Distributed classification in P2P networks

Ang, Hock Hee

2015

Ang, H. H. (2015). Distributed classification in P2P networks. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/62190>

<https://doi.org/10.32657/10356/62190>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**DISTRIBUTED CLASSIFICATION
IN P2P NETWORKS**

ANG HOCK HEE

SCHOOL OF COMPUTER ENGINEERING

2015

DISTRIBUTED CLASSIFICATION IN P2P NETWORKS

ANG HOCK HEE

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2015

Acknowledgments

I would like to acknowledge and extend my heartfelt gratitude to the following persons who have made the completion of this phase of my graduate study possible:

- Dr. Ng Wee Keong, my supervisor, who has given me guidance, critical advice and much inspiration.
- Dr. Vivekanand Gopalkrishnan, my co-supervisor, who has given me much attention, guidance, encouragement and support.
- Dr. Steven Hoi Chu Hong, our project collaborator, who has provided me with much guidance, insights and research knowledge.
- Dr. Anwitaman Datta, our project collaborator, for the help and inspiration he extended.
- Lab technicians of CAIS, for all the technical help they have provided me.
- Staff of SCE graduate office for all the administrative support they have given me.
- Family members and friends for their understanding and support.

Contents

Acknowledgments	1
List of Figures	5
List of Tables	7
Abstract	8
1 Introduction	9
1.1 Overview	9
1.2 Motivations	10
1.3 Issues & Challenges	12
1.3.1 Classification in a Basic P2P Environment	13
1.3.2 Imbalanced Data Distribution in P2P Environments	16
1.3.3 Asynchronous Concept Drift in P2P Environments	20
1.3.4 Data Privacy in P2P Environments	23
1.4 Research Objectives & Scope	24
1.5 Contributions	24
1.6 Organization	25
2 Background and Related Work	26
2.1 Distributed Classification	27
2.1.1 Single Classifier Systems	27
2.1.2 Multiple Classifier Systems	29
2.2 Limitations of Existing Approaches	33
2.2.1 Anytimeness	34
2.2.2 Asynchronism	36
2.2.3 Decentralization	37
2.2.4 Tolerance to Peer Dynamism	39
2.2.5 Scalability	40
2.2.6 Invariance to Data Distribution	41
2.2.7 Distributed Concept Drift	42
2.2.8 Data Privacy	43
2.3 Summary	44

3	Classification in P2P Networks with Imbalanced Data Distribution	46
3.1	P2P Cascade RSVM (AllCascade)	47
3.1.1	Local Model Construction	47
3.1.2	Model Propagation and Merging	48
3.1.3	Prediction	51
3.1.4	Complexity Analysis	51
3.2	P2P Bagging Cascade RSVM (RandBag)	52
3.2.1	Local Model Construction	52
3.2.2	Model Collection	53
3.2.3	Prediction	53
3.2.4	Coupon Subset Collection	55
3.2.5	Complexity Analysis	57
3.3	Communication-efficient Multiple Parameter Robust (CEMPaR) Framework	58
3.3.1	Structured P2P Networks	58
3.3.2	Communication Structure Overlay	59
3.3.3	Learning Modules	61
3.3.4	Maintenance Modules	63
3.3.5	Complexity Analysis	66
3.4	P2P Adaptive Classification Ensemble (PACE) Framework	67
3.4.1	Training and Clustering Phase	68
3.4.2	Prediction Phase	70
3.4.3	Complexity Analysis	72
3.5	Experiments	74
3.5.1	Experimental Setup	74
3.5.2	Classification Accuracy	75
3.5.3	Computational Cost for Cascading Models	76
3.5.4	Communication Cost	76
3.5.5	Data Size Distribution	77
3.5.6	Data Class Distribution	78
3.6	Summary	80
4	Classification in P2P Networks with Asynchronous Concept Drifts	81
4.1	Reactive and Proactive Concept Drift detection Ensemble (RePCoDE) Framework	81
4.1.1	Overview	81
4.1.2	Training Phase	82
4.1.3	Prediction Phase	85
4.1.4	Complexity Analysis	86
4.1.5	Communication Cost	87

4.2	Predictive and Parameter Insensitive Ensemble (PINE) Framework . . .	87
4.2.1	Overview	87
4.2.2	(Re-)Training Local Classifiers	90
4.2.3	Mining Temporal Associations	92
4.2.4	The Ensemble of Proactive Models (PEM)	95
4.2.5	The Ensemble of Reactive Models (REM)	95
4.2.6	Balancing Reactive and Proactive Ensembles	96
4.2.7	Prediction by Reactive and Proactive Ensemble Models	97
4.3	Experiments	98
4.3.1	Datasets	98
4.3.2	Experimental setup	99
4.3.3	Performance of the compared approaches	99
4.4	Summary	102
5	Classification in P2P Networks with Privacy Preservation	103
5.1	P2P Privacy-Preserving SVM Framework (PRICADE)	103
5.1.1	P2P secure SVM construction	104
5.1.2	P2P secure prediction	105
5.1.3	Secure multiparty SMO	105
5.1.4	Analysis	113
5.1.5	Security Analysis	114
5.2	Experiments	116
5.2.1	Experimental Setup	116
5.2.2	Classification Accuracy	117
5.2.3	Complexity	117
5.2.4	Number of Dimensions	118
5.3	Summary	119
6	Conclusions and Future Work	120
6.1	Future Work	121
6.1.1	Multisource Learning and Data Fusion	121
6.1.2	Fairness and Quality of Service (QoS)	122
6.1.3	Multi-Label Learning	122
6.1.4	Multi-Instance Learning	123
	References	124

List of Figures

1.1	Feature space of a 2-class problem.	17
1.2	The global distribution of data and local distributions at peers. The horizontal axis represents time, the vertical axis represents data mean. A, B, C, D, E and F represent different concepts (environments).	21
2.1	Illustration of the framework of a generic single classifier system.	27
2.2	Illustration of the framework of a generic ensemble approach. The dotted arrows represent possible communication between peers during model construction (e.g., distributed boosting).	30
2.3	Illustration of the framework of a generic meta-learning approach. The dotted arrows present the different possible data sources for the creation of the meta-data.	32
3.1	Illustration of merging support vectors.	49
3.2	Example of training phase for the two P2P cascade learning approaches: total number of peers (N) is 5, models to be cascaded (k) for RandBag is 3 and number of peers to vote (v) is 3	54
3.3	Example of prediction phase for the two P2P cascade learning approaches: total number of peers (N) is 5, models to be cascaded (k) for RandBag is 3 and number of peers to vote (v) is 3	55
3.4	Identifier ring with 10 peers in 4 groups. Round and square nodes represent peers and super-peers respectively.	60
3.5	Flowchart of CEMPaR processes. Dotted arrow indicates a network communication.	61
3.6	Processes in PACE. Dotted arrows represent network communications.	68
3.7	Effects of number of models cascaded on average training time.	77
3.8	Effects of number of peers on communication cost.	78
3.9	Effects of imbalance class distribution on accuracy	79
4.1	Flowchart of the RePCoDE Framework.	82
4.2	The proposed framework as a sequence diagram.	88

4.3	An illustration of optimal <i>proactive</i> concept drifts for two peers. The shaded areas indicate non detected concept drift periods and the labels m_i indicate the optimal proactive models, where i is the peer number from Figure 1.2.	89
4.4	An illustration of optimal <i>reactive</i> concept drifts for two peers. The shaded areas and their labels indicate optimal reactive models.	89
4.5	An example lag in drift detection. $CDD(B)$ represents the detection of concept drift of concept B . Shaded area indicates the transition period where the concept drift has changed but has not yet been detected. . . .	92
4.6	Moving average of prequential error rate over 100 data points.	100
5.1	Privacy-preserving P2P SVM construction flowchart.	104
5.2	Dependency diagram of secure multiparty SMO.	108
5.3	Effects of the number of peers on accuracy.	118

List of Tables

1.1	Non i.i.d. class distributions.	19
2.1	Comparison of the existing work given the P2P settings.	45
3.1	Description of symbols	73
3.2	Cost comparison.	73
3.3	Summary of the datasets.	74
3.4	Classification accuracy, with equally partitioned data and random class distribution.	76
3.5	Effects of different local training data size $ D_p $ distribution on accuracy (Multiclass Covertypes).	78
3.6	Effects of independent class distribution on accuracy (Multiclass datasets).	79
4.1	Peer p_1 's segmented sequences of the global sequence of concept drift occurrences.	94
4.2	Frequent episodes leading to concept drifts of peer p_1 . For simplicity, the consequent p_1 is removed.	95
4.3	Average communication cost per peer in terms of kilobyte (models propagated).	101
4.4	Average time taken (in msecs).	101
5.1	Commonly used notations.	109
5.2	Total time taken (computation and communication) for varying number of peers.	118
5.3	Total time taken (computation and communication) for varying number of dimensions.	119

Abstract

In recent years, the popularity of Peer-to-Peer (P2P) networks has increased exponentially, owing to the huge amount of usable resources pooled together by the connected peers. Consequently, due to the huge amount of data and computing resources available, learning from the P2P networks in a distributed manner can drastically improve the performance of classification tasks. However, learning in P2P networks is faced with many challenging issues due to the scale and dynamic settings of the P2P networks such as scalability, peer dynamism, asynchronism and fault-tolerance. In addition, existing distributed classification solutions are unsuitable for learning in P2P networks. Therefore, the objective of this thesis is to address the challenges of learning in the P2P networks to allow anyone to construct an accurate and efficient classification model under any type of P2P environment for a diverse domain of applications. This thesis took a systematic approach to (i) analyse the various types of P2P environments and highlight issues that are unique to these environments, (ii) study the existing distributed classification approaches and identify their limitations which make them unsuitable for the P2P environments, and (iii) propose several P2P classification solutions to address the identified challenges of learning in the P2P environments and the limitations of existing approaches. The challenges and limitations have been addressed using the multiple classifier system (cascade SVM and ensemble of classifiers) which has been proven with theoretical studies and experiments on real-life and synthetic datasets to be very effective for the P2P environment. In summary, this thesis has achieved its objective to provide an encyclopedic guide and solutions to learning in the P2P environments, allowing anyone to construct an accurate and efficient classification model under any type of P2P environment for a diverse domain of applications.

Chapter 1

Introduction

1.1 Overview

Classification, also known as supervised learning, is a fundamental research problem in data mining and machine learning. It refers to the task of constructing a model that learns hidden patterns / knowledge from a set of labelled training data to accurately and efficiently predict the labels of unseen data. Formally, the task can be defined as producing a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ from an input vector $\mathbf{x} \in \mathcal{X}$ in d -dimensional space to its class label $y \in \mathcal{Y}$, where $\mathcal{Y}$ denotes the class label space; e.g., $\mathcal{Y} = \{+1, -1\}$ for binary classification. The mapping is then applied for classifying unseen data that is assumed to be drawn from the same underlying distribution. A pair of an input vector and its corresponding class label make an instance $(\mathbf{x}, y)$. A set of instances make a dataset. Let $\mathbf{D} = [\mathbf{x}_1, \dots, \mathbf{x}_\ell]^T = [(x_{i,1}, \dots, x_{i,d})_{i=1}^\ell]^T$ be a set of input data, and let $\mathbf{y} = [y_1, \dots, y_\ell]^T$ be a set of corresponding class labels, where ℓ denotes the total number of training data instances, d denotes the dimensionality of the input data points.

Classification has many practical applications in many domains such as weather prediction, disease diagnosis, stock market prediction, face recognition and network intrusion. In recent years, the advent of distributed systems has revolutionized the way information are generated and collected. Unlike traditional information systems, present day information system spread across various geographical locations, generating varying types and amount of data, depending on the locality and size of business operations. As a result, multiple distributed data sources containing large amount of data are created. With the abundance of knowledge hidden away at each of these data sources, it would be waste not to learn from all these data sources. In addition, learning from all these distributed data sources will definitely generate a better classification model than learning from only one single location. For example, hospitals sharing their medical knowledge to aid in more accurate disease diagnosis, and banks sharing withdrawal information to improve fraud detection. However, traditional classification algorithms that learn from a centralized data source is no longer applicable for these systems. Not only is it not possible to centralize the data due to reason such as communication cost and data privacy, traditional classification algorithms are unable to scale well to these large

datasets. Consequently, large amount of efforts are placed into studying the problem of distributed classification.

Distributed classification refers to learning from the data in a distributed manner without centralizing the data to a single location. The classification problem extends to a distributed setting as follows. Suppose there are N peers in the distributed network, each peer p is a partition $\mathbf{D}_p$ of $\mathbf{D}$ where $\ell = \sum_p \ell_p$. The goal of distributed classification is to collaboratively learn a global prediction function $f : \mathcal{X} \rightarrow \mathcal{Y}$ from the training data of all peers, which maps a d -dimensional vector $\mathbf{x} = (x_1, \dots, x_d)^T \in \mathcal{X}$ to a corresponding class label $y \in \mathcal{Y}$.

Distributed classification not only has to deal with the problem of classification but also the networking issues. There are numerous types of distributed systems but among all, the most notable and well received are the peer to peer (P2P) networks. A P2P network consists numerous peers that are interconnected forming a large distributed network such as the Internet where they share resources such as disk space, bandwidth and CPU cycles. A P2P network is characterized by several properties such as large scale network and dynamic peers with dynamic data. Due to these distinct characteristics which distinguish P2P systems from typical distributed systems, classification in the P2P network is significantly more challenging but with abundance of benefits. *This research investigates the problem of distributed classification in P2P networks.*

1.2 Motivations

In recent years, users of peer to peer (P2P) networks are exponentially increasing as more have come to learn about the advantages of this powerful networking paradigm. One of the main reasons behind the popularity of P2P networks is the sharing of resources among all the connected peers. In a P2P network, heterogeneous peers are arbitrarily interconnected with each other, unlike traditional networks where clients are only connected to the server. These interconnected peers collaboratively come together to share their individual resources such as bandwidth, CPU cycles and disk space to benefit all users in the P2P network. These huge amounts of available pooled resources have encouraged more applications, especially in the area of distributed data mining.

P2P classification is indeed more challenging than learning in a typical distributed system due to the added constraints in the networking component affecting the entire distributed learning algorithm. However, the benefits of learning from these large amounts of data and availability of the vast amount of pooled computing resources far outweigh the efforts needed to overcome the challenges.

For instance, consider the problem of mining rare classes such as network intrusion. Given a peer receives a network intrusion packet out of 10000 network traffic, attempting to learn from such raw data will be difficult as the intrusion traffic is sparse to produce any pattern. However, if a P2P network with 1000 peers collaboratively come together

to learn an intrusion detection model, the combined intrusion traffic will be many times more substantial allowing the learning to be done in a more efficient manner (better use of imbalanced learning techniques such as over sampling).

The availability of the large pool of computing resources is another attractive quality of the P2P system where distributed computing techniques that can make full use of the available resources will definitely be more efficient than centralized computing. By incorporating distributed computing paradigms into the learning algorithms, scalability issues of the traditional learning algorithms will be solved easily. These will allow more knowledge to be mined from larger amount of raw data.

Moreover, P2P classification is important and useful to a broad range of real world applications. For example, in a P2P content sharing system, user preferences such as types of files shared can be mined to optimize delivery, and also to provide targeted advertising. In media annotation tasks [VSN05], users typically only produce tag information for their own repositories. However, by employing P2P classification, peers are able to collaboratively auto-annotate their repositories (at least partially) by learning from the annotations of other peers.

In typical recommendation systems, recommendations are made by users who interact with the system server. In these centralized systems, the server has to store mass amount of information and these servers become the bottleneck of the entire system. In addition, these recommendation information are only available at the particular server (monopoly and single point of failure) and privacy of the users are also at the mercy of the servers. However, considering that the learning of the recommendation is done at each individual user, who then collaboratively come together to produce a distributed model that all peers have access to, this would relieve the stress of the servers and there will be no single bottleneck in the system. Moreover, users are able to control the release of information without comprising their privacy and security.

There are many more such examples (system failure diagnosis, web usage mining) where knowledge can be processed at the individual peer's level and then combined to produce a pool of knowledge. Such systems will give the peers more control and better flexibility which will provide much more benefits to the users as they themselves are the providers of the knowledge.

