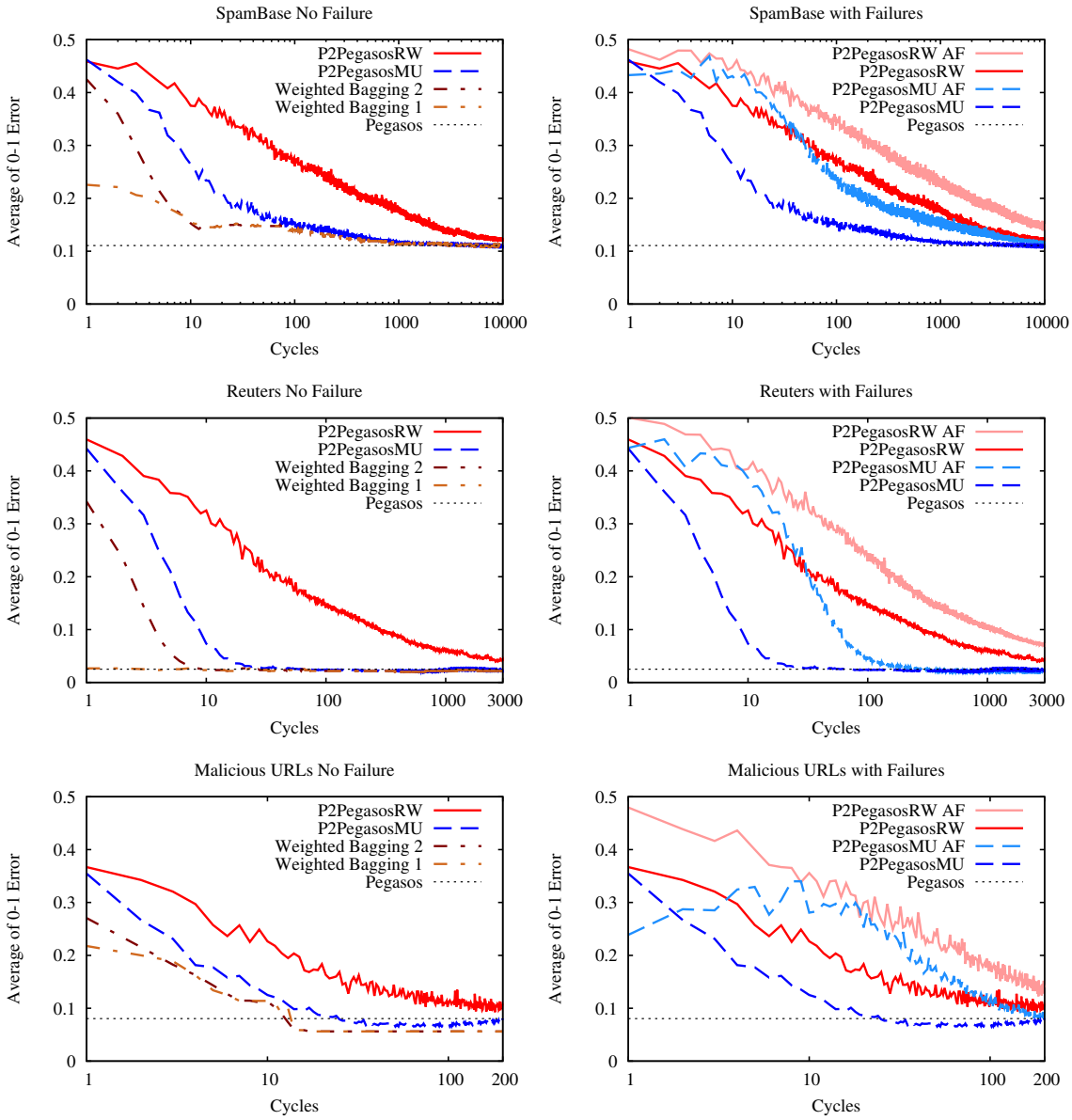

Figure 7.1: Experimental results without failure (left column) and with extreme failure (right column). AF means all possible failures are modeled.

Figure 7.1 also contains results from our extreme failure scenario. We can observe that the difference in convergence speed is mostly accounted for by the increased message delay. The effect of the delay is that all messages wait 5 cycles on average before being delivered, so the convergence is proportion-