However, simply solving the problem of P2P classification under certain specific conditions is not enough. As the dynamics of P2P network are very large, unless there is a perfect classification algorithm that can always obtain the maximum classification algorithm with the minimum cost regardless of the environment, one would have to choose an appropriate classification algorithm for deployment under different settings and constraints. Alas the choice of an appropriate classification algorithm is not so straight forward as it depends on a lot of factors. Typically, when a user wants to perform classification in a centralized environment, the user would have no idea what

classifier to use or what are its parameters. Hence, the natural step would be to test several different classifiers with varying parameters in a trial and error fashion, hoping that the most accurate classifier and its best parameter can be found. With the large number of studies on classification, if one is well experienced or knowledgeable enough, one can make use of these information to reduce the choices available and speed up the search for an optimal solution. However, in a distributed settings, the dimensions of the problems are much larger than that of centralized classification, more so for the P2P networks. Hence, it will be very difficult and costly to perform an exhaustive search. Moreover, current P2P classification approaches are very limited which may not cater for all conditions. In such situations, it will be great if there are guidelines that can provide information on which classification algorithms will be applicable for what environments and settings and better yet suggests ways to improve the accuracy or efficiency. This would definitely allow the users to reduce the cost of deployment and still achieve desirable results.

1.3 Issues & Challenges

The scale and dynamic settings of a P2P network make it difficult for a distributed classification algorithm to efficiently create an accurate model that is representative of the global data. Distributed classification in a P2P network has several problems attributed mainly to the following factors of the data, networks and peers: size and locality of network, connectivity, heterogeneity and dynamism of peers, distribution and dynamism of data. At this point, the following research questions have been posed for consideration. How do the dynamics of the P2P networks affect the classification accuracy and efficiency trade-off? How does the different attributes of data, network and peers affect each other? Is there any correlation between the attributes? How does each of these different characteristics of a P2P network affect classification in terms of accuracy and efficiency? Given specific settings of a P2P environment, what is the maximum achievable classification accuracy and what is the minimum cost? Which attribute(s) should one focus on to obtain a specific accuracy and/or efficiency?

This thesis studies the problem of learning in P2P environments from various aspects and identified several issues base on the above research questions. The following subsections examine the above research questions and discuss how the attributes of the P2P networks affect classification algorithms. In addition, the desired properties of classification algorithms under varying P2P environments has been established — learning in a basic P2P environment, learning in P2P networks with imbalanced data distribution, learning in P2P networks with asynchronous concept drift and learning in P2P networks with privacy preservation.

1.3.1 Classification in a Basic P2P Environment

Distributed classification in a generic P2P network is a non trivial task that comprises many research problems. Solving all these problems simultaneously would require a significant amount of time and effort. However, it was noted that there exists a set of fundamental issues [GKSS06, LXLS07, SS06] that relates to any type of P2P classification problem. This means that all P2P classification approaches have to at least address this minimum set of research problems and such P2P environment constrained by this minimum set of research problems only is denoted as a basic P2P network in this work. Next, the characteristics of the data and P2P network which this set of fundamental issues occurs from and the desired properties of the P2P classification approach for solving these fundamental issues are presented.

Characteristics and Issues of a Basic P2P Environment

Network Characteristics A P2P network consists of a set of arbitrary interconnected heterogeneous and dynamic peers. Unlike a typical network which consists of tens of peers, a P2P network can consist of up to hundreds, thousands or even millions of peers, characterizing it as a *massively distributed system*. Peers in a P2P network are *arbitrarily connected*, meaning that each peer is connected to a very small subset of other peers (as compared to the total number of peers in the P2P network). Hence, in order for a message to reach some designated peers, routing of the message may be required which increases the latency of the message transfer. In addition, the message routing are additional communication cost which can take up the bandwidth of the intermediate routing peers. Moreover, a P2P network consists of *heterogeneous peers*, whose computation and communication resources can vary widely. Connections of peers may range from 56k modems to ISDN Digital Subscriber Line to fiber-optics. Some peers may be using an old Intel Pentium Pro processors while others may be using the latest Intel Multi-Core Xeon processors, etc. In addition, a P2P network can also be formed by *dynamic peers* who can join or leave (voluntary or involuntary) the network at any time on an ad-hoc basis.

Data Distribution Data in a P2P system are *naturally distributed* among peers (i.e., data already exists on the peers without any (artificial) distribution from a centralized data source). The distribution of the data with respect to size, class and feature space is usually not known a priori. However, in most cases, the size of training data of peers can vary greatly. Note that the size of training data can affect both the computation and communication cost especially when every peer has different computation and communication resources. For instance, a peer with a larger training data and less processing power will take longer to compute the solution compared to a peer with lesser

training data and more processing power. In addition, it was noted most P2P applications have *dynamic data* at both the peer and network level, hence increasing the size of the global training dataset; i.e., peers' local training data change and increase over time (e.g., streaming data) and with the joining of new peers, new data are contributed to the P2P network.

In order to learn from the data of the large number of peers in a P2P network, significant amount of communications would be required at different stages depending on the learning algorithm. With the need to communicate among such a large number of peers, *scalability* of the communication cost of the algorithm becomes a major problem. Two factors affecting the communication cost are the communication frequency and the size of the communication content. In addition, although every peer may hold only a small subset of the training data (a thousand instances for each peer), combination of the entire P2P network data set will result in a very large training dataset (a million peers will result in a global training set of a billion instances). Hence, scalability of the learning algorithm computational requirements is also an important factor. Note that a P2P network can evolve and grow over time, hence it would be ideal if the computation and communication costs are independent of the size of the P2P network.

With the consideration of the *massive network size* and the *heterogeneous* and *dynamic* peers that are *arbitrarily connected*, *coupling* of the learning algorithm becomes an important factor. Assuming there exists a learning algorithm that is iterative and tightly coupled; i.e., it is required to wait for all peers to finish the computation in the current step before it can proceed to the next iteration. Given the size of the network and assuming that all peers have the same network connection, it would still take a very long time before the algorithm can receive the computation results from all peers since many of the messages have to be routed producing very high latency. Moreover, it is most likely that every peer has different bandwidth and computational resources and different sizes of training dataset, which can further increase the latency of the computation and result transfer. If the algorithm has to wait for all peers, then a substantial amount of time will be required thus delaying the output of the solution.

Furthermore, given that *peers* in the network are highly *dynamic*, if some of the peers leave the network before their part of the computation is completed and propagated, then these computations will be incomplete and the learning algorithm may have to wait forever or even restart the entire computation. Hence, it is impossible for an algorithm to perform global synchronization in a P2P network. In addition, the follow up actions in the event that peers suddenly leave the network will have to be taken into consideration.

As noted above, data in the P2P network can be highly dynamic due to reasons such as growing local training data of peers or the additions of new peers in the P2P network. Therefore, the global training data of a P2P systems can continue to grow over time. This implies that new data will arrive over time and if the algorithm is unable to

learn from these new data, knowledge will be lost and the classification model will be outdated and inaccurate. In addition, prediction task can be requested by any peer at any time, for instance when the algorithm is still learning from the new data (assuming that model computation is not immediate). Hence, the purpose of the algorithm to predict labels of new unlabelled data will be defeated if it is unable to produce a partial result when learning from new data.

Given the size of the network and the dynamism of the peers, an algorithm may not be efficient or function correctly if it requires centralization or the help of a facilitator. This is because centralization of any sort of information is very costly due to the size of the network. In addition, the centralized facilitator could very well be the bottleneck of the algorithm due to the size of the network and amount of data. Moreover, a centralized facilitator is also one of the peers in the P2P network and peers being dynamic may leave or fail at any time, causing breakage of links between the peers and failure of the algorithm.

Desired Properties Summarizing from the above characteristics and issues of a basic P2P network, for any P2P classification approach to be usable, it should possess the following properties (defined as the *basic properties* [DBG⁺06]):

Anytimeness The algorithm should construct the classification model in an incremental manner such that the model can adapt to the new data that arrives. In addition, it must be able to provide a partial solution as and when needed to fulfill its purpose.

Asynchronous The algorithm should not require global synchronization of the network and if possible minimize synchronization between peers such that the computations will not be delay due to reasons such as slower computational power and longer network latency.

Decentralized The algorithm should work in a decentralized manner without causing any bottleneck and failure due to departure of peers.

Tolerance to Peer Dynamism The algorithm must be able carry on working under all situations even if it has lost part of the computational result due to departure of peers.

Scalable The algorithm must be able to scale well regardless of the size of the data and the number of peers. It must be able to complete the model construction and produce a solution in an acceptable amount of time.

1.3.2 Imbalanced Data Distribution in P2P Environments

Distribution of data is an important factor that can affect the classification accuracy for distributed classification [CEH01, CBHK02, WP01]. The importance of this factor is amplified in systems where data are naturally distributed and scale of the distributed system is very large, e.g., P2P systems. This is because, unlike smaller scaled systems, it is not possible to globally manipulate the data as it would require data statistics to be collected from all data sources, which will incur high communication cost and not to mention the cost for data relocation, especially for large scale networks such as P2P networks. Hence, it is important that a distributed algorithm be invariant to the imbalanced data distribution of the training data. Three main factors affecting the performance of distribution classifiers are feature space distribution, size distribution and class distribution. Next, the characteristics and issues of each of these factors will be explained in details.

Characteristics and Issues of Imbalanced Data Distribution

Data Distribution in Feature Space In distributed classification system, distribution/assignment of data is an important component since the classifier learns from each individual distributed subset and different distribution may produce very different classification models, especially true for ensemble solutions.

There are several supporting evidences from previous work that show how different data distribution techniques affect classification performance. For instance, Chawla *et al.* [CEH01] have conducted experiments and compared between random partitioning (each dataset is independent and identically distributed in the feature space, regardless of the class labels), stratified random partitioning (each dataset covers an equal amount of different class data that is independent and identically distributed in the feature space) and intelligent partitioning, which is based on clustering (each dataset covers a different region of the feature space). It is shown that intelligent partitioning generally outperforms random partitioning, although the cost associated with intelligent partitioning may become a concern and outweigh the benefits when the size of the dataset grows. Comparison between random partitioning and stratified random partitioning shows similar results. Therefore, it might not be worth the extra effort to perform data stratification.

However, in cases like learning in P2P systems where data are naturally distributed, for instance web documents, transaction records of various stores, medical records of hospitals, data partitioning and distribution is not required. In such cases, data distributed among different peers/sites could have similar or varying data distribution in the feature space (heterogeneous distribution). Hence, for such situations, the learning task becomes even more challenging since manipulation of the data distribution in

Figure 1.1: Feature space of a 2-class problem.

feature space would incur high communication cost (data statistics collection and data relocation), especially for P2P networks.

Hence, performance of the distributed classification algorithms will then depend on their ability to adapt to the distribution of the data. Although it is noted in [CEH01] that disjoint data distribution in feature space could help to improve the accuracy of an ensemble of classifiers, but in the case of P2P learning, this is not always the case. If there are too many peers with data from a single region of the feature space, then results can be biased and hence affecting the classification accuracy, especially for multiple classifier system. Scenario 1 presents an illustration of how data distribution in feature space can affect classification in P2P networks.

Scenario 1: disjoint data distribution One typical challenge with P2P classification is the issue of *disjoint data distribution and bias* in the ensemble learning. Figure 1.1 depicts a sample 2-D feature space for a two-class classification problem. The data space is represented by two symbols: circles and squares (each representing one class), and the solid line (M0) denotes the optimal decision plane/model. Labels inside the symbols represent the peer/classifier (C1, C2 and C3) that own the data, and their decision planes/models are represented by dotted lines (M1, M2 and M3) respectively. In a regular ensemble learning approach, assuming that peers' training data are independent and identically distributed (i.i.d.), the ensemble solution should be close to M0. However, in this scenario, the ensemble model will be biased towards M1 and M2 causing its accuracy to suffer. Note that even if the data distribution between the decision planes (M1, M2 vs. M3) is equal, the problem will still occur due to the difference in number of votes. However, in reality, it is not possible to adjust the bias by simply using the data density.

Note that the effects of data distribution in feature space mainly affects multiple classifier systems (e.g., ensemble and Meta learning approaches) while its effects on single classifier system (e.g., parallelized and collaborative approaches) are not clear.

However, assuming that the single classifier systems are able to produce exact same results as the corresponding centralized classifiers, then the effects of data distribution will not affect these systems. Hence, if it is known a priori that the feature distribution is highly imbalanced, single classifier systems or P2P classification algorithms that adapt well to imbalanced data distribution should be used. The design of multiple classifier systems for P2P classification should factor in these considerations.

Data Distribution in Class Space The class distribution can be categorized into two types, either balanced or imbalanced. Both of these distributions have very different effect on the training of a classifier. Assuming the cost of misclassification and the level of difficulty for learning the instances are equal for all classes, then balanced class distribution would be the most ideal. However, the natural distribution of a large number of dataset are usually imbalanced, not to mention that the cost of misclassification for each class can defer greatly and level of difficulty for learning the instances are usually not known a-priori.

There have been many researches on how imbalanced class distribution affects classification [WP01, CBHK02]. Weiss and Provost [WP01] have found that for decision tree classifier, imbalanced class distribution affects the classifier in two ways: 1) minority class instances are more probable to wrong classification and 2) minority class rules have higher probabilities of classifying instances wrongly. The experiments also show that although having a balanced class distribution may not be the optimum class distribution for training classifiers, but balanced class distribution often performs better than the imbalanced natural class distribution.

Unlike over-sampling with replacement on minority class data, Chawla *et al.* [CBHK02] presented an oversampling approach, SMOTE, which creates *synthetic* minority class data. SMOTE improves the generalization performance of the trained model by expanding the decision boundary of the minority class to cover more minority class data. SMOTE not only performs better than oversampling, together with undersampling, it has also shown better performance than plain under-sampling.

It should be noted that all the above researches is based on traditional classifiers, and it is not known how imbalanced class distribution (on individual data subset and the global dataset) affects distributed classification. Although, not much is known on how the imbalanced class distribution affects distributed classification, two situations where class distribution can have an adverse effect on the distribution learning (especially on multiple classifier systems) for a P2P network are hypothesized and illustrated.

Scenario 2: skewed class distribution The second scenario is about the issue of *skewed class distribution*. Table 1.1 presents non i.i.d. data and class distributions that may be present in a P2P learning environment. The numbers denote the percentage of the class data that is held by the peers. Note that peers may not hold the same

Table 1.1: Non i.i.d. class distributions.

Peer	skewed				disjointed			
	Class 1	Class 2	Class 3	Class 4	Class 1	Class 2	Class 3	Class 4
P1	70	10	10	10	50	50	0	0
P2	10	70	10	10	50	50	0	0
P3	10	10	70	10	50	50	0	0
P4	10	10	10	70	0	0	50	50

amount of data locally. In an ideal case, peers' data would be i.i.d. and balanced, so the resulting ensemble classifier can perform as well as individual classifiers or better. In reality, however, skewed class distribution is usually unavoidable, and can considerably deteriorate performance of the ensemble classifier. Although techniques such as sampling or cost-sensitive approaches have been proposed to partially alleviate the imbalance issue, no existing technique can completely resolve this challenge, especially in the challenging P2P learning environment. Therefore, if the training data is skewed in an arbitrary manner, the regular ensemble approach that combines all classifiers in a (weighted) majority voting scheme could fail to improve the classification performance if the majority of the classifiers makes wrong predictions. For example, according to a recent empirical study [AGHN13], the accuracy of an SVM ensemble classifier trained from an imbalanced two-class data set decreases when the skew between classes increases.

Scenario 3: disjoint class distribution In extreme cases of *skewed class distribution*, some peers may have no data from certain classes. This special scenario is referred to as *disjoint class distribution* (Table 1.1). The regular ensemble approach that simply combines outputs from all classifiers could perform very poorly in this scenario. For example, since peers P1, P2, and P3 contain no training data from class 3, they will simply make a wrong prediction for a test sample from class 3. As a result, no matter how peer P4 performs, the ensemble classifier that combines all the four peers in a majority voting approach will always make a wrong prediction for any example from class 3.

Note here that for naturally distributed data, data distribution in feature and class space are sometimes related, especially when systems are spread across vast geographical regions. Another observation is that similar to the feature space distribution, the above scenarios may not be applicable to single classifier systems (e.g., parallelized SVM and distributed induction of decision trees). Hence, the design of ensemble based approaches for learning in P2P should emphasize more on these considerations.

Size Distribution The size of the data subset on each peer is an important factor that can affect the overall classification performance. By having more data, the model constructed by the classifier should be able to better represent the knowledge of the data. Likewise, less data would mean a less accurate representative model. Data in

the P2P network are naturally distributed and it is not known a priori how the data are distributed or the size of the training data of the individual peers. Hence, it is an important task to find out if the global accuracy will be affected by the size of the training data of the individual peers. If it affects, then the next task would be to try to minimize this effect.

Desired Properties Summarizing from the above characteristics and issues of a P2P network with imbalanced data distribution, a P2P classification approach that targets on such system should possess the following properties, in addition to the *basic properties*:

Invariant to imbalanced feature space distribution The algorithm must be able to adapt to all kinds of feature space distribution and produce high classification accuracy even in the presence of imbalanced feature space distribution.

Invariant to imbalanced class distribution The algorithm must be able to adapt to all kinds of class distribution regardless if it is a skew in the class distribution or disjoint class distribution among the peers training data and produce high classification accuracy as though the classifier is trained on a centralized data.

Invariant to imbalanced size distribution The algorithm must be able to ensure good global classification accuracy regardless of the local training dataset set size of the peers.

1.3.3 Asynchronous Concept Drift in P2P Environments

Concept drift is a common challenge for many real-world data mining and knowledge discovery applications. It refers to the learning problem where the target concept to be predicted, changes over time in some unforeseen behaviours. Concepts drift in a number of dynamic environments, such as data streams and distributed systems, affect many applications, such as network intrusion detection, spam categorization, fraud detection, epidemiological, climate or demographic data, marketing and web analytics, financial analysis and many more.

Although concept drift has been actively studied in typical centralized settings, the phenomenon exhibits fundamental differences in distributed environments and the solutions are insufficient for the distributed learning environment and are unable to adapt swiftly to the drifted concept, resulting in significant loss of classification accuracy. Hence, it is crucial to address the concept drift problem in distributed classification. Next, the characteristics and issues of distributed concept drift shall be described.

Figure 1.2: The global distribution of data and local distributions at peers. The horizontal axis represents time, the vertical axis represents data mean. A, B, C, D, E and F represent different concepts (environments).

Characteristics and Issues of Asynchronous Concept Drift Typical scenarios only model concept drift from a single source of data. However, in P2P networks, each peer can be viewed as an independent data source. Hence, concept drift in a P2P network affects different peers in diverse ways, such as varying degree or varying time occurrence.

In order for peers to benefit from knowledge sharing, it is assumed that data streams of a subset of peers follow the same unknown probability distributions where the concept may change from time to time. However, it cannot be assumed that the concept always changes instantaneously for the entire set of peers. Data is not i.i.d. across the peers at a given point in time. However, data across a subset of peers will be identically distributed if observed over longer period of time.

Examples of such distributed environments include disease diagnosis during epidemic outbreaks where the epidemic originates from one country and spreads to another with some delay, or weather prediction across different regions where a cyclone brings about weather changes while moving from region to region with a certain speed and direction. In these examples, the concept drifts from peers in some geographical location to other peers in other locations. Although global changes happen and need to be handled unexpectedly, at the local peer level adapting to changes can be made more effective if the first peers to ‘suffer’ can share their knowledge with other peers in a controlled manner.

Fig. 1.2 illustrates the case of *asynchronous concept drift*, where peers encounter the same concepts, but with different delays. It is observed that peer p_2 experiences concept drift identical to the global distribution whereas peer p_1 experiences the same

concept drifts two time steps later. Peer p_3 always experiences concept drifts later than peer p_2 , but faster or slower than peer p_1 and misses some concepts. It is obvious that some (temporal) association exists between peers p_1 and p_2 . The drifts that happen at individual peers will be referred to as *local drifts* and the drift that happens at the system level will be referred to as *global drift*. Hence, mining and exploiting such associations to improve both the detection of and adaptation to the concept drifts is an essential part in handling concept drifts in a distributed environment.

In the typical (stationary) classification scenario each training instance $(\mathbf{x}, y)$ is drawn from some unknown but i.i.d. distribution $P(\mathbf{x}, y)$. In a non-stationary environment the target concept is expected to change over time. *Concept drift* is defined as the change of the underlying unknown probability distribution, i.e., $P_k(\mathbf{x}, y) \neq P_j(\mathbf{x}, y)$, which has occurred from time t_k to t_j ($t_k < t_j$). As a result, the global mapping $f: \mathcal{X} \rightarrow \mathcal{Y}$ that has been learned at time t_k may be no longer accurate at time t_j , where $t_j > t_k$. In a changing environment the training data of a peer p can be represented as $\mathbf{D}_p = \{D_p^1, \dots, D_p^j\}$, where t_j is the current time, and D_p^j is drawn from some unknown probability distribution $P_j(\mathbf{x}, y)$. In this setting given an optimal prediction function f_k^* for $P_k(\mathbf{x}, y)$, it is no longer optimal for $P_j(\mathbf{x}, y)$, i.e. $Err(f_k^*, \mathbf{D}_p^k) < Err(f_k^*, \mathbf{D}_p^j)$, where $Err(f, \mathbf{D}) = (\sum_{\mathbf{x}_i \in \mathbf{D}, y_i \neq f(\mathbf{x}_i)} 1) / |\mathbf{D}|$ is the error rate of f on $\mathbf{D}$.

Hence the ultimate goal of distributed classification in a changing environment is to minimize the error ε for all peers over time

$$\varepsilon = \sum_{p,t} Err(f_p, \mathbf{D}_p^t) \quad (1.1)$$

while satisfying the constraints of the distributed environments, where f_p is the most up-to-date classification model (potentially consisting of a combination of multiple classifiers) of peer p at the time $\mathbf{D}_p^t$ is to be classified. In addition, it should be noted that peers should make use of all sorts of information from other peers to minimize the error ε .

Desired Properties Summarizing from the above characteristics and issues of a P2P network with asynchronous concept drift, a P2P classification approach that targets on such system should possess the following properties, in addition to the basic properties:

Fast concept drifts adaptation The algorithm must be able to quickly adapt to any concept drifts so as to minimize the error.

Accurate concept drifts adaptation The algorithm must be able to accurately adapt to a peer's new concept and should not introduce errors due to inaccurate adaptation.

Maximum information use The algorithm must be able to make use of all available information from all peers to maximize the speed and accuracy to adapt to concept drifts.

1.3.4 Data Privacy in P2P Environments

Data privacy is another area of concern of classification in P2P systems. With the increasing widespread deployment of information systems, an increase in the number of data sources can be observed. Classification, one of the tasks in the data mining is now commonly applied to distributed data sources where several parties such as companies or individuals collaborate to learn the underlying knowledge from these data. The purpose of these collaborations is to allow statistically more significant models to be learned such that they can generalize well on all situations.

However, concerns for the privacy of the data are deterring such collaborations that can provide significantly better classification models. Due to the nature of distributed classification, it is common for these algorithms to exchange data among the different distributed data sources. This leads to the possibility of disclosure of sensitive or confidential data such as names, address, etc. Especially in the environment of P2P where data spans across many sources, it is very likely that these data contain individually identifiable information, and in some cases, is forbidden to be disclosed by the law. Hence, the need for privacy preservation in P2P classification.

Characteristics and Issues of Data Privacy As described above, peers in the P2P networks collaboratively learn a classification model on global data $\mathbf{D} = \bigcup_p \mathbf{D}_p$ where $\mathbf{D}_p$ is the data of a peer p . However, the challenge is the prevention of other peers q from seeing the data $\mathbf{D}_p$ of peer p . In the most protective sense, peers q should not be able to obtain any information on $\mathbf{D}_p$, such as the statistics, distribution, etc. As such, the issue lies in learning from other peers without revealing any information? If learning is possible (demonstrated by many of the privacy preservation works), how can the adherence of the basic properties be ensured and that there is no loss in accuracy without revealing any information as compared to learning with full data disclosure.

Desired Properties Summarizing from the above characteristics and issues of a P2P network with privacy preservation, a P2P classification approach that targets on such system should possess the following properties, in addition to the basic properties:

Minimum information disclosure The algorithm must be able to minimize the disclosure of information to other peers.

Minimum loss in accuracy The algorithm must be able to minimize the loss in accuracy due to the preservation of data privacy.

1.4 Research Objectives & Scope

In this thesis, the objectives are to study the problem of distributed classification in P2P networks, examine how the different characteristics of a P2P network affect classification and propose distributed classification solutions for varying P2P conditions and constraints such that anyone will be able to efficiently learn an accurate classification model that is representative of the global data in any P2P networks.

Hence, the aim is to address the following questions to aid in achieving the objectives. Given some user settings of a P2P system with autonomously learners in a specific domain, what is the possible *accuracy and efficiency tradeoff*? Which classification algorithm will allow the user to accomplish the desired accuracy and efficiency? Given that the user wants to obtain specific higher accuracy or efficiency, what are the attributes or areas that need to be improved and what are the scarifies that need to be made to achieve the goal and how can it be done?

To summarise, the scope of this thesis includes addressing the challenges of learning in (a) P2P environments with imbalanced data distribution, (b) P2P environments with asynchronous concept drift, and (c) P2P environments with privacy preservation.

1.5 Contributions

The contributions of this thesis are as follows:

- This thesis took a systematic approach to flesh out *unique* issues and applicable concerns (from distributed learning) for performing classification in P2P systems. Unlike existing work which only discusses problems of learning in other smaller scale and less dynamic networking environments worsen by the P2P environments, this work includes and highlights issues that are unique to classification in P2P networks. In addition, the desired properties for guiding the design of the P2P classification approaches have been listed.
- A comprehensive survey of the state of the art distributed and P2P classification approaches has been performed. The existing works were critically analysed and their limitations on P2P classification are presented. In addition, possible modifications that may allow the existing distributed learning algorithms to adapt to the P2P systems have been suggested.
- This thesis proposes four novel solutions, viz., AllCascade [AGHN08, AGHN13], RandBag [AGH⁺08, AGHN13], CEMPaR [AGNH09] and PACE [AGHN10] to address the issues of learning in P2P networks with imbalanced data distribution. These solutions provide alternate approaches to address the same set issues with differing cost and benefits. Adding on, this thesis provides guidelines to aid

practitioners in selecting the best solution based on their requirements. Empirical studies have demonstrated that these solutions perform better than existing state-of-the-art solutions.

- This thesis proposes two novel solutions, viz., RePCoDE [AGNH10] and PINE [AGZ⁺13], to address the problem of classification in P2P networks with asynchronous concept drift. These solutions adopt both reactive and proactive approaches to handle asynchronous concept drift and differ by how they detect and handle concept drifts. Empirical studies have demonstrated that these solutions perform better than existing state-of-the-art solutions.
- This thesis proposes a novel solution, PRICADE, to address the problem of classification in P2P networks with privacy preservation. To ensure complete privacy preservation, PRICADE is built using a set of privacy-preserving multiparty SVM protocols, specifically designed for the P2P environment. In addition, PRICADE is based on an ensemble of cascade SVMs framework to provide different levels of privacy-cost trade-off by varying the size of the SVM ensemble and level of secure cascading. Theoretical and empirical studies have demonstrated the effectiveness of PRICADE.

1.6 Organization

This thesis is organized as follows:

- Chapter 1 presents an overview of the problem of distributed classification in the P2P networks and the issues and challenges.
- Chapter 2 provides a background study on the existing distributed classification algorithms and presents their limitations and possible modifications needed to learn in a P2P network.
- Chapter 3 presents four solutions, viz., AllCascade [AGHN08, AGHN13], RandBag [AGH⁺08, AGHN13], CEMPaR [AGNH09] and PACE [AGHN10], that deal with learning in P2P networks with imbalanced data distribution.
- Chapter 4 describes two solutions, viz., RePCoDE [AGNH10] and PINE [AGZ⁺13], which address the problem of classification in P2P networks with asynchronous concept drift.
- Chapter 5 introduces the proposed solution, PRICADE, to address the problem of classification in P2P networks with privacy preservation.
- Chapter 6 concludes this thesis and presents future research directions.

Chapter 2

Background and Related Work

In general, classification refers to a task of learning a prediction model $f : \mathbb{R}^d \rightarrow \mathcal{Y}$ from a collection of training data examples $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$, $\mathbf{x}_i \in \mathbb{R}^d$ denotes a data example and $y_i \in \mathcal{Y}$ denotes its class label where $\mathcal{Y} = \{+1, -1\}$ for binary classification. The goal of a classification task is to efficiently learn an accurate classifier f from the training data D , such that class labels of unseen data examples can be correctly predicted.

Traditionally, learning has only been done on a single machine. Examples of such algorithms include artificial neural networks, Bayes classifier, decision trees and Support Vector Machines (SVM). However, in the present situation, many of these traditional classification algorithms though proven to perform very well on small sized datasets producing very good accuracy with small training time, failed to scale well on large datasets. Exorbitant training time and enormous amount of memory is required to complete the learning process. Hence, to overcome the scalability problem of learning from these large datasets, alternative techniques such as distributed and parallelized learning have been explored.

In addition to the scalability problem, the need to learn from data sources that are naturally distributed among different locations (termed as naturally distributed data), which is becoming more and more typical, is another major motivation for the research in distributed classification. Note that current information systems can span over several different physical locations. In addition, there is a growing need to mine data across multiple systems for improving the quality of the knowledge learned. In most cases, it is impractical to transfer the data to a single location for the purpose of learning, due to various reasons such as cost of data transfer or the consolidated size of the data or privacy issues that prevent disclosure of data to another location.

The rest of this chapter provides an overview of the existing distributed and P2P learning approaches, grouped in a shallow hierarchical manner according to their respective structures. Then their suitability for deployment in the P2P environment based on the desired properties of P2P classification approaches are examined. In addition, some possible modifications required to make the algorithms compatible for classification in P2P environments are discussed.

The presentation of the related work is organized such that readers will be able to easily follow the existing work and obtain a quick understanding without being flooded by the excessive details. Depending on the approach, there could be many options for the different components of the approach resulting in different properties. Hence, categorizing shallowly by the framework of the related work will be able to provide a more satisfactory summary of the related work. In addition, the limitations of the approaches are presented in an issue oriented style allowing readers to easily compare the existing approaches.

2.1 Distributed Classification

Formally, the problem of distributed (and parallelized) classification is defined as follows: Given a set of training data $D = \bigcup_1^N D_j$ spread across N machines, the objective of distributed classification is to learn from these distributed datasets without centralizing the data. Note here that the main difference between algorithms designed for large datasets and naturally distributed data is the need to manually distribute the data. Distributed classification approaches can be broadly categorized into single or multiple classifier systems. The various single classifier systems will be introduced followed by the multiple classifier systems.

2.1.1 Single Classifier Systems

To improve the scalability of the centralized classification algorithms, a natural and intuitive way is to parallelize the internal computational tasks. In addition, many distributed approaches have also been designed in similar fashion where model construction tasks are distributed and collaboratively performed. Examples of such work includes decision trees [BF01, SAM96, ZHA99, GLOK04, JKK98, JA03, JYA05, CSH04, BWGK08] and SVM [CKO⁺06, ZSZ06, ZWB⁺08, CKL⁺07]. The generic framework of a distributed single classifier system is illustrated in Figure 2.1.

Figure 2.1: Illustration of the framework of a generic single classifier system.

In general, distributed induction of decision tree is performed by having each machine iteratively computes the statistics of all the local attributes and then broadcasts these

statistics such that the combined statistics will allow the best splitting attribute to be selected. For instance, Shafer *et al.* [SAM96] presented their approach SPRINT, which performs parallel construction of the attribute list and scanning, then with a coordinator that computes the global statistics to decide the splitting attribute. Joshi *et al.* [JKK98], on the other hand used the parallel hashing paradigm for constructing and searching its statistical data structures to efficiently parallelize the splitting phase. Giannella *et al.* [GLOK04] proposed another approach which builds the decision tree using distributed dot product, with random projection-based dot product estimation to reduce the communication cost. Using agent based data mining system, Caragea *et al.* [CSH04] proposed a distributed decision tree induction approach for learning in heterogeneous and autonomous data sources. Distributed induction of decision tree has also been designed on special machines such as shared memory multiprocessor systems [ZHA99, JYA05] and cache coherent non-uniform memory architecture system [BF01]. An important point to note for these distributed decision tree induction approaches is that for each selection of splitting node, a synchronization step is required for computing the global statistics.