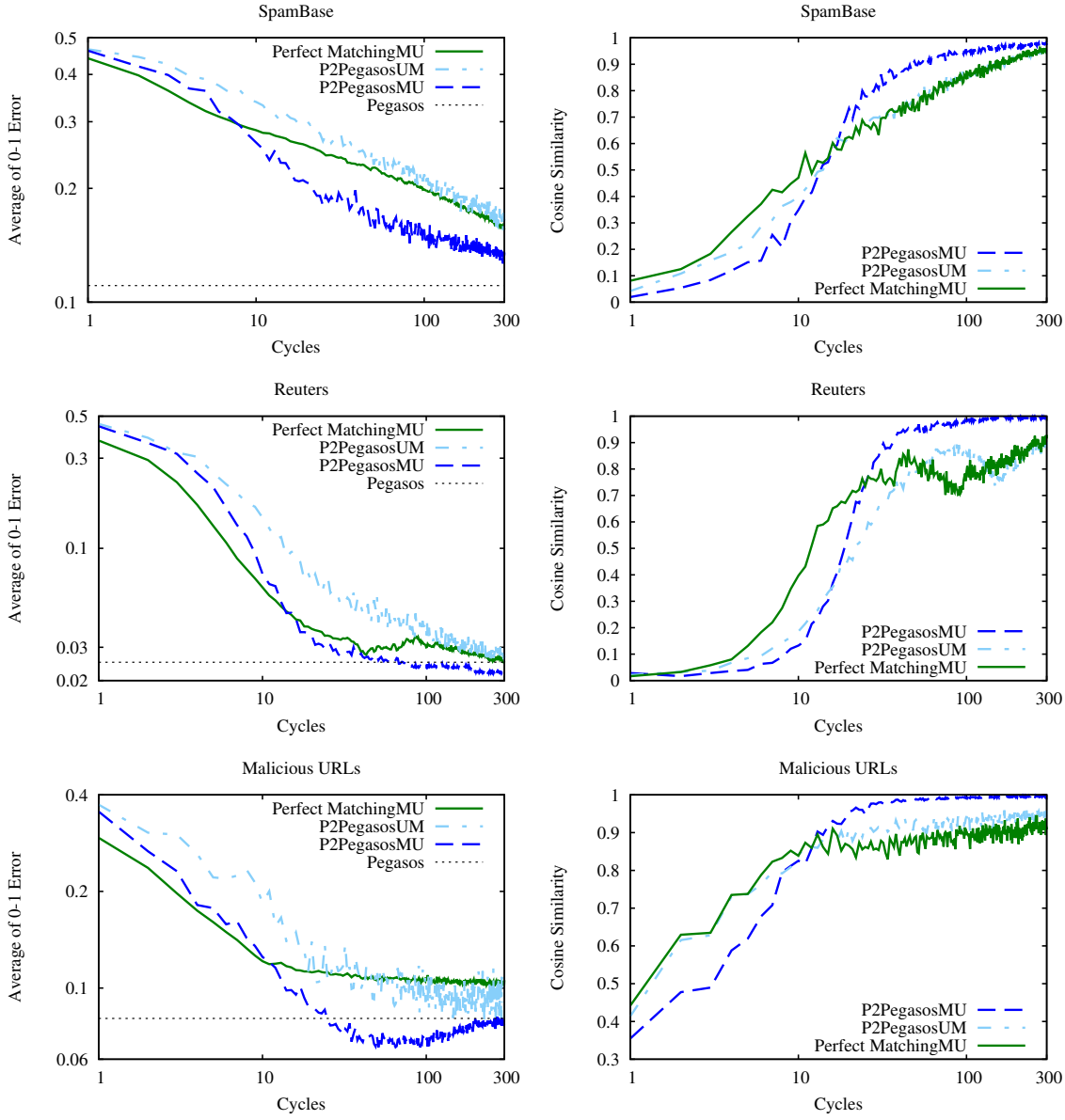

Figure 7.2: Prediction error (left column) and model similarity (right column) with PERFECT MATCHING and P2PEGASOSUM.

ally slower. In addition, half of the messages get lost too, which adds another factor of about 2 to the convergence speed. Apart from slowing down, the algorithms still converge to the correct value despite the extremely unreliable environment, as expected.

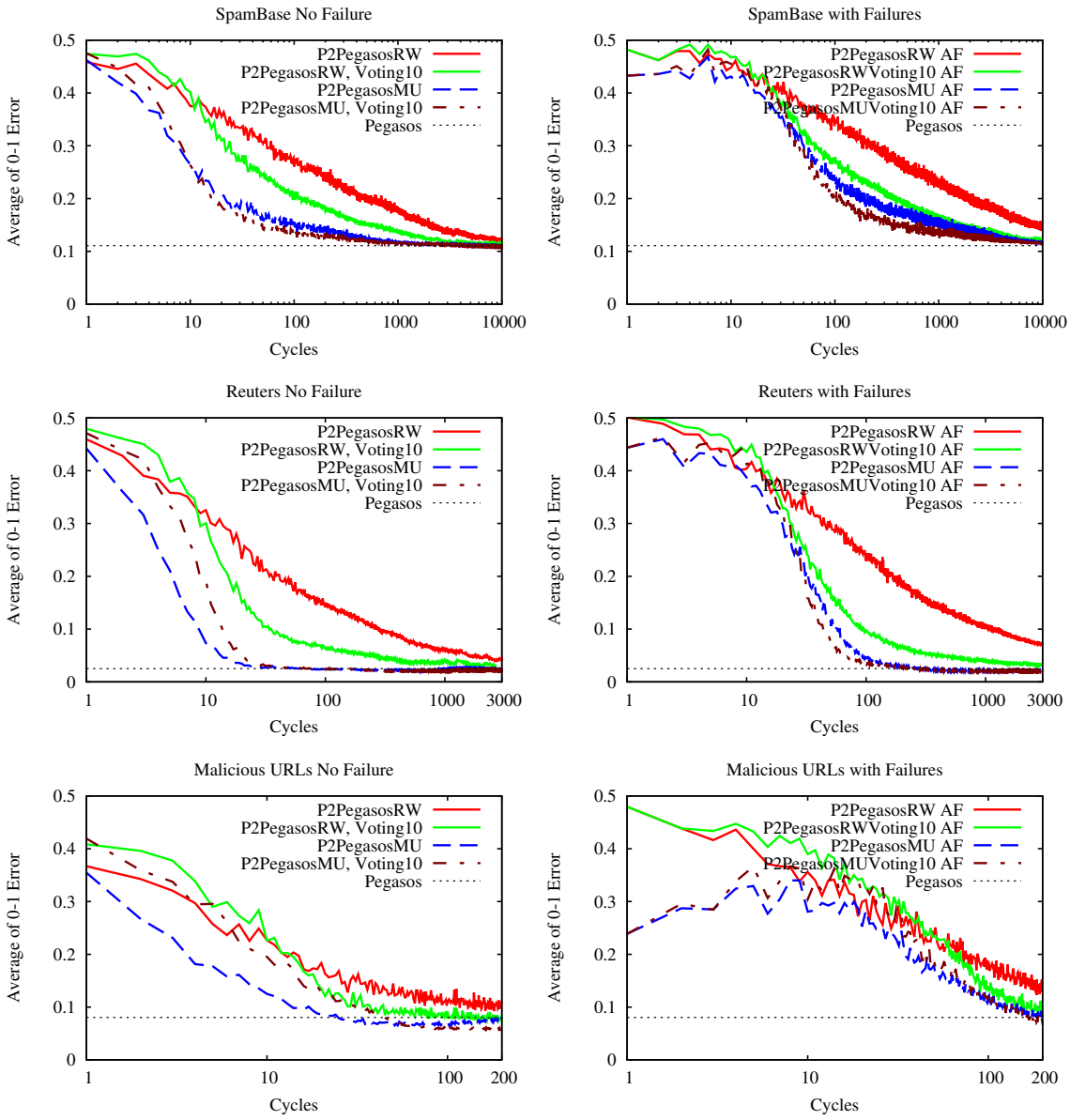

Figure 7.3: Experimental results applying local voting without failure (left column) and with extreme failure (right column).