However, Bhaduri *et al.* [BWGK08] proposed a distributed decision tree induction approach that is specifically designed for the P2P networks. It uses distributed majority voting to compute the attributes' statistics. The proposed approach fulfils all the requirements needed for performing classification in the basic P2P environment. In another work, Bar-Or *et al.* [BOKSW05] proposed a hierarchical approach to distributed decision tree induction (DHDT). By propagating the attribute statistics in a semi-centralized hierarchical manner, redundant communication can be reduced during the splitting attribute selection.

With respect to the SVM model construction, bulk of the computational cost originates from the convex problem optimization which remains above $O(n^2)$ [Joa99, Pla98]. Hence, much effort has been made to parallelize the optimization computations such that the computational complexity can be significantly reduced. In general, the optimization computations such as matrix reduction, Interior-Point computations and sequential minimal optimization tasks are distributed among the different machines and results are combined. This process is iterated until the solution converges. Zanni *et al.* [ZSZ06] proposed the parallel gradient projection-based decomposition technique (PGPDT) that performs iterative decomposition that solve the quadratic subproblem in parallel, using gradient projection. Cao *et al.* [CKO⁺06] proposed a parallelized SMO that they implemented with MPI, with experimental results demonstrating that the speedup is almost linear with respect to the number of processors used. More recently, Chang *et al.* [ZWB⁺08] presented their parallel SVM algorithm which loads essential data to various machines and performs row-based approximate matrix factorization in parallel. Chu *et al.* [CKL⁺07] delivered an alternative solution that makes use of the map-reduce

[DG04] paradigm to parallelize summation tasks which can be used on algorithms that reduce to the Statistical Query Model [Kea98]. Among the various data mining algorithms implemented, Chu *et al.* demonstrated how to parallelize SVM by solving the optimization problem in its primal form.

Focusing on the issue of privacy preserving in distributed learning, Agrawal and Srikant [AS00] proposed a data randomization approach which perturbs the individual records and constructs a decision tree by reconstructing the distribution from the perturbed data set. Liu *et al.* [LKR06] on the other hand presented a random projection based multiplicative data perturbation approach to handle data security for computing statistical aggregated matrices, e.g., inner product, correlation coefficient and Euclidean distance, which can be used in many data mining tasks including classification. More recently, Mangasarian *et al.* [MWF08] proposed a privacy preserving SVM that is constructed from random kernel computed using the original data set and randomized matrices.

Other than the randomization approaches, several works [HNWL10, YZ09, YVJ06, YJV06] based on secure multi-party computation (SMC) also addresses the privacy preservation problem. Assuming the presence of a neutral peer, Yu *et al.* [YJV06] proposed a privacy-preserving SVM for horizontally partitioned data (PPSVM). PPSVM first computes the gram matrix by securely computing the scalar product on all peers' data in a round robin manner. The SVM is then constructed on the decrypted gram matrix by the neutral peer. A similar approach is taken for prediction where all peers jointly compute the secure scalar product on their data and the test data, after which the neutral peers compute the prediction.

Han *et al.* [HNWL10] proposed a privacy-preserving gradient descent based on the stochastic approach for vertically partitioned data where all peers are required to jointly compute the secure scalar product [GLLM05] on their private data for updating the function weight vector, until convergence is achieved.

Yi and Zhang [YZ09] proposed a two-party privacy-preserving Naïve Bayes classification approach where the Bayes probabilities are computed securely. They also provide a multiparty extension, where two neutral parties are employed to securely collect shares of the data statistics and then execute the two party protocol.

2.1.2 Multiple Classifier Systems

Multiple classifier systems, unlike single classifier systems, are constructed using multiple base classifiers where one or more base classifiers can be constructed from each individual data subset. Multiple classifier systems work very well in distributed settings since classifiers can be independently trained on the distributed data and combined at a later stage. For most multiple classifier systems, the choice of base classifier can be arbitrary and even a mixture of different types, except for specialized approaches (e.g., cascade

SVM [GCB⁺05, LWW04, TE03, ZLY05]). Multiple classifier systems can be broadly categorized into ensemble (non meta-learning) [CHB⁺02, CMH⁺03, CHBK04, LR05] and meta-learning [CS93a, CS⁺93b, CS97, CS98, PCS00, LYWZ06, GCB⁺05, LWW04, ZLY05] approaches. The ensemble approaches will be presented followed by the meta-learning approaches.

Ensemble Approaches Majority voting is one of the simplest approaches to combine the outputs of multiple classifiers. However, construction of the ensemble of classifiers need not be simple as demonstrated by distributed Ivotes and distributed boosting. The framework of a generic ensemble approach is illustrated in Figure 2.2.

Figure 2.2: Illustration of the framework of a generic ensemble approach. The dotted arrows represent possible communication between peers during model construction (e.g., distributed boosting).

Inspired by the work on pasting of small votes by Breiman [Bre99], Chawla *et al.* [CHB⁺02] extended the work and presented the distributed version of pasting Rvotes, based on random sampling (DRvotes), and Ivotes, based on intelligent sampling (DIvotes). Basically, DRvotes/DIvotes partitions the large dataset into disjoint subsets to be processed on different machines, where the pasting of Rvotes/Ivotes is performed independently on each subset. The final prediction is then made by weighted voting on the outputs of all constructed classifiers. Chawla *et al.* [CHBK04] extended the study of DRvotes/DIvotes to provide in-depth analysis and included neural networks, other than decision tree, as its base classifiers in the experiments.

More recently, Luo *et al.* [LXLS07] presented the P2P Distributed Ivotes (P2P Ivotes) where every peer in the P2P network constructs a set of classifiers on the local training data using Ivotes. Prediction is made by propagating the test instances to other sites and collecting the votes. The main focus of the work is the efficient collection of the prediction votes in the P2P environment using an optimal communication protocol (Distributed Plurality Voting). In another work on P2P classification, Siersdorfer and Sizov [SS06] proposed a framework for classifying web documents by propagating the learned local models to a number of other peers. To reduce the communication cost of model propagation, linear SVM was used as the base classifier which is made up of a single feature vector. However, the approach only collects from a small subset of the

(neighboring) peers and the experiments were conducted with very small number of peers (16 peers) which do not validate the efficacy of approach under P2P settings.

In general, the distributed version of boosting creates a series of classifiers by adaptively re-sampling and combining (ARC) the results of all created classifiers to improve the classification accuracy in a distributed setting. Inspired by centralized boosting, Fan *et al.* [FSZ99] proposed a simplistic approach to perform boosting in a distributed data source iteratively at a single site each time. Lazarevic and Obradovic [LO01] experimented with a more complex approach of distributed boosting, where after each iteration of independent local training, all sites will broadcast their local models to others and collect the learning statistics of all sites for updating the sampling distribution. The rationale of this approach is to allow adaptive re-sampling at a global level. To further extend the distributed boosting, Lazarevic and Obradovic [LO02] showed how distributed boosting can be performed on heterogeneous databases by means of attribute selection through clustering. In another work, Lozano and Rangel [LR05] experimented with various ways to pre-generate all the sampling distributions in order to allow parallel boosting. Lozano and Rangel demonstrated that their proposed approaches, distributions update uncorrelated with margins (DUUM) and parallel boosting via information shifting (PBIS) achieved accuracy comparable to Adaboost while incurring only a fraction of the computation cost.

Another interesting approach was proposed by Skillicorn and McConnell [SM08], which builds classifiers based on the attributes available on the local dataset, for learning on vertically partitioned data in a distributed manner. In [TAV04], the authors presented a complex approach to combine the outputs of the ensemble by first clustering the classifier (based on distances measured by means of classifier validation), and then using majority voting to combine the outputs.

Contrary to majority voting, the following approaches use more complex techniques to combine outputs of multiple classifiers. In [KPHJ99], the authors pioneered the work of representing decision tree in Fourier spectrum, and performed collaborative learning. Kargupta and Park [KP01, KP04] extended the previous work and demonstrated how the ensemble of decision tree in the Fourier representation can be aggregated and re-constructed as a single decision tree. To improve the previous approaches, Kargupta *et al.* [KPD06] proposed an approach to create orthogonal decision trees where all trees are functionally orthogonal to each other. The orthogonality property is a guarantee that the resultant decision tree ensemble is efficient and non-redundancy.

Addressing the issues of mining in real-time scenario, Cho and Wüthrich [CW02] experimented with distributed mining of classification rules. The approach generates a classification rule on each data source and then ranks all the rules based on the performance statistics for selection. Experimental results confirmed huge speedup in rule discovery and efficiency of the training phase. However, the approach is not suitable

for small databases and databases with many classes that can be represented with simple rules.

Basak and Kothari [BK04] proposed a Bayesian based ensemble combination approach for vertically distributed data. This approach allows the use of any classifiers given that the classifier is able to estimate the posterior probability of a data vector.

Meta-learning Approaches Different from ensemble approaches, meta-learning approaches further extends the combination of results by learning from the outputs (meta-data) of these base classifiers to create the meta-learner. The framework of a generic meta-learning approach is illustrated in Figure 2.3.

Figure 2.3: Illustration of the framework of a generic meta-learning approach. The dotted arrows present the different possible data sources for the creation of the meta-data.

Over the years, Chan and Stolfo [CS93a, CS⁺93b, CS97, CS98] have proposed several meta-learning approaches, most notably the arbiter and combiner approaches. The arbiter approach builds the base classifiers using a subset of the whole dataset, and with the outputs of the base classifiers, trains an arbiter by selecting instances that match some selection criteria (a form of adaptive resampling). The prediction is made based on the arbitration rule which decides depending on the prediction of the base classifiers and the arbiter. An arbiter tree approach was also proposed that hierarchically trains a tree of arbiters from the base classifiers (leaf nodes). The combiner approach is a more straight-forward approach that combines the outputs of based classifiers to train the meta-learner (combiner). Several combining strategies have been experimented with, such as using only the prediction of the base classifiers or using the attributes with the prediction of base classifiers to train the meta-learner. In many cases, the arbiter approach has been able to generate higher accuracy than the combiner approach. This is possible due to the more complex meta-learning strategy.

Prodromidis *et al.* [PCS00] surveyed existing meta-learning approaches and the issues related to meta-learning. In addition, they presented a distributed agent-based meta-learning system, JAM [SPT⁺97], and showed how it can be used to perform distributed meta-learning from large distributed system. It is noted that meta-learning approaches

are very dynamic as they can be represented as multi-level tree approaches as meta-learner can be trained on base classifiers or other meta-learners. However, with the addition of new learners, a composite meta-learner can grow to be very inefficient. Hence, related work have be done in [PS98, PSC99] to study the pruning of classifiers in meta-learning to improve efficiency. However, note that meta-learning approaches such as combiner and arbiter are expensive to retrain and require the use of a validation set which is hard to obtain in a P2P environment.

Tsoumakas and Vlahavas [TV02] noted that the meta-learning based on stacking has two scalability issues: 1) training data of the meta-learner grows with the number of distributed databases, which can significantly increase the complexity of training the meta-learner when a large number of distributed databases exists; 2) meta-learning accuracy can be improved when probability distribution of the base classifiers are used which will further increase the dimension of the meta data. To overcome these issues, Tsoumakas and Vlahavas proposed the use of the average of the probability distribution with the class label as the meta data, in addition, create multiple meta-learners from the base classifiers of other sites and using the local data as the validation set (similar to the k fold cross validation concept).

More recently, Lai *et al.* [LYWZ06] employed the meta-learning approach to solve the problem of credit scoring. Due to the lack of data, the approach used bagging to create multiple overlapping subsets, and with neural network as the base classifiers, performed parallelized learning. Similar to the combiner approach, the meta-learner was built using the predictions of the base classifiers.

Specifically for the SVM classifier, cascade learning is another form of meta-learning which has slight similarity with the arbiter approach. It was proposed mainly for the purpose of reducing the computation cost of learning with SVM. Cascade learning is performed in a hierarchical manner, where at the base level, a SVM model is built for each of the training subset. Then, at the next level, the support vectors of some of these models (depending on the hierarchical structure) are merged at the next level and used to construct a new SVM model. These merging and retraining is performed until only a single SVM model is left. A pioneering work by Tveit and Engum [TE03] focused on parallelizing the computation of Proximal SVM with a heap-based tree topology framework. Thereafter, numerous efforts were made to improve the cascade learning. Lu *et al.* [LWW04] compared various ways of cascading SVM while Zhang *et al.* [ZLY05] tried to improve cascade SVM by exploring feedback methods to obtain a global optimal solution. With a similar concept, Graf *et al.* [GCB⁺05] also proposed to cascade SVMs with feedback channels and provide a proof of convergence.

2.2 Limitations of Existing Approaches

This section discusses the limitations and suitability of existing distributed classification approaches for use in the P2P settings. To guide the analysis, the questions raised in

Chapter 1 will be used. To allow easy comparisons between the approaches, the study is organized according to the desired properties of the P2P classification algorithms. This way of structuring naturally answers the questions of as to what properties are unfulfilled or missing from the distributed classification approaches for use in P2P environments.

2.2.1 Anytimeness

Anytimeness refers to the incremental construction of classification models and the ability to report a partial solution anytime. It is an important property as data in the P2P environments is dynamic where new data frequently arrives due to joining of new peers or creation of new data by old peers. Hence, being able to learn incrementally from these new data is important and it is also important to be able to produce a partial result as and when needed. It will not be acceptable to ask the users to wait while the model is being updated and no solution can be provided.

The examination of whether the distributed algorithm fulfils the property of anytime-ness is presented as follows. The focus here is on the incremental building of classification models instead of the ability to produce partial solution. The difference between the different approaches lies in the time taken to produce the answer instead of when it is feasible.

For the distributed decision tree [BF01, SAM96, ZHA99, GLOK04, JKK98, JA03, JYA05, CSH04, CSH04, BOKSW05] (except for P2P decision tree [BWGK08]) and SVM [CKO⁺06, ZSZ06, ZWB⁺08, CKL⁺07] algorithms, the current work does not allow incremental model construction. P2P decision tree [BWGK08] is the only exception in this category that fulfils the anytime-ness property as it is designed for the P2P environment.

All the ensemble approaches [LXLS07, CHB⁺02, CHBK04, TAV04, SM08, SS06, LO01, LO02, LR05, FSZ99, BK04, CW02, KPHJ99, KP01, KP04, KPD06] are in fact able to construct classification models incrementally. On the other hand, only some of the meta-learning approaches such as [GCB⁺05, TE03, ZLY05, LWW04, TV02] are able to construct model incrementally without any retraining while others like [LYWZ06, PCS00, CS93a, CS⁺93b, CS97, CS98, CFPS99, KPHJ99, KP01, KP04, KPD06] require partial retraining in order to achieve incremental learning.

Next, the reasons why the existing distributed classification approaches are unable to construct classification model incrementally is studied. In addition, situations where partial solutions cannot be produced by the distributed classification algorithms was also investigated.

The reason why decision tree induction and distributed SVM approaches are unable to construct model incrementally is because of the nature of the base classifiers employed. Since the distributed decision tree induction and distribution SVM are based on parallelization / distribution of the centralized algorithm, then a simple and intuitive

answer to the incremental model update would be to parallelize / distribute the tasks of an incremental centralized algorithm. To the best of the candidate's knowledge, there is no existing work which discusses how these distributed decision tree induction algorithms can be made incremental but there are studies which examined and proposed incremental versions of the decision tree induction. Hence, an in-depth examination of the existing distributed decision tree induction and incremental centralized decision tree induction algorithms will be required to determine if they can be constructed incrementally under the distributed settings. As for the distributed SVM approaches, a simple solution to address incrementally training is to initialize the old data with the previously obtained coefficients, combined with the new training data with newly initialized coefficients. Although this can be considered as retraining on the entire dataset, the initialization of the old coefficients could help to speed up the convergence of the SVM solution (though the amount of speed up is not known).

For all of the ensemble approaches and many of the meta-learning approaches, they can be built incrementally simply by training another new classification model on the new data. Whereas for the rest of the meta-learning approaches that cannot be built incrementally, the reason is due to the creation and learning of meta-data. This is because some of the approaches use each of the outputs of a base classifier as a feature to the inputs of the meta-learner (e.g., combiner). Therefore, the meta-learner will need to partially rebuild the meta-data, and retrain on the entire modified meta-data. Hence, the key for meta-learning algorithms to be incremental lies in the construction of the meta-data and meta-learner.

Here, the problem of creating a partial solution is discussed. Unless a previous classification model has already existed, there is no way to produce a partial solution until the current training is completed. Moreover, if the most updated result is required, then updating of classification models will have to be done and during the process of updating, classification will have to fall back on an older model until the current update is completed. This applies to the algorithms of all types. However, as multiple classifier systems construct the base classifiers locally, it should be able to generate or update the classification model faster than the distributed single classifier system.

Hence, from the above findings, it can be seen that incremental learning not only depends on the base classifier but also on the algorithm framework. With considerations to the above findings, the following questions of interest are asked. Is it possible to parallelize, distribute updatable base classifiers as the single classifier system solution to allow incremental learning? Are there more events that can hinder the production of a partial solution? How can the time required to generate a partial solution be minimized, especially for single classifier system?

2.2.2 Asynchronism

As noted earlier, due to the size of the P2P network and the communication delays, it is not possible for any approaches to perform global synchronization. For clarification, any approach that does not perform global synchronization is defined as an asynchronous approach and synchronous otherwise.

First, the distributed classification approaches that are not able to fulfil the asynchronous property are listed as follows.

For the single classifier systems (decision tree and SVM) all [ZHA99, BF01, JKK98, CSH04, JA03, JYA05, GLOK04, SAM96, CKO⁺06, ZSZ06, ZWB⁺08, CKL⁺07] except the following decision tree approaches [BOKSW05, BWGK08] are synchronous.

As for the multiple classifier systems, the boosting approach [LO01, LO02, LR05] and meta-learning approaches such as stacking [TV02] and the arbiter and combiner [CS93a, CS⁺93b, CS97, CS98] approaches are synchronous.

Next, the reasons why the distributed classification approaches are synchronous are studied. Due to the need to synchronize all attributes' statistics for the selection of each splitting node, the following distributed decision tree induction approaches [ZHA99, BF01, JKK98, CSH04, JA03, JYA05, GLOK04, SAM96] are synchronous. Similar to distributed decision tree induction, parallel SVM [CKO⁺06, ZSZ06, ZWB⁺08, CKL⁺07] which requires synchronization of the computations for each iteration also does not fulfill the asynchronous requirement.

For distributed boosting approaches, they are generally synchronous as each iteration requires the collection of all peers' models to adaptively resample training data for the next iteration. Although meta-learning approaches are in general asynchronous, some meta-learning requires the meta-data to be generated from the entire network's dataset, which results in a synchronous approach. For instance, the stacking approach proposed by Tsoumakas and Vlahavas [TV02] and the arbiter (tree) approach proposed by Chan and Stolfo [CS93a, CS⁺93b, CS97, CS98] are synchronous whereby they all require the creation of meta data from all training data.

After knowing the components of the algorithms that violate the asynchronous property, the following questions are examined to determine if asynchronous is possible for these algorithms. Here, it is assumed that peers have no prior knowledge on the data characteristics of other peers. Hence, if a peer has to acquire and wait for remote information for computation, is it possible to reduce the number of parties it has to get the information from? Can the same remote computation be handled by multiple peers such that the dependency on any one peer is minimized?

For distributed induction of decision tree, Bhaduri *et al.* [BWGK08] has already demonstrated that it is possible to design asynchronous algorithm. However, the structure of the P2P decision tree induction is vastly different from the other distributed

decision tree induction approaches. Therefore, it is still unclear how the existing synchronous distributed decision tree induction can be made asynchronous. Similarly, although there are existing asynchronous convex problem solvers, they are yet to be used with SVM. Hence more in-depth explorations will be needed.

For distributed boosting, by relaxing the adaptive resampling requirement to allow subset collection of models and without synchronizing the global data distribution, it is possible for distributed boosting to become asynchronous. However, in such cases, the termination conditions will have to be refined and the amount of classification improvement which the arbitrary boosting can bring have to be re-examined.

The stacking approach by Tsoumakas and Vlahavas [TV02] can be made asynchronous by relaxing the meta data creation condition such that only a subset of peers' training data are used and the choice of meta classifier should be one that can perform incremental learning. As for the arbiter (tree) approach proposed by Chan and Stolfo [CS93a, CS⁺93b, CS97, CS98], one can consider using a validation set instead of the entire network's training data to make the algorithm asynchronous. However, this will very likely reduce the classification accuracy of the resultant classification model.

While cascade SVM is mentioned to be asynchronous, if feedback of the algorithm is performed, it will cease to be asynchronous [GCB⁺05]. Therefore, the solution is to either ignore feedback or to perform partial and incremental feedback.

To gain addition insights to the problems related with synchronicity, the following questions based on the findings from the above discussion are asked. Will the asynchronous convex problem solvers be able to help distributed SVM achieve asynchronism and efficiently construct the classification model? How can the adaptive resampling of boosting algorithms be relaxed to achieve asynchronism and how will the classification accuracy be affected? How can a validation dataset be chosen such that it is representative of the entire training data for accurate meta-model creation in the P2P network? How can partial and incremental feedbacks be performed for the cascade SVM?

2.2.3 Decentralization

Decentralization is required for a P2P network due to several reasons such as size of network where the centralized server or coordinator becomes a bottleneck for the P2P algorithm and also because of the peer dynamism where the centralized server or coordinator could leave the network anytime and break the algorithm. Here, a centralized algorithm is defined as one which requires the existence of a centralized coordinator or the need to centralize part of the computation, where failure of the coordinator or centralized resource location will require the algorithm to repeat the computation for all or a major part of the computed results.

To understand how centralization affects current work, algorithms from single classifier systems (decision tree and SVM) [ZHA99, BF01, JKK98, CSH04, JA03, JYA05,

GLOK04, SAM96, CKO⁺06, ZSZ06, ZWB⁺08, CKL⁺07, YVJ06, YJV06] and multiple classifier systems (arbiter, combiner and cascade) [PCS00, CS93a, CS⁺93b, CS97, CS98, CFPS99, GCB⁺05, TE03, ZLY05, LWW04] are first identified.

Next, components of the algorithms that are causing the centralization are determined. For the distributed decision tree induction, the centralized process lies in the choosing of the best splitting node. As for the distributed SVM, it lies in the computation of the statistics (e.g., pivot point for matrix reduction). For the various meta-learning approaches, the centralized process lies in the construction of the meta-learner; meta-model for combiner, arbiter model for arbiter and support vector merging for cascade learning. In general, the centralization serves to reduce computation and communication cost by reducing duplication.

As such, the issues at hand are as follows. Is it possible for the centralized algorithm to be decentralized? How they can be decentralized? What are the additional cost that will be incurred for decentralizing these algorithms?

Firstly, note that all except the distributed hierarchical decision tree induction [BOKSW05] and privacy preserving SVM [YVJ06, YJV06] can be decentralized. These approaches cannot be decentralized due to the structure of the algorithm otherwise the desired properties from these algorithms will be lost (e.g., the distributed secure dot matrix computation from [YVJ06, YJV06] requires the initiator to compute the final secure matrix, where failure of this initiator will cause the secure computation to fail).

In general, converting from centralized to decentralized can be done by propagating the required information to all peers instead of having a centralized server or coordinator. In this way, every peer will act like the coordinator and duplicate all the computations. Hence, the decentralization comes at the cost of increased computation and communication cost. For instance, distributed induction of decision approaches propagate the attribute statistics to the coordinator who will then find the best splitting node. Therefore, a decentralized version will have all peers propagating the attributes' statistics to everyone and then each peer will independently find the best splitting node. Similarly for the cascade SVM learning, after the construction of the base level SVM, each peer can then propagate the support vectors to all other peers who can then independently cascade and merge the SVMs.

Hence, it is observed from the above findings that decentralization may not be possible for some specific algorithmic structures and significant amount of additional cost may be incurred to enforce decentralization. Naturally, for the rest of the research, the following questions are asked to guide this work. How can the computation cost of the duplicated task be reduced due to decentralization? Can the amount of information required to propagate to others for decentralizing be reduced? Also, for approaches that are still centralized, is it possible to reduce the chances of the coordinator being a bottleneck? Can the algorithm be designed to recover from the lost of coordinator if the coordinating peer was to depart from the network (e.g., self electing coordinator)?

2.2.4 Tolerance to Peer Dynamism

Tolerance to peer dynamism is a required property as peers in the P2P network may leave (without notice) the network or fail at anytime in an ad-hoc manner. Hence, the P2P classification algorithms need to be able to recover from the loss of these peers which can result in the loss of data and/or computation results. By being tolerant to peer dynamism, an approach is able to continue computations even when data or partial computation results are lost, without having to restart the *entire* model construction or prediction phase. Or in another word, the degree of an algorithm's tolerance to peer dynamism depends on the amount of recomputations that are required due to the loss of peers. In addition, the restriction that the created model must be correct have to be put in place.

First, it will be indicated if the approaches are tolerant to peer dynamism, and the correctness of their constructed models discussed. In general, distributed induction of decision tree [BF01, SAM96, ZHA99, GLOK04, JKK98, JA03, JYA05, CSH04, CSH04, BOKSW05] are not tolerant to peer dynamism except for the P2P decision tree [BWGK08]. With some additional condition checks and small amount of recomputations, distributed SVMs [CKO⁺06, ZSZ06, ZWB⁺08, CKL⁺07] are in general tolerant to peer dynamism.

For the multiple classifier systems, most if not all of the approaches (ensemble, cascade and stacking based) [LXLS07, CHB⁺02, CHBK04, TAV04, SM08, SS06, LO01, LO02, LR05, FSZ99, BK04, CW02, KPHJ99, KP01, KP04, KPD06, TV02, GCB⁺05, TE03, ZLY05, LWW04] are tolerant to peer dynamism.

The only exception is the arbiter and combiner meta-learning approaches [CS93a, CS⁺93b, CS97, CS98].

Next, the components of the algorithms that caused the unsatisfiability of the tolerance to peer dynamism property are examined. For distributed decision tree induction, although computation of the decision tree can still continue even if some of the peers leave the network, but since the previous nodes were built on old data, they may not be correct and these errors will compromise the accuracy of the decision tree.

For the arbiter and combiner meta-learning approaches, a dependency on the base model and the meta-model (e.g., output of base models is a feature of the meta-data) is present. Due to this dependency, when some of the peers fail or leave the network, and their models become unavailable, then the inputs for the meta-models will not be correct since it will be missing some attributes which will degrade the classification accuracy of the meta-model. This will then require the reconstruction of the entire meta-model.

Hence, to address these issues, the following questions are asked. Can the decision tree be fixed without recomputing the entire tree? Is there a way to retain the knowledge (e.g., models or computation results) of peers such that when they leave the network their knowledge is still *alive* within the network.

For the first question, Bhaduri *et al.* [BWGK08] have demonstrated one such possibility but the structure of the algorithm is significantly different from other distributed decision tree induction algorithms. Hence, it is not trivial to find an answer to this question. For the second question, the simplest way is to propagate the peers' models, which will allow knowledge retention.

However, these answers open up even more questions. How can the knowledge from within the P2P network be efficiently retained and retrieved? How substantial is the loss in accuracy if the model is constructed from partial information? If re-computation cannot be avoided, is there a way to speed up the computation, for instance by reusing previously computed results?

2.2.5 Scalability

Scalability is an important property for P2P classification approaches since the number of peers in the network usually exceeds hundreds, thousands and millions. Having a large number of peers also implies a large dataset size. Notice that distributed classification algorithms have been designed with scalability in mind in its original version where all computation are performed in a distributed manner. However, with consideration to the above desired properties, modification may be made to the algorithms so as to ensure their suitability for P2P classification. Therefore, the following analysis will be based on the proposed modified algorithms.

First some possible unscalable distributed classifiers and the reasons for their poor scalability are examined. For distributed decision tree [BF01, SAM96, ZHA99, GLOK04, JKK98, JA03, JYA05, CSH04, CSH04] and SVM [CKO⁺06, ZSZ06, ZWB⁺08, CKL⁺07], to allow decentralization, the algorithm will have to propagate some information to *all* peers. The propagation of information may cause the algorithms to become unscalable as the number of peers in the P2P network is very large.

As for the multiple classifier system, if the base models are propagated to other peers for the purpose of decentralization and fault tolerance, then these algorithms will become unscalable. Moreover, the processing of all these peers models may or may not scale as the P2P network grows.

Next, the following questions are asked to see if the scalability of the above algorithms can be improved. Is it possible not to propagate to all peers? Is it possible to reduce the size of the information to be propagated? Can the processing requirements on the collected information be reduced?

For the distributed decision trees and SVM, it would be impossible not to propagate to all peers. Since the decision tree induction is a recursive process, if only a subset of peers' statistics are collected, then this set of peers' statistics will have to be collected for rest of the process. This may cause problems such as selective issues and synchronization problems. Since only the peers' statistics are propagated, it is already very small,

hence cannot be further reduced. Reduction of the processing requirement of collected information is also not applicable.

For the multiple classifier systems, it would be possible not to propagate or collect from all peers. However, classification accuracy will be affected as a result. As for the reduction of information to be propagated, with careful selection of the base classifier, it would be possible. For instance, instead of choosing a non-linear SVM model with a large number of support vectors, one could choose a linear SVM with only a single weight vector. On the other hand, if test instance propagation and vote collection approach is employed, efficient communication protocols should be used. As for reduction of collected information, it would be possible if model summarization or selection is performed. By using less models, there should be a reduction in the computational overhead.

In addition to the above questions, the following considerations should also be examined so as to ensure the design of a scalable algorithm. Can the available computational resources in the P2P network be exploited to improve the computational scalability? Can the size of the information to be exchanged be minimized and the frequency of peer communications be reduced? Is there a more efficient approach to data propagation?

2.2.6 Invariance to Data Distribution

As discussed earlier in Chapter 1.3.2, it is shown that distribution of the peers training data can be imbalanced in several ways (feature space, class and size). Hence, it is important for an algorithm to be invariant to the data distribution to ensure good classification accuracy regardless of the peers' training data distribution.

Before deciding the remedial actions to take for distributed classification approaches adversely affected by the imbalanced data distribution, there is a need to know which of the approaches are affected.

Since distributed decision tree induction [BF01, SAM96, ZHA99, GLOK04, JKK98, JA03, JYA05, CSH04, CSH04, BOKSW05, BWGK08] and distributed SVM [CKO⁺06, ZSZ06, ZWB⁺08, CKL⁺07] approaches outputs a single classifier, they are not affected by how data is distributed among the large number of peers in the P2P network.

Unlike ensemble approaches such as [LO01, LO02, LR05, FSZ99] (boosting) which adapt to the *global* data distribution, initial observation of the following voting approaches such as [CHB⁺02, LXLS07, SS06, SM08] highlights that these algorithms could be adversely affected by the imbalanced data distribution. As for the following algorithms [BK04, CW02, TAV04, KPHJ99, KP01, KP04, KPD06], it is unclear how they will be affected by the imbalanced data distribution. Hence, further investigation will be required.

For meta-learning approaches, it is very likely that they are able to handle imbalanced data distribution especially if the validation dataset used for creating the meta-data is representative of the global data. In addition, note that cascade SVM approaches

are invariant to the imbalanced data distribution since the output of these approaches is a single SVM classifier, i.e., the output classifier is considered to be trained on the global data.

A simple approach to invariance of the data distribution is for the peers to propagate their data distribution information. However, it is unclear at the moment how it can be done. Moreover, this will increase the communication cost. Hence the questions to ask are how can the data distribution be accurately measured and the information be efficiently exchanged? How can these data distribution information be effectively made use of to design a distributed classification algorithm that is invariant to imbalanced data distribution among peers or better yet exploit the information to improve the classification accuracy?