Figure 7.2 illustrates the difference between the UM and MU variants. Here we model no failures. In Section 7.4.2, we pointed out that—although the UM version looks favorable when considering a single node—when looking at the

full history of the learning process, P2PEGASOSMU maintains more independence between the models. Indeed, the MU version clearly performs better according to our experiments. We can also observe that the UM version shows a lower level of model similarity in the system, which probably has to do with the slower convergence.

In Figure 7.2 we can see the performance of the perfect matching variant of P2PEGASOSMU as well. Contrary to our expectations, perfect matching does not clearly improve performance, apart from the first few cycles. It is also interesting to observe that model similarity is correlated with prediction performance also in this case. We also note that in the case of the Adaline-based gossip learning implementation, perfect matching is clearly better than random peer sampling (not shown). This means that this behavior is due to the context-dependence of the update rule discussed in 7.4.2. Perfect matching also has an UM and an MU variant, and here UM indeed performs slightly better than MU, but the difference between the MU and UM variants is rather small (not shown).

The results with local voting are shown in Figure 7.3. The main conclusion is that voting results in a significant improvement when applied along with P2PEGASOSRW, the learning algorithm that does not apply merging. When merging is applied, the improvement is less dramatic. In the first few cycles, voting can result in a slight degradation of performance. This could be expected, since the models in the local caches are trained on fewer samples on average than the freshest model in the cache. Overall, since voting is for free, it is advisable to use it.

7.6 Conclusions

Here we proposed a general technique by introducing an ensemble learning component to learn P2PEGASOS models much faster than we proposed in the previous chapter. In sum, the basic idea of our proposal is that many Pegasos SVM models perform a random walk over the network, while being updated at every node they visit, and while being combined (merged) with other models they encounter. The algorithm was shown to be extremely robust to mes-

sage drop and message delay, furthermore, a very significant speedup was demonstrated with respect to the baseline PEGASOS algorithm detailed in the previous chapter due to the model merging technique and the prediction algorithm that is based on local voting.

Moreover, the algorithm makes it possible to compute predictions locally at every node in the network at any point in time, yet the message complexity is acceptable: every node sends one model in each gossip cycle. The main features that differentiate this approach from related work are the focus on fully distributed data and its modularity, generality, and simplicity.

An important promise of the approach is the support for privacy preservation, since data samples are not observed directly. Although in this chapter we did not focus on this aspect, it is easy to see that the only feasible attack is the multiple forgery attack [83], where the local sample is guessed based on sending specially crafted models to nodes and observing the result of the update step. This is very hard to do even without any extra measures, given that models perform random walks based on local decisions, and that merge operations are performed as well. This short informal reasoning motivates our ongoing work towards understanding and enhancing the privacy-preserving properties of gossip learning, while we extend the basic idea to tackle with different learning scenarios (like concept drift [53, 55]), and different models (like boosting models [54], multi–armed bandit models [114]).

The presented results are mainly based on our previous paper [94].

CHAPTER 8

Summary

8.1 Summary in English

The main goal of this thesis was to investigate the *adaptivity* of various support vector-based learning approaches. Along this, we focused on two distinct aspects of the adaptivity. First, we investigated the so-called algorithmic adaptivity of the support vector-based learners. That is, we examined how the basic idea of the algorithm family, the maximal margin heuristic, can be applied to handle tasks which cannot be tackled with the traditional formalisms of the SVMs. Second, in the latter part of the thesis, we gradually moved the focus from the algorithmic aspects to the system model aspects of the adaptivity. Here our main question was how we can adapt efficiently the SVMs to a specific system model, the fully distributed system model, which introduced a number of challenges, like the handling of network errors, user dynamics and the unbalanced load in the network.

In this chapter, we overview the main goals and results of this thesis by giving a brief summary of each chapter (Chapters 3-7). During the summarization, we reveal our contributions as well by giving a itemized list of the

key contributions of the given chapter.

8.1.1 VMLS-SVM for Time Series Forecasting

In Chapter 3, we investigated the problem of time series forecasting. That is, how we can extend the Least Squares SVMs to become more suitable algorithm for predicting the future values of a time series. Our proposal (the VMLS-SVM algorithm) introduced a weighted variance term in the objective function of the LS-SVM. This idea is based on the preliminary observation which says that if we have two time series forecasting models with the same prediction error, the one with the smaller variance results in a better overall performance aside from the overfitting. The proposed method can be considered as the generalization of the LS-SVM algorithm, which keeps all the advantages of the original algorithm, like the applicability of the kernel-trick, unique and linear solution. However, it introduces a new hyperparameter which makes the fine-tuning of the algorithm more complicated.

The first part of the chapter briefly overviewed the related approaches and introduced the basic properties of the original LS-SVM approach, then it gave a detailed description of the proposed method. It proposed a mechanism for handling the above mentioned increased complexity of parameter setting as well. A thorough empirical evaluation closed the chapter which points out that with appropriate parameter tuning the proposed method achieves a significantly better performance against a numerous state-of-the-art baseline algorithms measured on three different, widely used benchmark datasets.

The key contributions and their effects are:

- The theoretical and algorithmic introduction of the VMLS-SVM algorithm.
- Parameter optimization mechanism for the VMLS-SVM algorithm.
- The proposed algorithm outperforms significantly a number of baseline approaches on three widely used benchmark datasets.

8.1.2 Transformation-based Domain Adaptation

This chapter (Chapter 4) investigated the problem of domain adaptation on an opinion mining task. We proposed a general framework, called DOMAIN MAPPING LEARNER (DML) and two instantiations of this general idea: the first one is based on SVMs a source model and the second one applies the Logistic Regression (LR) as source model. The main idea of the general approach can be summarized as follows: it models the relation between the source and target domain by applying a model transformation mechanism which can be learnt by using labeled data of a very limited size taken from the target domain.