2.2.7 Distributed Concept Drift

Concept drift has been well studied in the typical centralized setting but the same is not true for the P2P environment. As explained in Chapter 1.3.3, asynchronous concept drifts in the P2P environments could affect peers in various different ways as compared to the centralized setting. However, existing approaches [BWGK08, LXLS07] simply assume the concept drift scenario of a centralized setting whereby all peers are affected in the same manner at the same time. This assumption could lead to peers adapting to the wrong concept at the wrong time, hence adversely affecting their classification accuracy.

For instance, single model approaches [BWGK08, IGD11] which handle concept drifts by regularly updating the previous model are unable to handle asynchronous concept drifts as all peers adapt to the same concept and converge to the same global model. This will cause peers whose concepts have yet to (or will not) drift to prematurely adapt to the new concept or cause the global model to continuously switch between the old and new concepts.

While ensemble approaches [LXLS07] have been developed for the P2P environments, they do not address the problem of asynchronous concept drift in the P2P environments. Nevertheless, ensemble approaches that build multiple models and adapt to concept drifts via combination rules or model selection could potentially be used to address the problem of asynchronous concept drifts in P2P environments, since different models could be selected and used on different concepts for different peers.

However, note that most existing ensemble approaches are reactive by nature, meaning that the concept drift needs to have occurred for awhile, i.e. drop in accuracy, before the drift can be detected and adapted. As such, proactive approaches [YWZ06] that aim to predict concept drifts before they happen have been proposed. However, as these approaches are based on the centralized settings, the drift prediction are only based on the local historical data without consideration of other peers data or the relationships among the peers.

As such, the following questions lead to addressing the challenges of asynchronous concept drifts in the P2P environments. How can each peer adapt to the concept drifts as soon as possible and yet not affecting other peers? How can peers make use of information from other peers to aid them in the adaptation of concept drifts and to improve accuracy?

2.2.8 Data Privacy

Although data privacy issues have been addressed in typical distributed settings by approaches such as randomization and secure multi-party computation (SMC). They are not sufficient for the P2P environment.

While the randomization approaches [LKR06, MWF08] efficiently preserve the privacy of data by using (random) data perturbation techniques, they incur a significant loss in accuracy due to the addition of noise. With the large number of peers in the P2P environment, the loss in accuracy could be worsen.

On the contrary, SMC approaches achieve high accuracy since their resultant solutions are essentially the same as that of the unsecured versions as SMC approaches use cryptography methods to securely perform computations. However, they are unable to satisfy the P2P requirements due to several reasons. First, existing SMC approaches only build a *single* model involving *all* peers. SMC requires high synchronization between the peers involved and by building a single model, the synchronization among peers is maximized. Moreover, since all peers are necessary for the computations [YJV06] (training and prediction), the single constructed model will definitely be affected by any peer failure, requiring the entire model of all peers to be reconstructed.

Second, as SMC computations incur higher computational cost, existing approaches worsen this problem by performing computations in a sequential manner, involving all peers, such as the gram matrix computations of PPSVM. Thus, reducing the scalability of the approach.

Third, many of these approaches adopt non-participating neutral peers to facilitate the privacy preservation process. Neutral peers do not contribute private data for computations are unavailable in the P2P network since all peers participate with the intention to share knowledge and benefit from the sharing. In addition, when such a peer acts as a central coordinator, it becomes a bottleneck affecting scalability.

Looking at the above issues, it is clear the existing privacy preserving solution does not possess all the desired property of P2P algorithms. Hence, it is clear that the following questions have to be answered to achieve the goal of data privacy for distributed classification in the P2P environment. How to ensure that peers' data are protected while still having the solution possesses the desired properties of P2P algorithms (cf. Chapter 1.3.1)? How to ensure that the accuracy is not affected by the privacy preservation process?

2.3 Summary

This chapter has provided a comprehensive overview of the existing distributed classification approaches, ranging from distributed and parallelized traditional classifiers to multiple classifier systems such as ensemble and meta-learning approaches. Moreover, suitability of the existing distributed classification approaches for use in the P2P environment based on the desired properties described in Chapter 1 have been examined. A summary of the related work with regards to the desired properties is provided in Table 2.1. In addition, possible modifications or extensions to improve the suitability of existing distributed classification algorithms on P2P networks have been suggested.

As discussed, for the problem of learning in a basic P2P networks, multiple classifier systems do seem to be more promising whereby many of the issues with desired properties can be addressed with fewer difficulties compared to the single classifier systems. For instance, asynchronism is a big issue for the single classifier system while for multiple classifier systems, most of the approaches are naturally asynchronous. In addition, it is not so trivial for single classifier systems to achieve anytimeness (incremental and fast and recent partial solution) as compared to multiple classifier systems where there are many possible and simpler approaches to extend and improve the anytimeness. Moreover, most multiple classifier systems are inherently fault tolerant due to the basic architecture of the algorithm unlike single classifier systems where loss of data, due to reasons such as failure of peers, will most likely require a portion of the solution to be recomputed.

However, when learning from imbalanced data distribution, single classifier systems are a better choice as they are intrinsically invariant to the imbalanced data distribution among peers. Fortunately, there are also a number of multiple classifier systems that are invariant to the imbalanced data distribution such as boosting, cascade learning and some of the meta-learning approaches.

All in all, it seems that multiple classifier systems are emerging as the better choice for future explorations due to their highly flexible and extensible characteristics.

Table 2.1: Comparison of the existing work given the P2P settings.

Approaches	Classifier	Anytimeness	Asynchronous	Decentralized	Peer Dynamism Tolerant	Scalable	Data Distribution Invariant	Distributed Concept Drift Tolerant	Data Privacy Preserving
Distributed Decision Tree ([BF01, SAM96, ZHA99, GLOK04, JKK98, JA03, JYA05, CSH04])	Decision tree	No	No	No	No	No	Yes	No	No
P2P Decision Tree ([BWGK08])	Decision tree	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Parallel SVM ([CKO ⁺ 06, ZSZ06, ZWB ⁺ 08, CKL ⁺ 07])	SVM	No	No	No	Yes	No	Yes	No	No
Randomized Privacy Preserving ([AS00, LKR06, MWF08])	Decision tree, SVM, etc	No	No	No	No	No	Yes	No	Yes
Secure Multi-party Computation Privacy Preserving ([HNWL10, YZ09, YVJ06, YJV06])	Stochastic gradient descent, SVM, etc	No	No	No	No	No	Yes	No	Yes
Cascade SVM ([TE03, LWV04, GCB ⁺ 05, ZLY05])	SVM	Yes	No	No	Yes	Yes	Yes	No	No
Ensemble ([CHBK04, SS06, LXL07])	Decision tree, Linear SVM, etc	Yes	Yes	Yes	Yes	Yes	No	No	No
Distributed Boosting ([FSZ99, LO02, LR05])	Neural network, Rule learner, etc	No	No	No	Yes	Yes	Yes	No	No
Meta-learning ([CS93a, CS ⁺ 93b, CS97, CS98, PCS00])	Decision tree, Bayes, etc	No	No	No	No	Yes	Yes	No	No

Chapter 3

Classification in P2P Networks with Imbalanced Data Distribution

This chapter addresses the problem of learning in the P2P networks. As a preliminary study, the problems of learning in a basic P2P environment with the issues of imbalanced data distribution are tackled.

Based on the studies of the existing work on distributed classification approaches, it was found that multiple classifier systems are to be more suited for performing classification in P2P networks. Hence, the proposed solutions are driven in the direction of the multiple classifier systems, where both the ensemble and meta-learning approaches have been explored. The proposed approaches are designed to fulfil the desired properties of learning in basic P2P networks and learning in P2P networks with imbalanced data distribution.

With regards to their architecture, P2P networks generally implement some form of virtual overlay network on top of the physical network topology. There are in general two types of virtual overlay networks: unstructured or structured P2P networks. Unstructured P2P networks does not impose particular structure on the overlay network allowing easy expansion and local optimization of the network. However, due to the lack of structure, it is often difficult to perform search and communication is normally done through flooding. Structured P2P networks on the other hand organize peers into specific topology to enable efficient search and communication but incur additional overheads to maintain the topology. As such, the proposed approaches take into consideration the possible types of P2P network it can be deployed on and how the choice of network affects the performance and cost. Furthermore, to improve tolerance to peer dynamism, existing P2P file-sharing systems are mimicked and the model propagation techniques examined where peers' base classifiers are propagated to other peers. This improves the tolerance to peers' failures by increasing the availability of the models. As such, considerations have been made to reduce the communication cost of model propagation to improve the approaches' scalability. Care has been taken to decentralize the algorithm

and ensure that the approaches are asynchronous and incremental so that a partial solution can be generated as and when needed. Moreover, the proposed approaches are made to be invariant to the data distribution of peers' training data.

In the following sections, four different P2P classification approaches (AllCascade, RandBag, CEMPaR and PACE) are presented. To summarise, AllCascade, RandBag and CEMPaR are based on the cascading SVM paradigm while PACE is based on the multiple classifier system. In addition, note that CEMPaR is specifically designed for structured P2P networks with the aim of drastically reducing the communication cost of learning. Finally, the experimental results to justify the effectiveness of the proposed approaches are presented.

3.1 P2P Cascade RSVM (AllCascade)

Here, the feasibility of building SVM in a P2P network regardless of the P2P network architecture are investigated. SVM is a class of powerful classification and regression algorithms but its usability is affected by its high memory and computational requirements. A proven approach for alleviating such requirements, while not degrading classification performance, is the cascade SVM approach, where the data set is partitioned into smaller chunks and small-scale SVM learning is performed on these chunks. Support vectors obtained from these small-scale SVM models are combined with other chunks to derive better support vectors and improve the final set of support vectors.

This section shows how the cascading of SVM learning can be mapped to a P2P network of data propagation. As network communication is a performance issue, this section illustrates how the sending of support vectors from peer to peer can be improved using the Reduced Support Vector Machine (RSVM) [LM01, LL03] approach. The proposed P2P Cascade RSVM (AllCascade) provides a method for learning classifiers in a P2P network that has classification accuracy comparable to a centralized classifier, yet satisfies the characteristics of P2P computing and has an upper bound on the communication overhead. Without any synchronization requirements, the proposed approach is well suited for all types of P2P networks especially the loosely coupled environments of unstructured and decentralized P2P networks. Experimental results confirm the feasibility and attractiveness of using this approach.

The rest of this section shows how each phase of the Cascade SVM approaches are transformed to suit the P2P environments. The phases are 1) local model construction, 2) model propagation and merging and 3) prediction.

3.1.1 Local Model Construction

Based on the paradigm of Cascade SVM, AllCascade has been proposed as follows. In the initial step of building the local model, with consideration to the weakness of SVM with respect to the size of the resultant support vectors set, the use of RSVM in

place of SVM has been proposed. Although research has shown that RSVM being an approximate solution has lower accuracy than SVM, it is possible to restrict the size of the subset RSVM uses to solve the SVM optimization problem and in turn caps the size of the local model. This property of RSVM is very important as it allows the restriction of the local model size and know a priori the communication cost that will be incurred due to model propagation. In addition, a reduction in the number of support vectors also means a reduction in the computation cost of merging the support vectors.

However, with the reduction of support vectors, the classification accuracy of the proposed approach may also be affected as the SVM hyperplane is constructed directly from the support vectors. Regardless, previous work has shown that the classification accuracy of RSVM is only slightly lower than SVM. Another drawback with the use of RSVM is the inability to guarantee that the solution will converge to the global optimal solution since RSVM is an approximate solution. Graf *et al.* [GCB⁺05] has shown that in order for Cascade SVM to converge to the global optimal solution, a feedback process validating the cascaded model is required. However, in the P2P environment, it is not possible to perform feedback since it requires the synchronization of all peers in the P2P network; hence even with SVM, Cascade SVM in P2P network would not be able to converge to the optimal solution. Hence, it is still beneficial to replace SVM with RSVM in initial building of the local models in spite of the drawbacks of RSVM.

In a P2P environment, one would expect that the local data of peers change frequently. In such situations, given that the new data arrive in batches, each of these new batches of training data will be treated as a new peer's data, going through the same process as the initial local training data, i.e., building of RSVM and model propagation. This allows the new data to be merged with the existing data which can be viewed as a form of incremental learning and hence address the issue of data dynamism. However, in this work, only the issue of incremental learning and not decremental learning which is related to the concept drift issues that will be discussed in the later chapter.

3.1.2 Model Propagation and Merging

Once the local model is generated, it is propagated to other peers for merging. Although model propagation incurs high communication cost (minimized as mentioned above), it allows the proposed approach to reduce the adverse effect of peer dynamism; even if peers have gone offline, as long as they have propagated their models successfully, their models will still be in the P2P network, which allow other peers to learn from the data of offline peers. It is noted that the learning of data of offline peers is also important as P2P networks are highly dynamic and peers may go offline for various reasons. Therefore, in order to ensure high classification accuracy, it would be best to learn from peers' data. Another advantage of model propagation is that it allows the proposed approach to achieve local optima through local feedback, which will be discussed later. In addition,

Figure 3.1: Illustration of merging support vectors.

the proposed approach ensures that every model is only sent to each peer once, to prevent redundancy in communication and the high duplication rate of models (among peers) allows high throughput for the model propagation.

Like the automatic document organization approach, the proposed approach is not coupled with any model propagation method. Therefore, it can be deployed in any type of P2P network, making it very flexible. Moreover, by mapping the problem of model propagation to file propagation in P2P networks, which has been extensively studied, many of the existing solutions can be employed. One such solution could be the UPTReC [WDKS07] algorithm. UPTReC provides a probabilistic guarantee in file consistency and ensures that models can be properly propagated within the P2P network. Wang *et al.* show that UPTReC can reduce up to 70% overhead messages compared with other existing techniques.

Unlike the Cascade SVM, peers do not have control over how, when and the number of models that they will collect at any one time. Hence, the proposed approach is unable to use a structured manner to merge the collected support vectors and has to perform merging in an arbitrary fashion as follows. Given an interval duration d , for all models collected within this interval, it will be merged to the local cascaded model at the end of the interval. With d taking the most extreme values, given $d = 0$, this implies that whenever a model is collected, it is immediately merged to the local cascaded model; given $d =$ the time required to collect models of all uncollected peers in the P2P network, it means that merging will only be performed when all peers' models are collected. For example, given a scenario where peer j joins the P2P network at time t_0 . Then, before the end of the time interval d , peer j receives three new models from other peers. Hence at time $t_1 = t_0 + d$, peer j will merge the three newly received models with his own local model. Again, before the end of another time interval, peer j receives two new models. Then at time $t_2 = t_1 + d$, peer j will merge the previously cascaded model with the two newly received models and this process continues. This example is illustrated in Figure 3.1, and algorithm of the training phase is given in Algorithm 3.1.

Algorithm 3.1: AllCascade Model Construction for peer p_i

input: the percentage p of support vectors to use,
the duration t to wait before merging,
local training data D_i

- 1 $SSV_i = \{\}$
- 2 $PSV_i = \{\}$
- 3 training data $T = \emptyset$
- 4 Train local classifier model M_i using RSVM on D_i
- 5 Propagate the support vectors SV_i of M_i to other peers
- 6 **while** *true* **do**
 - 7 **while** *waiting time* $< t$ **do**
 - 8 **foreach** SV_j of peer p_j received **do**
 - 9 **if** $SV_j \notin SSV_i$ and $SV_j \notin PSV_i$ **then**
 - 10 $PSV_i = PSV_i \cup SV_j$
 - 11 **if** PSV_i is not empty **then**
 - 12 $T =$ support vectors of M_i
 - 13 **forall** the $SV \in PSV_i$ **do**
 - 14 $T = T \cup SV$
 - 15 $M_i =$ SVM model trained using T
 - 16 $SSV_i = SSV_i \cup PSV_i$
 - 17 $PSV_i = \{\}$

As seen in the above illustration, note that different peers may receive the models of other peers at different times and hence the merging process may not produce the same result due to the sequence of model arrivals. In addition, when merging is performed, some of the support vectors in the previously cascade models may be missed out and thus causing the final cascaded model to be locally non-optimal. Hence, Lemma 3.1 shows that the cascade model built by AllCascade converges to the global optimal of the union of all support vectors of the peers local (RSVM) model. Therefore, AllCascade is invariant to the sequence of local models' arrivals.

Lemma 3.1 *The final classification model built by the cascade SVM with feedback loop is guaranteed to converge to the global optimal (on the original dataset).*

The *feedback loop* in Lemma 3.1 includes a repeated validation process. The validation process refers to the Karush-Kuhn-Tucker (KKT) conditions test on the original dataset after the final cascaded model for the current iteration is built. Thereafter, if there are the KKT violators, they will be merged with the support vectors of the cascaded model and the cascading process is repeated. Otherwise if there is no KKT violator, then the cascaded model will have converged to the global optimal. For the proof of Lemma 3.1, please refer to the formal convergence proof of Cascade SVM provided in [GCB⁺05].

From Lemma 3.1, it is obvious that if the support vectors of each peer's local RSVM model is taken as a data subset, then AllCascade will converge to the global optimal

where the original dataset is the union of all support vectors of all peers' local RSVM models.

Corollary 3.1 *The cascade model built by AllCascade with feedback is invariant to the arrival sequence of local models.*

Proof: Given two peers p_i and p_j with the same set of local peers' models $S_i = S_j$ but with different model arrival sequence. Since the cascade model built by AllCascade with feedback will converge to the optimal solution $model_{opt}$, therefore $model(S_i) = model(S_j) = model_{opt}$ regardless of the sequence of model arrivals.

Therefore, any peers holding the same set of local models will be able to produce the same cascade model and a peer that holds the local models of all peers will be able to arrive at the global optimal solution (based on the RSVM set).

3.1.3 Prediction

Since every peer has its own cascaded model, prediction of unseen data is simply based on the prediction of the cascaded model. Given that all peers have collected the same set of local models, the cascaded models of every peer will converge to the same solution and hence, prediction based on the local cascaded model will be sufficient without incurring any communication cost.

To summarise, the main modification of the Cascade SVM for use in the P2P environment lies in the replacement of SVM with RSVM and the ad-hoc merging of the collected models. RSVM can significantly reduce the communication overhead for distributing the data. In addition, the ad-hoc merging with local feedback allows the approach to perform incremental learning and converge to the local optimal. These improvements make it feasible to perform cascading of SVM in the P2P environments and achieve accuracy comparable to the centralized solution while reducing computation and communication cost.

3.1.4 Complexity Analysis

Next, analysis of the computation and communication costs of the proposed algorithms are conducted. The following assumptions are made to simplify the analysis process: 1) the local training dataset size ℓ_i of all peers are equal and 2) the percentage of data $s \ll 1$ (s defaults to 0.01 in the experiments) used by each peer for training the local RSVM model is the same. Given these assumptions, note that the maximum size of the support vector set resulting from the RSVM training is at most $m_i = s\ell_i$ for every peer $p_i, i \in \{1, \dots, N\}$ and the maximum size of the training set for the cascade model is $m = \sum_{i=1}^N m_i$.

Time Complexity For AllCascade, the cost of performing local RSVM training for each peer p_i is $O(\ell_i m_i^2)$ [LM01]. As the maximum size of the cascaded model is m , assuming the use of a traditional SVM solution, the cost of training the cascaded model for each peer is then $O(m^3)$. Since $\ell_i m_i^2 \ll m^3$ the time complexity of training for AllCascade is $O(m^3)$. For prediction, the time complexity is $O(mt)$ where t is the number of testing data points.

Communication Cost Analysis As the proposed approaches are independent of the underlying communication protocol, the only concern lies with the cost of propagating the data points. Assuming reliable delivery of the data, only the communication cost associated with the data propagation is computed.

For AllCascade, since every peer will perform a full propagation (sending models to all other peers), the communication cost of training is $O(Nm)$. For the prediction phase, as only the local cascaded model is used, no communication cost is required.

3.2 P2P Bagging Cascade RSVM (RandBag)

With careful observation of the AllCascade approach, one would notice that although AllCascade is able to perform accurate classification in the P2P network, the model propagation of AllCascade is similar to the flooding of request in the unstructured P2P networks, which incurs expensive communication cost. In addition, the independent collection and cascading of the local models in the P2P network by each peer causes a lot of redundancy in both the communication and computational cost.

This section presents a new approach for P2P classification by combining cascade learning with the concept of bootstrap aggregation (bagging) [Bre96, Geu00, KPJ⁺02] for learning classifiers in P2P networks. The new solution, P2P Bagging Cascade RSVM (RandBag), reduces the redundancy of AllCascade, and thus significantly reduces the communication and computation cost while maintaining accuracy comparable to that of centralized solutions.

The rest of this section presents the main processes of cascade SVM and how these processes are modified to encapsulate the concept of bagging.

3.2.1 Local Model Construction

Although using RSVM can create a small and concise representative of the local data, the full propagation of the models of all peers will still result in a significant amount of communication cost. In addition, as the prediction solely depends on the local cascaded model, this implies that accuracy is dependent on the number of models collected, which may require some substantial time initially before a satisfactory accuracy can be achieved.

Hence, in order to address these limitations, the integration of the concept of bootstrap aggregation (*bagging*) is proposed to reduce the communication cost and duplicated computation cost that is incurred due to the replicated computation of the cascade model).

The modification of the AllCascade mainly lies in the model propagation and the prediction phase, which will be discussed next. Since the construction of the local model is the same as the AllCascade, the details shall be omitted in this section and move on directly to the model propagation and prediction.

3.2.2 Model Collection

Contrary to AllCascade, RandBag adopts a pull mechanism where instead of receiving models from all other peers, a peer will now randomly choose and collect models from k peers. This can be done in the P2P networks in several ways. For instance, in unstructured P2P networks, model requests could be sent using random walk and in structured P2P networks such as distributed hash table (DHT) based, peers can be selected using randomly generated identifier key. These randomly collected models RSV are then merged, similar to the AllCascade, to create the local cascade model $CM_p = \text{SVM}(SV_p \cup RSV_1 \cup \dots \cup RSV_{k-1})$ where RSV_i is a randomly chosen set of support vector and $RSV_i \neq RSV_j, \forall i, j, i \neq j$. Observe that if the number of models to be collected k is equal to the number of peers in the P2P network N , then RandBag simplify to the AllCascade.

By collecting models of random peers, the creation of random training subsets are simulated for the cascade models; similar to sub-sampling without replacement. With the increase in the number of models collected, the computation and communication cost increases as well. However, as k the number of models collected is usually less than N the total number of peers in the P2P network, this means that only a subset of the entire model set is collected and the size of support vectors to be merged are reduced. These reduce the communication and computation cost incurred during the training phase as compared with AllCascade.

Moreover, since the model collection requires the participation of fewer peers, RandBag is also more tolerant to peers' failures as peers that are not available can simply be replaced with other peers. The algorithm of the training phase is given in Algorithm 3.2.

3.2.3 Prediction

Next, how the prediction is performed in RandBag is presented. As opposed to AllCascade which only uses the local cascaded models for the prediction, RandBag is based on the (weighted) majority voting of v randomly chosen peers (including the initializing peer). A simple approach to obtain the weight of the local cascaded model of a peer is to use the accuracy of the local cascade model on the training data. The exact process of the prediction phase is as follows:

Figure 3.2: Example of training phase for the two P2P cascade learning approaches: total number of peers (N) is 5, models to be cascaded (k) for RandBag is 3 and number of peers to vote (v) is 3

- (i) The initializing peer p_i will send the t test instances to $v - 1$ randomly chosen peers.
- (ii) Each voting peer will predict the class label of the t test instances and return the results together with the weight of the local cascaded model
- (iii) Once peer p_i received all the votes of the $v - 1$ randomly chosen peers, peer p_i will aggregate the weighted votes, including the prediction of its own local cascade model, and generate the predicted class labels for the t test instances.

If any of the selected peers are unavailable or failed to response after some time, similar to the model collection, RandBag simply request the votes from some other peers. As the number of peers required for the voting is very small, there should not be any difficulty in finding replacement peers.

As the number of voting peers increases, communication cost of prediction increases as well. This is required as a trade-off for decreasing the communication cost of model propagation. Hence, depending on the situation, one may try to optimize either the number of models to cascade k or the number of peers to vote v , to achieve the best trade-off to minimize communication cost.

With voting on the cascaded models of peers, the variance of the voting ensemble that is introduced by the random sampling can be reduced, therefore decreasing the generalization error of the prediction. Even though the best accuracy is likely to be achieved by voting with all peers in the P2P network, the experimental results show that voting with a small number of peers can also achieve satisfactory classification accuracy thereafter the negligible increase in accuracy does not justify the additional time and communication cost.

Figure 3.2 and Figure 3.3 illustrate the training and prediction phases, respectively for both AllCascade and RandBag. In Figure 3.2 and Figure 3.3, the total number of peers in the network N is 5 and the number of models collected k and the number of voting peers v is 3 for RandBag. The collected local models of peer i is represented by the data (parallelogram) labelled p_i and M represents peers' cascaded model. Locality of the peers is bounded by the outer box. Observe that during the training phase,

Figure 3.3: Example of prediction phase for the two P2P cascade learning approaches: total number of peers (N) is 5, models to be cascaded (k) for RandBag is 3 and number of peers to vote (v) is 3

AllCascade collects all peers' models while RandBag only collects k randomly selected models (inclusive of its own model) for merging. For prediction, RandBag has to requests for the votes of v peers (inclusive of its own vote), and performs weighted aggregation but AllCascade only predicts using the local cascaded model. Details of the model construction and prediction are presented in Algorithms 3.2 and 3.3 respectively.

Algorithm 3.2: RandBag Model Construction for peer p_i

input: percentage of support vectors to use (p), number of models to collect/cascade (k), local training data (D_i)

- 1 $SSV_i = \{\}; PSV_i = \{\}; models_collected = 0$; training data $T = \emptyset$;
- 2 Train local classifier model M_i using RSVM on D_i ;
- 3 $models_collected = 1$;
- 4 **while** $models_collected < k$ **do**
- 5 Collect the support vectors SV_j of a randomly chosen peer j ;
- 6 **if** $SV_j \notin SSV_i$ and $SV_j \notin PSV_i$ **then**
- 7 $PSV_i = PSV_i \cup SV_j$;
- 8 **if** PSV_i is not empty **then**
- 9 $T =$ support vectors of M_i ;
- 10 **forall** the $SV \in PSV_i$ **do**
- 11 $T = T \cup SV$;
- 12 $M_i =$ SVM model trained using T ;
- 13 $SSV_i = SSV_i \cup PSV_i$;
- 14 $PSV_i = \{\}$;

3.2.4 Coupon Subset Collection

Since it is not possible to evaluate the data of all peers in the P2P network, and without any prior knowledge, it is assumed that data of all peers are equally important. Therefore, an ideal classification model would be one that makes prediction based on knowledge of all data. Hence the following theorems are provided to show the expected number of peers that will be covered by a classification model given N , k and v and the probability of a 100% coverage.

Algorithm 3.3: RandBag Weighted Majority Voting for peer p_i

input : test instance $\mathbf{X}_t$
 local Cascade RSVM M_i
 the number p of peers to vote
output: predicted class label y_t

- 1 initialize the zeros vector $\mathbf{V}$, where size of $\mathbf{V}$ = number of classes and $\mathbf{V}_c$ = total aggregated weightage for class c
- 2 c_i = prediction of M_i on $\mathbf{X}_t$
- 3 w_i = weightage of peer p_i
- 4 $\mathbf{V}_{c_i} = \mathbf{V}_{c_i} + w_i$
- 5 propagate $\mathbf{X}_t$ to $p - 1$ other peers for voting
- 6 **foreach** vote c_j , weight w_j received from peer j **do**
- 7 $\mathbf{V}_{c_j} = \mathbf{V}_{c_j} + w_j$
- 8 c = class with max weightage in $\mathbf{V}$
- 9 return c

Theorem 3.1 *Let C be the number of unique peers involved in the prediction. Given the number of peers N in the P2P network, the number of models cascaded per peer k and the number of voting peers v , the expected number of unique peers' models involved in the prediction is*

$$E(C) = N[1 - (1 - k/N)^v]$$

and the probability that all peers will be involved in the prediction is

$$P(C = N) = \sum_{j=0}^N (-1)^j \binom{N}{j} \left[\binom{N-j}{k} / \binom{N}{k} \right]^v$$

Proof: Given that every peer randomly chooses k models without replacement for cascading, this is equivalent to every peer choosing a combination from the set A of $\binom{N}{k}$ possible combinations, where every combination has the same probability of being chosen. By counting the number of distinct peers involved in the prediction based on the v number of cascaded models, this problem can be generalized as a special case of the coupon collector's problem in the case of sampling in groups of constant size [Sta90].

Given $A \subset S$, $\ell = |A|$, $s = |S|$, $\omega \subset S$ and $|\omega| = k$, suppose v sets of ω are drawn from S with replacement and that each possible ω has equal probability of being drawn. The coupon collector's problem in the case of sampling in groups of constant size is the counting of the distinct elements of A contained in at least one of the v drawn subset. The problem here is a special case where $S = A$, as such, the expected value of distinct elements drawn is

$$\ell[1 - (1 - k/s)^v] = N[1 - (1 - k/N)^v]$$

where $\ell = s = N$. The probability of the n distinct elements drawn for a given k and v is

$$\binom{l}{n} \sum_{j=0}^n (-1)^j \binom{n}{j} \left[\binom{s+n-l-j}{k} / \binom{s}{k} \right]^v$$

which is equivalent to

$$\sum_{j=0}^N (-1)^j \binom{N}{j} \left[\binom{N-j}{k} / \binom{N}{k} \right]^v$$

for $\ell = s = n = N$.

In Theorem 3.1, the expected coverage of the peers is based on the situation where voting peers can be chosen with replacement. Such a selection will in fact increase the frequency of duplicate peers. Note that it is possible to select peers for voting without replacement using methods such as creating unique testing tokens to avoid repeated votes and choosing voting peer from DHT based P2P networks using unique identifier key. Hence, the implemented solution selects the voting peers without replacement, to improve the expected coverage.

3.2.5 Complexity Analysis

Next, analysis of the computation and communication costs of the proposed algorithms are conducted. The following assumptions are made to simplify the analysis process: 1) the local training dataset size ℓ_i of all peers are equal and 2) the percentage of data $s \ll 1$ (s defaults to 0.01 in the experiments) used by each peer for training the local RSVM model is the same. Given these assumptions, it is noted that the maximum size of the support vector set resulting from the RSVM training is at most $m_i = s\ell_i$ for every peer $p_i, i \in \{1, \dots, N\}$ and the maximum size of the training set for the cascade model is $m = \sum_{i=1}^N m_i$.

Time Complexity For RandBag, the cost of performing local RSVM training for each peer p_i is $O(\ell_i m_i^2)$ [LM01]. As each peer only collects k RSVM models, the maximum size of training data for cascading in RandBag is km/N . Assuming the use of a traditional SVM solution, the cost of training the cascaded model for each peer is then $O((km/N)^3)$. Since $\ell_i m_i^2 \ll (km/N)^3$, the time complexity of training for RandBag is $O((km/N)^3)$. For prediction, the time complexity is $O(kmt/N)$.

Communication Cost Analysis As the proposed approaches are independent of the underlying communication protocol, the only concern lies with the cost of propagating the data points. Assuming reliable delivery of the data, only the communication cost associated with the data propagation is computed.

For RandBag, since every peer only collects k peers' models, the communication of training is $O(km)$. For the prediction phase, as $v - 1$ votes are required from other peers, RandBag needs to send the t testing data points to $v - 1$ randomly selected peers. Therefore the communication cost of prediction is $O(vt)$.

3.3 Communication-efficient Multiple Parameter Robust (CEMPaR) Framework

It is noticed that the problem of heavy communication cost due to high interaction between peers is common for most of existing P2P classification approaches, which is usually caused by either training model propagation or test instance propagation. Therefore, the aim is to address the challenge of reducing large amount of peer communication cost required. To this end, CEMPaR, an effective P2P classification scheme has been proposed to satisfy the critical constraints of performing classification in P2P networks (e.g. scalability, asynchronism, and peer and data dynamism).