In the chapter, we briefly overviewed the related approaches for the task of domain adaptation. Then, we formally defined the problem and proposed our general, transformation-based approach, the DML algorithm. We introduced the novel instantiation of this abstract algorithm, based on SVMs and LR methods. Our experiment evaluations validated that our approach is capable of training models for the target domain which use a very limited number of labeled samples taken from the target domain. This phenomenon is even true when we have enough samples, but the baseline methods cannot generalize well. From this evaluation, we also concluded that the SVM-based instantiation is a more suitable choice since it is more robust than the LR-based variant.

The key contributions and their effects are:

- The transformation-based formalism of the problem of domain adaptation.
- The introduction of the general algorithm.
- The two instantiations of the general idea based on SVM and LR based models.
- The proposed algorithms result in models which have a better performance than the direct method (baseline), and two other state-of-the-art baseline algorithms (SCL and SCL-MI).

8.1.3 SVM Supported Distributed Recommendation

In this chapter (Chapter 5), we dealt with the problem of distributed recommendation. The goal of this chapter is twofold, first we described and motivated the process of inferring ratings from implicit user feedbacks, then we introduced two heuristic approaches (direct and time-shift based approaches) to overcome the problem. Here we presented an interesting use-case of the SVMs. We use them to validate our inferring heuristics indirectly and proved that—without any ground truth—the inferred dataset has some interesting properties, and the one generated based on the improved heuristics variant (time-shift based variant) has a more interesting inner structure. Second, we introduced a novel overlay management protocols which support the implementation of user-based collaborative filtering (CF) approaches in fully distributed systems, while keeping low the overall network load.

In the chapter, first we reviewed both the centralized and distributed CF approaches. Then, we introduced our inferring heuristics, described the validation methodology through learnability, and performed this validation applying the SMO SVM solver. Later, we pointed out that most of the CF dataset has almost power-law in-degree distribution which can cause serious issues in distributed setting. To overcome this problem, we introduced a bunch of overlay management approaches (random sampling based kNN approach, T-MAN based variants (GLOBAL, VIEW, BEST and PROPORTIONAL) and their randomized variants) and performed a thorough empirical evaluation against each other and a real world overlay management protocol called BUDDYCAST. We can draw multiple conclusions: the aggressive peer sampling can cause untreatable network load, while the proposed T-Man based approach with the GLOBAL peer selection strategy is a good choice considering that it has a fully uniform load distribution with an acceptable convergence speed.

The key contributions and their effects are:

- The inferring heuristics (direct and time-shift based approaches).
- Learnability based indirect validation technique.
- The FileList.org inferred recommender dataset [1].

- Overlay management approaches: random sampling based kNN approach, T-MAN based variants (GLOBAL, VIEW, BEST and PROPORTIONAL) and their randomized variants
- The GLOBAL peer selection based overlay management tool provides a good trade-off between the convergence speed and network load.

8.1.4 P2PEGASOS—A Fully Distributed SVM

This chapter (Chapter 6) focused on the problem of fully distributed data mining. That is, our goal here was to propose an SVM-based algorithm which performs in a fully distributed network realizing good quality models with an affordable communication complexity while assuming as little as possible about the underlying communication model. Our proposal, the P2PEGASOS algorithm, introduces a conceptually simple, yet powerful SVM implementation. The key of our contribution is that many models perform a random walk over the network while updating themselves applying an online update rule.

At the beginning of the chapter, we carefully defined the system and data model. Then, we overviewed the basic concept of the Pegasos SVM solver, and the related fully distributed machine learning approaches. After a detailed algorithmic description of our proposal, we turned to evaluate our approach. Here we considered a number of baseline algorithms (mainly centralized SVM implementations) and investigated the speed of the convergence in various scenarios including ones with extreme network failures. These experimental evaluations show that our approach is robust against various network failures, while provides reasonable models with affordable network load.

The key contributions and their effects are:

- The P2PEGASOS algorithm.
- Local voting mechanism for improving the prediction performance.
- The algorithm shows amazing convergence properties even in scenarios with extreme network failures.

8.1.5 Speeding Up the Convergence of P2PEGASOS

In Chapter 7, we continued to investigate the fully distributed setting and proposed a mechanism which increases the convergence speed of the P2PEGASOS with almost an order of magnitude. These algorithms are referred to as P2PEGASOSMU and P2PEGASOSUM. The idea of the proposed algorithms is based on applying an ensemble component by introducing a mechanism which averages the models that "meet" a on certain node. We demonstrated that in the Adaline model our proposal behaves exactly as if an exponentially increasing number of model would be collected and voted through the prediction, but in our case the model requires only a constant size! We pointed out that in the case of the P2PEGASOS algorithm, the exact equality is not true, however, the behavior of the approach is pretty similar. A convergence proof of the P2PEGASOSMU was provided as well.

The chapter began with a motivating section of fully distributed data, then the related work of ensemble learning was briefly discussed. The detailed algorithmic description of the proposed mechanism followed with the discussion of the Adaline model, the Pegasos model and the convergence proof of P2PEGASOSMU algorithm. The empirical evaluation shows that the improved approaches can result in almost an order of magnitude faster convergence speed than the original P2PEGASOS algorithm, while they keep all the advantages of the method. We pointed out that P2PEGASOSMU maintains more independence between the models, hence this is the favorable variant between the two proposals.

The key contributions and their effects are:

- The merging mechanism for the P2PEGASOS algorithm (resulting in the P2PEGASOSMU and P2PEGASOSUM algorithms).
- The convergence proof of P2PEGASOSMU.
- The algorithms show even faster convergence speed than the original P2PEGASOS algorithm while keeping all the advantages of the original one.

8.2 Summary in Hungarian

A tézis fő célkitűzése, hogy a különböző support vektor alapú tanuló módszerek adaptációs lehetőségeit vizsgálja. Ezen belül az adaptációs lehetőségek két eltérő aspektusára fókuszáltunk. Először az úgynevezett algoritmikus adaptáció lehetőségét vizsgáltuk. Azaz arra kerestük a választ, hogy az említett algoritmuscsalád alapötlete, a maximális margó heurisztika, hogyan használható olyan feladatok megoldására, amelyekre az eredeti SVM-formalizmusok nem térnek ki. Másodszor, a tézis későbbi felében az algoritmikus adaptivitásról fokozatosan átirányítottuk a fókuszt az adaptáció rendszermodell aspektusára. Ebben a részben a fő kérdés az, hogy vajon hogyan tudjuk hatékonyan alkalmazni az SVM algoritmusokat egy különleges, az úgynevezett teljesen elosztott rendszermodellben. A rendszermodell alkalmazása számos kihívást foglal magában, mint a hálózati hibák kezelése, a felhasználói kör dinamikus változása, illetve a kiegyensúlyozatlan hálózati terhelés.

Ebben a fejezetben áttekintjük a tézis fő céljait és eredményeit. Ennek során egy rövid összegzést adunk minden fejezetről (3-7. fejezetek). Az összegzés során felsorolás jelleggel kiemeljük az adott fejezet főbb hozzájárulásait és azok hatását.

8.2.1 VMLS-SVM idősorelemzésre

A 3. fejezetben az idősorelemzés problémáját vizsgáltuk, vagyis azt, hogy hogyan tudjuk kiterjeszteni a Least Squares SVM-eket, hogy az idősorok előrejelzésére megfelelőbb algoritmust kapjunk. Az általunk ajánlott algoritmus (VMLS-SVM) egy súlyozott variancia taggal bővíti az LS-SVM eredeti célfüggvényét. A módosítás alapötlete azon az előzetes megfigyelésen alapul, hogy két, azonos predikciós teljesítménnyel rendelkező, idősorelemzésre alkalmas modell közül, a kisebb varianciával illesztő modell nyújt összegzett teljesítményt a túlillesztéstől (overfitting) eltekintve. Az ajánlott módszer az LS-SVM egy kiterjesztésének tekinthető, amely megtartja az eredeti algoritmus összes előnyét, mint pl. a kernel-trükk alkalmazhatósága és lineáris megoldás. Mindamellett egy új hiperparamétert hoz be az algoritmus, ami megnehezíti a módszer finomhangolását.

Főbb hozzájárulások és azok hatása:

- A VMLS-SVM elméleti és algoritmikus levezetése
- Egy, a hiperparaméterek finomhangolására alkalmas módszer
- Az algoritmus szignifikánsan felülmúl számos baseline módszert három elterjedt benchmark adatbázison

8.2.2 Transzformációalapú doménadaptáció

A 4. fejezetben a doménadaptáció problémáját vizsgálta véleménydetekciós feladaton. A fejezetben egy általános keretrendszert ajánlunk (DOMÉN MAPPING LEARNER (DML)) és annak két példányosítását: az első SVM-alapú modelleket használ forrásmodellként, míg a második logisztikus regresszió (LR) alapú tanulókat alkalmaz mint forrásmodell. Az általános módszer alapötlete a következőképpen foglalható össze: a forrás- és céldomének között fennálló relációt egy olyan modelltranszformációs mechanizmussal modellezi, amely egy, a céldomén-ből származó, nagyon kis méretű tanuló adatbázis alapján tanulható.

Főbb hozzájárulások és azok hatása:

- A doménadaptációs feladat transzformációalapú formalizmusa.
- Az általános DML megközelítés.
- Az általános megközelítés két példányosítása (SVM és LR alapú megközelítések)
- Az ajánlott algoritmusok jobb teljesítményt érnek el, mint a direkt megközelítés (baseline) és két state-of-the-art algoritmus (SCL és SCL-MI).

8.2.3 SVM-mel támogatott elosztott ajánlás

Ebben a fejezetben (5. fejezet) az elosztott ajánlás problémájával foglalkoztunk. A fejezet célja kettős. Először bemutatja és motiválja az implicit felhasználói tevékenységekből történő értékelések előrejelzését. Ezután két újsz-

erű heurisztikus eljárást (direkt és időcsúsztatásos módszer) vezet be a probléma megoldására. Ennek során egy érdekes, mindazonáltal rendhagyó alkalmazását mutattuk be az SVM-nek az SMO algoritmus használatával. Arra használjuk az SVM-et, hogy validáljuk a heurisztikáinkat és empirikusan bizonyítsuk azt, hogy az előrejelzett adatbázisnak "érdekes" tulajdonságai vannak, továbbá a szofisztikáltabb heurisztika olyan adatbázist eredményez, amely még inkább érdekes belső struktúrával rendelkezik. Másodsorban újszerű overlay kezelő protokollokat vezettünk be, amelyek a teljesen elosztott collaborative filtering (CF) alapú ajánló algoritmusok teljesen elosztott környezetben történő hatékony megvalósítását teszik lehetővé.

Főbb hozzájárulások és azok hatása:

- Előrejelzési heurisztikák (direkt és időcsúsztatásos módszer).
- Tanulhatóságalapú indirekt validációs technika.
- A FileList.org-ból kinyert ajánló adatbázis [1].
- Overlay kezelő megközelítések: véletlen minta alapú kNN megközelítés, T-MAN alapú variánsok (GLOBAL, VIEW, BEST és PROPORTIONAL) és azok randomizált változatai.
- A GLOBAL peer választáson alapuló technika megfelelő kompromisszumot eredményez a konvergenciasebesség és a hálózati terhelés között.

8.2.4 P2PEGASOS—Egy teljesen elosztott SVM

A 6. fejezetben a teljesen elosztott adatbányászat problémájára fókuszáltunk. A célunk egy általános SVM-alapú algoritmus kidolgozása volt, amely egy teljesen elosztott hálózatban jó minőségű modellek tanulására képes elfogadható kommunikációs komplexitás mellett, mialatt a lehető legkevesebb elvárást támasztjuk a kommunikációs modellel szemben. Az ajánlott algoritmus a P2PEGASOS algoritmus egy koncepcionálisan egyszerű, mégis erőteljes SVM-implementáció. Az alapötlet szerint sok modell végez véletlen sétát a hálózatban, mialatt egy grádiens alapú, online tanuló szabály alkalmazásával javítják magukat.

Főbb hozzájárulások és azok hatása:

- A P2PEGASOS algoritmus.
- Szavazásalapú mechanizmus a predikciós teljesítmény javítására.
- Az algoritmus meglepően jó konvergenciatulajdonságokat mutat még az extrém hibákkal terhelt szimulációkban is.

8.2.5 A P2PEGASOS konvergenciájának további gyorsítása

A 7. fejezetben folytattuk a teljesen elosztott környezetben történő tanulás vizsgálatát, és egy olyan mechanizmust dolgoztunk ki, amely közel egy nagyságrenddel gyorsítja a P2PEGASOS algoritmus konvergenciasebességét. A létrejött algoritmusok a P2PEGASOSMU és a P2PEGASOSUM. Az alapötlet az, hogy vezessünk be egy modellkombinációs eljárást, amely átlagolja a találkozó modelleket. A fejezetben bemutattuk, hogy az ajánlott eljárás az Adaline modell esetében éppen úgy viselkedik, mintha exponenciálisan növekvő számú modellt gyűjtenénk és szavaztatnánk a predikció során, de konstans tárral. Továbbá megmutattuk, hogy habár a P2PEGASOS algoritmus esetén az egzakt ekvivalencia nem valósul meg, a viselkedése nagyon hasonló. A P2PEGASOSMU algoritmus konvergencia bizonyítását is közöljük.

Főbb hozzájárulások és azok hatása:

- Az egyesítő (merging) mechanizmus a P2PEGASOS algoritmushoz, ami a P2PEGASOSMU és P2PEGASOSUM algoritmusokat eredményezi.
- A P2PEGASOSMU algoritmus konvergenciájának bizonyítása.
- Az algoritmusok jelentősen gyorsabb konvergenciasebességet mutatnak, mint az eredeti P2PEGASOS, mialatt megőrzik annak összes előnyös tulajdonságát.

References

[1] The FileList.org-based inferred recommendation dataset. `http://www.inf.u-szeged.hu/rgai/recommendation/`.

[2] FileList. `http://www.filelist.org`, 2005.

[3] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, and J. Reich. Mobiscopes for human spaces. *Pervasive Computing, IEEE*, 6(2):20–29, 2007.

[4] Gediminas Adomavicius and Er Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734–749, 2005.

[5] Hock Ang, Vivekanand Gopalkrishnan, Steven Hoi, and Wee Ng. Cascade RSVM in peer-to-peer networks. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, volume 5211 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2008.

[6] Hock Ang, Vivekanand Gopalkrishnan, Wee Ng, and Steven Hoi. Communication-efficient classification in P2P networks. In *Machine*

Learning and Knowledge Discovery in Databases (ECML PKDD), volume 5781 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2009.

[7] Hock Ang, Vivekanand Gopalkrishnan, Wee Ng, and Steven Hoi. On classifying drifting concepts in P2P networks. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, volume 6321 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2010.

[8] J. Armstrong and F. Collopy. Error measures for generalizing about forecasting methods - empirical comparisons, 1992.

[9] Xiao Bai, Marin Bertier, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. Gossiping personalized queries. In *Proc. 13th Intl. Conf. on Extending Database Technology (EBDT'10)*, 2010.

[10] Arno Bakker, Elth Ogston, and Maarten van Steen. Collaborative filtering using random neighbours in peer-to-peer networks. In *Proceeding of the 1st ACM international workshop on Complex networks meet information and knowledge management (CNIKM '09)*, pages 67–75, New York, NY, USA, 2009. ACM.

[11] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1):105–139, 1999.

[12] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2007. MIT Press.

[13] Danny Bickson, Dahlia Malkhi, and Lidong Zhou. Peer-to-Peer rating. In *Proceedings of the 7th IEEE International Conference on Peer-to-Peer Computing, 2007. (P2P '07)*, pages 211–218, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[14] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine*

Learning (ICML '98), pages 46–54. Morgan Kaufmann, San Francisco, CA, 1998.

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

[16] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 440–447, Prague, Czech Republic, 2007. Association for Computational Linguistics.

[17] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *EMNLP '06: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[18] Léon Bottou and Chih-Jen Lin. Support vector machine solvers. *Large scale kernel machines*, pages 301–320, 2007.

[19] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[20] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, 2006.

[21] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[22] Leo Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1-2):85–103, 1999.

[23] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. PeerSoN: P2P social networking: early experiences and insights. In *Proc. Second ACM EuroSys Workshop on Social Network Systems (SNS'09)*, pages 46–52. ACM, 2009.

[24] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[25] Sylvain Castagnos and Anne Boyer. Modeling preferences in a distributed recommender system. In *Proceedings of the 11th international conference on User Modeling (UM '07)*, pages 400–404, Berlin, Heidelberg, 2007. Springer-Verlag.

[26] Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19:1155–1178, 2007.

[27] Senthilkumar G. Cheetancheri, John Mark Agosta, Denver H. Dash, Karl N. Levitt, Jeff Rowe, and Eve M. Schooler. A distributed host-based worm detection system. In *Proc. 2006 SIGCOMM workshop on Large-scale attack defense (LSAD'06)*, pages 107–113. ACM, 2006.

[28] Ciprian Chelba and Alex Acero. Adaptation of maximum entropy capitalizer: Little data can help a lot. *Computer Speech & Language*, 20(4):382–399, 2006.

[29] Gregory C Chow and An-loh Lin. Best linear unbiased interpolation, distribution, and extrapolation of time series by related series. *The review of Economics and Statistics*, pages 372–375, 1971.

[30] Bram Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

[31] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[32] Sven F. Crone and Swantje Pietsch. A naïve support vector regression benchmark for the nn3 forecasting competition. In *IJCNN*, pages 2454–2459, 2007.

[33] Marco Cuturi, Jean-Philippe Vert, Oystein Birkenes, and Tomoko Matsui. A kernel for time series based on global alignments. *CoRR*, abs/cs/0610033, 2006. informal publication.

[34] Souptik Datta, Kanishka Bhaduri, Chris Giannella, Ran Wolff, and Hillol Kargupta. Distributed data mining in peer-to-peer networks. *IEEE Internet Computing*, 10(4):18–26, 2006.

[35] Hal Daumé, III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263. Association for Computational Linguistics, 2007.

[36] Hal Daumé, III and Daniel Marcu. Domain adaptation for statistical classifiers. *J. Artif. Int. Res.*, 26(1):101–126, 2006.

[37] B. J. de Kruif and T.J.A. de Vries. Pruning error minimization in least squares support vector machines. *IEEE Transactions on Neural Networks*, 14(3):696–702, 2003.

[38] Diaspora. https://joindiaspora.com/.

[39] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, second edition, 2000.

[40] Amaury Lendasse Erkki. Time series prediction competition: The cats benchmark, 2004.

[41] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines, 2000.

[42] J. Faraway and C. Chatfield. Time series forecasting with neural networks: A case study, 1995.

[43] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[44] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[45] John De Goes. So what if you have big data? without data-driven processes and products, it's useless, 2012.

[46] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[47] Rahul Gupta and Sunita Sarawagi. Domain adaptation of information extraction models. *SIGMOD Rec.*, 37(4):35–40, 2008.

[48] Isabelle Guyon, Asa Ben Hur, Steve Gunn, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press, 2004.

[49] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[50] Peng Han, Bo Xie, Fan Yang, and Ruimin Shen. A scalable P2P recommender system based on distributed collaborative filtering. *Expert Systems with Applications*, 27(2):203–210, 2004.

[51] Peng Han, Bo Xie, Fan Yang, Jiajun Wang, and Ruimin Shen. A novel distributed collaborative filtering algorithm and its implementation on p2p overlay network. In *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *LNCS*, pages 106–115. Springer, 2004.

[52] Juris Hartmanis. Computational complexity of random access stored program machines. Technical report, Ithaca, NY, USA, 1970.

[53] István Hegedűs, Nyers Lehel, and Ormándi Róbert. Detecting concept drift in fully distributed environments. In *2012 IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics*, SISY'12, pages 183–188. IEEE, 2012.

[54] István Hegedűs, Busa-Fekete Róbert, Ormándi Róbert, Jelasity Márk, and Kégl Balázs. Peer-to-peer multi-class boosting. In *Euro-Par 2012 Parallel Processing*, volume 7484 of *Lecture Notes in Computer Science*, pages 389–400. Springer Berlin / Heidelberg, 2012.

[55] István Hegedűs, Ormándi Róbert, and Jelasity Márk. Gossip-based learning under drifting concepts in fully distributed networks. In *2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*, SASO'12, pages 79–88. IEEE, 2012.

[56] Chase Hensel and Haimonti Dutta. GADGET SVM: a gossip-based sub-gradient svm solver. In *Intl. Conf. on Machine Learning (ICML), Numerical Mathematics in Machine Learning Workshop*, 2009.

[57] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '99)*, pages 230–237, New York, NY, USA, 1999. ACM.

[58] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

[59] David W. Hosmer and Stanley Lemeshow. *Applied logistic regression*. A Wiley-Interscience publication. Wiley, New York, NY, USA, 2. ed. edition, 2000.

[60] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. Monash Econometrics and Business Statistics Working Papers 13/05, Monash University, Department of Econometrics and Business Statistics, 2005.

[61] Márk Jelasity and Ozalp Babaoglu. T-Man: Gossip-based overlay topology management. In *Engineering Self-Organising Systems: Third Intl. Workshop (ESOA 2005), Revised Selected Papers*, volume 3910 of *LNCS*, pages 1–15. Springer, 2006.

[62] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, 2005.

[63] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321–2339, 2009.

[64] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3):8, 2007.

[65] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, 1999.

[66] Kaggle. Facebook ii—mapping the internet, 2012.

[67] Diane Kelly and Jaime Teevan. Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum*, 37(2):18–28, 2003.

[68] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 482–491. IEEE Computer Society, 2003.

[69] Anne-Marie Kermarrec. Challenges in personalizing and decentralizing the web: An overview of GOSSPLE. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2009)*, volume 5873 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2009.

[70] Mehdi Khashei, Mehdi Bijari, and Gholam Ali Raissi Ardali. Improvement of auto-regressive integrated moving average models using fuzzy logic and artificial neural networks (anns). *Neurocomput.*, 72(4-6):956–967, 2009.

[71] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[72] Nozomi Kobayashi, Kentaro Inui, and Yuji Matsumoto. Extracting aspect-evaluation and aspect-of relations in opinion mining. In *Proceed-*

ings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 1065–1074, Prague, Czech Republic, 2007. Association for Computational Linguistics.

[73] Wojtek Kowalczyk and Nikos Vlassis. Newscast EM. In *17th Advances in Neural Information Processing Systems (NIPS)*, pages 713–720, Cambridge, MA, 2005. MIT Press.

[74] A. Kuh and P. De Wilde. Comments on "pruning error minimization in least squares support vector machines". *IEEE Trans Neural Netw*, 18(2):606–9, 2007.

[75] N.D. Lane, E. Miluzzo, Hong Lu, D. Peebles, T. Choudhury, and A.T. Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9):140–150, 2010.

[76] Neil D. Lawrence and Raquel Urtasun. Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*, pages 601–608, New York, NY, USA, 2009. ACM.

[77] Ping Luo, Hui Xiong, Kevin Lü, and Zhongzhi Shi. Distributed classification in peer-to-peer networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'07)*, pages 968–976, New York, NY, USA, 2007. ACM.

[78] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*, pages 681–688, New York, NY, USA, 2009. ACM.

[79] Yishay Mansour. Learning and domain adaptation. In *Discovery Science*, pages 32–34, 2009.

[80] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation with multiple sources. In *NIPS*, pages 1041–1048, 2008.

[81] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation: Learning bounds and algorithms. *CoRR*, abs/0902.3430, 2009.

[82] Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *Proc. 25th annual ACM symposium on Principles of distributed computing (PODC)*, pages 123–132. ACM, 2006.

[83] David A. McGrew and Scott R. Fluhrer. Multiple forgery attacks against message authentication codes. *IACR Cryptology ePrint Archive*, 2005:161, 2005.

[84] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[85] Alberto Montresor and Márk Jelasity. Peersim: A scalable P2P simulator. In *Proc. 9th IEEE Intl. Conf. on Peer-to-Peer Computing (P2P 2009)*, pages 99–100. IEEE, 2009. extended abstract.

[86] Damon Mosk-Aoyama and Devavrat Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.

[87] K.-R. Muller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines, 1997.

[88] M. O'Connor and J. Herlocker. Clustering items for collaborative filtering. *Recommender Systems Workshop at 1999 Conference on Research and Development in Information Retrieval*, 2001.

[89] Róbert Ormándi. Variance minimization least squares support vector machines for time series analysis. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 965–970, Washington, DC, USA, 2008. IEEE Computer Society.

[90] Róbert Ormándi, István Hegedűs, Kornél Csernai, and Márk Jelasity. Towards inferring ratings from user behavior in bittorrent communities. In

Proceedings of the 2010 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE '10, pages 217–222, Washington, DC, USA, 2010. IEEE Computer Society.

[91] Róbert Ormándi, István Hegedűs, and Richárd Farkas. Opinion mining by transformation-based domain adaptation. In *Proceedings of the 13th international conference on Text, speech and dialogue*, TSD'10, pages 157–164, Berlin, Heidelberg, 2010. Springer-Verlag.

[92] Róbert Ormándi, István Hegedűs, and Márk Jelasity. Overlay management for fully distributed user-based collaborative filtering. In *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part I*, EuroPar'10, pages 446–457, Berlin, Heidelberg, 2010. Springer-Verlag.

[93] Róbert Ormándi, István Hegedűs, and Márk Jelasity. Asynchronous peer-to-peer data mining with stochastic gradient descent. In *17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011)*, volume 6852 of *Lecture Notes in Computer Science*, pages 528–540. Springer, 2011.

[94] Róbert Ormándi, István Hegedűs, and Márk Jelasity. Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a, 2012.

[95] Sinno Jialin Pan, James T. Kwok, and Qiang Yang. Transfer learning via dimensionality reduction. In *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, pages 677–682. AAAI Press, 2008.

[96] Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. In *IJCAI*, pages 1187–1192, 2009.

[97] Yoon-Joo Park and Alexander Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems (RecSys '08)*, pages 11–18, New York, NY, USA, 2008. ACM.

[98] Alex (Sandy) Pentland. Society's nervous system: Building effective government, energy, and public health systems. *Computer*, 45(1):31–38, 2012.

[99] Georgios Pitsilis and Lindsay Marshall. A trust-enabled P2P recommender system. In *Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '06)*, pages 59–64, 2006.

[100] John C. Platt. Advances in kernel methods: Fast training of support vector machines using sequential minimal optimization. pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[101] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. TRIBLER: a social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 20(2):127–138, 2008.

[102] J.A. Pouwelse, J. Yang, M. Meulpolder, D.H.J. Epema, and H.J. Sips. Buddycast: an operational peer-to-peer epidemic protocol stack. In *Proceedings of the 14th Annual Conference of the Advanced School for Computing and Imaging*, pages 200–205. ASCI, 2008.

[103] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work (CSCW '94)*, pages 175–186, New York, NY, USA, 1994. ACM.

[104] V. M. Rivas, J. J. Merelo, P. A. Castillo, M. G. Arenas, and J. G. Castellano. Evolving rbf neural networks for time-series forecasting with evrbf. *Inf. Sci. Inf. Comput. Sci.*, 165(3-4):207–220, 2004.

[105] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.

[106] Jelle Roozenburg. Secure decentralized swarm discovery in Tribler. Mas-

ter's thesis, Parallel and Distributed Systems Group, Delft University of Technology, 2006.

[107] G. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proc. 15th International Conf. on Machine Learning*, pages 515–521. Morgan Kaufmann, San Francisco, CA, 1998.

[108] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming B*, 2010.

[109] Stefan Siersdorfer and Sergej Sizov. Automatic document organization in a P2P environment. In *Advances in Information Retrieval*, volume 3936 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2006.

[110] Jan A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms (Applied Optimization)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[111] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proc. 6th ACM Conf. on Internet measurement (IMC'06)*, pages 189–202. ACM, 2006.

[112] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[113] J. A. K. Suykens and J. Vandewalle. Sparse approximation using least squares support vector machines, 2000.

[114] Balázs Szörényi, Róbert Busa-Fekete, István Hegedűs, Ormándi Róbert, Márk Jelasity, and Balázs Kégl. Gossip-based distributed stochastic bandit algorithms. In *Proceedings of The 30th International Conference on Machine Learning (ICML), 3rd Cycle*, volume 28 of *JMLR: Workshop and Conference Proceedings*, pages 19–27. JMLR: W&CP, 2013.

[115] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk.

Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.

[116] Norbert Tölgyesi and Márk Jelasity. Adaptive peer sampling with newscast. In *Euro-Par 2009*, volume 5704 of *LNCS*, pages 523–534. Springer, 2009.

[117] Amund Tveit. Peer-to-peer based recommendations for mobile commerce. In *Proc. 1st Intl. workshop on Mobile commerce (WMC '01)*, pages 26–29. ACM, 2001.

[118] Robbert van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.

[119] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[120] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unified relevance models for rating prediction in collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 26(3):1–42, 2008.

[121] A. S. Weigend and N. A. Gershenfeld. Time series prediction: Forecasting the future and understanding the past. In *Time series prediction: Forecasting the future and understanding the past*, 1994.

[122] Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.

[123] C. Williams. Prediction with gaussian processes: From linear regression to linear prediction and beyond, 1997.

[124] C. Zhang, P. Dhungel, D. Wu, and K. W. Ross. Unraveling the bit-torrent ecosystem. In *Technical Report, Polytechnic Institute of NYU*, 2009.

[125] Chao Zhang, Prithula Dhungel, Di Wu, Zhengye Liu, and Keith W. Ross.

Bittorrent darknets. In *Proceedings of IEEE Conference on Computer Communications (IEEE INFOCOM '10)*, 2010.

[126] X. Zhang and J. Hutchinson. Simple architectures on fast machines: practical issues in nonlinear time series prediction. In *Time Series Prediction: Forecasting the Future and Understanding the Past*. Santa Fe Institute, Addison-Wesley, 1994.

[127] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 22–32, New York, NY, USA, 2005. ACM.