The rest of this section presents the CEMPaR framework that achieves significant reduction in communication and computational cost by exploiting the properties of structured P2P network (DHT) with a hybrid P2P overlay solution (with the introduction of dynamic server-liked peers).

3.3.1 Structured P2P Networks

DHT-based P2P Networks. DHT-based P2P networks are popular because they provide efficient message routing for resource discovery. These approaches generally use consistent hashing; i.e., they assign each peer a unique identifier in the identifier ring space. Chord [SMK⁺01] — a DHT-based lookup protocol, assigns identifiers from 0 to 2^b , where b is the number of bits for the identifier key. Using consistent hashing, identifier assignments remain unaffected by dynamic peers who join or leave arbitrarily. Moreover, there is a high probability that peers are well distributed in the identifier-ring space. As for data, they are hashed in a similar process and allocated to the node whose identifier is closest to (but not smaller than) the generated key. Each peer indexes a small number (b) of other peers' physical addresses. A resource can be found (or message routed) using its key by recursively looking up peers' indexes. This efficient divide-and-conquer approach of the identifier-ring space requires a number of hops at most logarithmic to the size of the network. The following is a key property of DHT protocols.

Property 3.1 *Given a DHT with a circular identifier key space (e.g., Chord), whenever a message is sent to key i , if the peer with the key exists, then the message will be delivered to the peer; otherwise, it will be routed to the peer with the next sequentially larger key (i.e., $\text{peer}(k + x)$ where $x > 0$ and x is minimum).*

Figure 3.4 illustrates a Chord identifier ring. N21 receives a message sent to key 21, N32 receives a message sent to key 25 and N1 receives a message sent to key 63.

3.3.2 Communication Structure Overlay

To reduce peer interactions considerably, the notion of *super-peer* in the P2P network was introduced. The super-peers are dynamically selected from peers in the P2P network such that each super-peer is a representative of some subset of peers in the P2P network. The usage of super-peers significantly reduces the huge amount of P2P communication among peers, but the difficulty is that peers may not be able to locate their associated super-peers since peers in a P2P network usually know only a small number of their own neighbors. In order to address this challenge, DHT-based P2P network protocols [BKK⁺03] (e.g. Chord [SMK⁺01]) has been applied to facilitate the tasks of resource discovery and communication. Below, an efficient communication overlay scheme built upon DHT-based network protocols is presented.

In the proposed approach, the whole identifier ring space of a DHT-based network is equally split into g groups (with consideration to peer distribution, load balancing, and ease for super-peer assignment), where *group* is formally defined below:

Definition 3.1 (*Group*) *A group $\mathbf{G}$ is a contiguous subset of the identifier ring space. Given that the identifier ring space is evenly split, and denote by g the number of groups and b the number of bits for an identifier key, then the number of identifiers contained in each group is $|\mathbf{G}| = 2^b/g$, i.e., $\mathbf{G} = [(2^b/g * i), (2^b/g * (i + 1))]$ for group $i \in [0, g)$.*

Next, for each group, a super-peer is assigned among the peers to represent and manage the group. The super-peer assignment should be easily managed and efficient for discovery by the peers. By exploiting the property of DHT-based network as shown in property 3.1, a simple yet effective super-peer assignment approach is suggested and formally defined below:

Definition 3.2 (*Super-peer*) *Given a set of peers whose identifiers $\mathbf{P}$ are a subset of an identifier group $\mathbf{G}$ in a DHT-based network, i.e., $\mathbf{P} \subset \mathbf{G}$, a super-peer will be assigned as the peer of the smallest identifier in $\mathbf{P}$: $s_p = \min_{i \in \mathbf{P}} i$.*

The above approach enables CEMPaR to easily and deterministically locate the super-peers using the DHT look-up service. In particular, by simply sending a message to the smallest identifier of a group, the message is guaranteed to be sent to the super-peer of the associated group, as shown in Corollary 3.2.

Corollary 3.2 *For a set of peers $\mathbf{P} \subset \mathbf{G}$, a message sent to the smallest identifier in group $\mathbf{G}$, denoted as s_g , will be always delivered to the super-peer of $\mathbf{P}$ in the time complexity of $O(\lg N)$.*

Figure 3.4: Identifier ring with 10 peers in 4 groups. Round and square nodes represent peers and super-peers respectively.

Proof: Given that the smallest identifier in group $\mathbf{G}$ is $s_g = \min_{i \in \mathbf{G}} i$ and the super-peer's identifier of $\mathbf{P}$ is $s_p = \min_{i \in \mathbf{P}} i$, since $\mathbf{P} \subset \mathbf{G}$, thus $s_g \leq s_p$. Therefore, according to the property of DHT-based network in property 3.1, the message sent to s_g will be guaranteed to be sent to s_p in the time complexity of $O(\lg N)$ according to the nature of DHT-based networks [BKK⁺03].

Therefore, the proposed approach allows only the super-peers to receive the models from other peers, and to make predictions on unseen test data for a classification task. Figure 3.4 illustrates the identifier ring of the proposed approach.

Remark. Note that the assumption that each group has at least one peer whose identifier lies in the group's identifier range may not be always satisfied, e.g., when the number of peers in the P2P network is less than the number of groups. However, it only affects the condition that the key of the super-peer must lie in group $\mathbf{G}$, not affecting the delivery of a message to the super-peer, as the identifier key overflows to the next group whose super-peer may be the super-peer of more than one group. This exceptional situation will be rectified by the relocation process (discussed later) when a peer with an identifier $id \in \mathbf{G}$ joins.

By introducing the concepts of *group* and *super-peer*, an efficient communication scheme in CEMPaR is developed. The communication scheme solves two critical tasks: (1) the discovery of super-peers, and (2) the communication between peers. In particular, two efficient solutions built upon the DHT-based protocols below are offered.

DiscoverSP(gid, irv) — this function uses the underlying DHT look-up protocol to route the message containing information request vector irv and sender's physical address to the super-peer of group id gid using the first identifier of the group. The irv encodes the sender's request for information such as physical address of receiver, mean vector, class counts, etc. This function incurs $O(\lg N)$ messages which is optimal as opposed to a linear search of the identifier ring costing $O(N)$ (c.f. [SMK⁺01]). Hence, this function is best used when the physical address of the recipient is unknown. The

Figure 3.5: Flowchart of CEMPaR processes. Dotted arrow indicates a network communication.

size of the message to be sent is very small, which includes 1-byte for *irv*, and 5-byte for the sender's physical address.

SendMsg(*rip*, *data*) — this aims to send the message containing sender's physical address, a set of content *data*, e.g., model, class count, mean vector, replica list, and etc, directly to the recipient's physical address *rip*. As the message is sent directly to the recipient, it is optimal ($O(1)$) and is best for sending large data. The size of the message includes 5-byte for the sender's physical address and the size of the content.

3.3.3 Learning Modules

Next, the core-learning modules for performing training and prediction tasks in the proposed framework are discussed. Figure 3.5 shows an overview of the whole processes in CEMPaR. The details of the key modules are discussed below.

Local Model Construction RSVM [LL03] is adopted for training local models for peers in each group since RSVM produces the training model containing support vectors that are at most s percent of the total training data for a local dataset [AGHN08, AGH⁺08, AGHN13], which in turn caps overall communication and learning cost.

Model Propagation and Cascading Following the local model construction is the processes of model propagation and cascading, in which peers send the local models to super-peers, and super-peers collect the models from peers and update the cascaded classification models.

One key issue for the model propagation is to determine which super-peer should a peer propagate its local model. A naive way is to send the model to all super-peers. Apparently, this is inefficient due to intensive communication and computation cost. Ideally, the local model should be sent to the best super-peer, which results in the best global classification performance. Unfortunately, in a P2P network, optimizing the global classification performance is often intractable.

In practice, the cascaded models of the super-peers should be as diverse as possible. This is because in ensemble classification, the best classification performance is often achieved when the models are diverse [Pol06]. In addition, previous experience [AGH⁺08, AGHN13] indicates that it would be ideal to ensure that every super-peer maintains an (approximate) equal class and data size distribution. This is achieved by balancing the load distribution and maintaining the natural class distribution of data on each super-peer. Although natural class distribution may not produce the best classification results [WP01], it provides an overview of the global class distribution to allow cost-sensitive learning. Finally, it is also desirable to reduce the overall redundancy in computation and communication cost.

To this end, a greedy approach for model propagation is proposed. When a peer p is ready for propagation, it first collects information from all super-peers, including the number of collected instances (for each class), and then the mean vector of the collected data (for each class) for each super-peer, via *DiscoverSP*. With the collected data, for each class type, the super-peer with the smallest instance count will be chosen, and meanwhile the instance from the local model that is closest to the mean vector of the selected super-peer will be assigned. This process repeats until all instances in all classes have been assigned. As the instances are assigned in a disjoint manner, duplicate communication cost can be avoided. Finally, the assigned data are sent to super-peers by using *SendMsg* since peer p has already obtained the physical address of all super-peers via *DiscoverSP*.

Note that to minimize the discrepancy of class count when multiple peers are performing model propagation, peers can first calculate the class count to be assigned and send the counts to the super-peer via *SendMsg* before the assignment of the support vectors; once all support vectors are assigned, they will be propagated to the respective super-peers. Finally, the model propagation algorithm is summarized in Algorithm 3.4.

Once the super-peers have received the models, in addition to merging the newly collected instances with the super-peer's cascaded model, each super-peer will also update the mean vector for the set of instances of each class and the instance count of each class. The model cascading algorithm is summarized in Algorithm 3.5.

Remark. In a stable network, the communication cost for the above model propagation process is only $O(m)$ where m is the total number of support vectors of all local models. In practice, as a P2P network is in nature highly dynamic, additional cost might be incurred to ensure correctness and robustness. Issues of relocation and replication will be discussed in subsequent parts.

Algorithm 3.4: Model Propagation for peer p_i .

```

input: number of groups  $g$ , local support vectors  $\mathbf{SV}_i$ 
1 for  $j \leftarrow 0$  to  $g - 1$  do
2   |  $\mathbf{MV}, \mathbf{CC}, \mathbf{IP} \leftarrow \text{DiscoverSP}(j, \text{irv}; \text{mean vector } (\mathbf{MV}), \text{class count } (\mathbf{CC})_i);$ 
3   foreach class label  $y$  in  $\mathcal{Y}$  do
4     | while  $\mathbf{SV}_i^y$  not  $\emptyset$  do
5       |  $j \leftarrow$  group with the least count of class  $y$  in  $\mathbf{CC}$ ;
6       |  $sv \leftarrow$  closest support vector in  $\mathbf{SV}_i^y$  to  $\mathbf{MV}_j$ ;
7       | remove  $sv$  from  $\mathbf{SV}_i$  and add it to  $\mathbf{SV}_j$ ;
8       | update  $\mathbf{CC}_j$ ;
9   for  $j \leftarrow 0$  to  $g - 1$  do
10  |  $\text{SendMsg}(\mathbf{IP}_j, \mathbf{SV}_j);$ 
    
```

Algorithm 3.5: Model Cascading for super-peer s_i .

```

input : received model  $RM_j$ , collected data  $\mathbf{CD}_i$ , cascaded model  $CM_i$ , mean
        vector  $\mathbf{MV}_i$ , class count  $\mathbf{CC}_i$ 
output:  $CM_i, \mathbf{MV}_i, \mathbf{CC}_i$ 
1  $\mathbf{CD}_i \leftarrow$  combine received model  $RM_j$  with  $\mathbf{CD}_i$  ;
2  $CM_i \leftarrow$  train SVM on local cascade model  $CM_i \cup$  received model  $RM_j$  ;
3 foreach class label  $y$  in  $\mathcal{Y}$  do
4   | update  $\mathbf{CC}_i^y$  and  $\mathbf{MV}_i^y$ ;
    
```

Prediction During prediction, since only g super-peers are performing data collection and cascading the models, peers that need to predict unseen data can simply send the test instances to these super-peers and then aggregate the votes returned by the super-peers. However, it will incur heavy computation load on the super-peers.

Therefore, the replication of the super-peers' cascaded models (c.f. Section 3.3.4) is proposed. With the replicas, peers requesting prediction will request the replica list (containing physical addresses of replicas) of super-peers via *DiscoverSP*. Then, for every replica, by sending a *ping* message and with the reply from the replica (via *SendMsg*), a round trip time (RTT) is obtained. The RTT measures the network distance from the initiating peer to the replica. The initiating peer will then send the test instances to the nearest replica of each group. Once the replicas has finished predicting the test instances, they will send their predictions back to the initiating peer via *SendMsg*. The initiating peer will then aggregate all votes to make the final prediction once all replies are received. The communication is efficient as all the messages sent are based on optimized communication functions. Finally, in order to reduce the communication of pinging replicas, caching of the RTT could be done for use in subsequent prediction. The prediction algorithm is summarized in Algorithm 3.6.

3.3.4 Maintenance Modules

Relocation In a P2P network, peers are free to join and leave the network anytime in an ad-hoc manner. Hence, there are frequent changes in the identifier space which

Algorithm 3.6: Prediction.

```

input  : test instance  $\mathbf{t}_i$ , number of groups  $g$ 
output: prediction  $y_i$ 
1 for  $j \leftarrow 0$  to  $g - 1$  do
2   |  $\mathbf{RL} \leftarrow \text{DiscoverSP}(j, \text{irv}; \text{replica list } (\mathbf{RL})_i);$ 
3   for  $j \leftarrow 0$  to  $g - 1$  do
4   |   foreach replica  $r \in \mathbf{RL}$  do
5   |   |    $\text{SendMsg}(r, \text{"ping"})$ ;
6    $\mathbf{PL} \leftarrow \text{select nearest replica of each group}$ ;
7   for  $j \leftarrow 0$  to  $g - 1$  do
8   |    $\mathbf{V}_j \leftarrow \text{SendMsg}(\mathbf{PL}_j, \mathbf{t}_i)$ ;
9    $y_i \leftarrow \text{select class with most votes in } \mathbf{V}$ ;
```

affects the assignments of super-peers. As the local models will always be propagated to the super-peers (c.f. Corollary 3.2), which could be different as time goes, relocation of data from the previous super-peer is needed to ensure that the newly selected super-peer will always hold the group's cascaded model.

Relocation is needed when a new peer that has an identifier key smaller than the group's current super-peer and the identifier key is within the range of the group's identifier key space. It may also be needed when new data are received and if the receiving peer is not the current super-peer (status request sent from the old super-peer). Note that the tracking of such peer changes requires no effort on CEMPaR's part since Chord tracks and notifies such events.

The relocation process is as follows. When a new peer joins the group with the smallest identifier key within the group, it is elected as the new super-peer and the data from the old super-peer will be relocated to the new super-peer. As the DHT network provides the physical address, in addition to the identifier key of a new peer. The cascaded model can be relocated in an efficient manner via the *SendMsg* function.

Instead of the relocation approach, there are other options to ensure that the super-peer will always be the peer with the smallest identifier key in the groups. For instance, 1) disallowing the allocation of keys smaller than the current super-peer, and 2) performing a switch of identifier keys between the new peer with the smaller identifier key and the existing super-peer.

It seems that the above two alternative approaches may be more efficient; however both of them have their own problems and may not necessarily reduce communication cost. For the first approach, it will create an imbalanced identifier ring and thus increase the chance of identifier key collisions. Given that the algorithm replicates the super-peer's cascaded models to its successive peers (discussed later), the second alternative approach will result in the same communication cost as relocation; in addition it may incur extra communication cost for fixing the routing indexes of other peers.

Therefore, the relocation of models is preferred over the manipulation of the identifier keys to keep the proposed approach simple and efficient without any modifications to the underlying P2P DHT protocol.

Replication Since data are only propagated to the super-peers, failure of any of these super-peers can result in catastrophic outcomes. This greatly reduces the overall classification accuracy and diminishing the purpose of model propagation, which is to try to maintain the availability of the peers' models in the P2P network in order to improve accuracy.

In addition, since it is necessary for all the cascade models involved in the prediction phase, the computation and communication load of these super-peers will be very high, resulting in low throughput for the prediction tasks.

Hence, in order for the proposed approach to be tolerant to peer dynamism while reducing the load of the super-peers (during predictions), replication of the cascaded models is needed. To create the replicas, a concept similar to the proposed proximity routing of Chord has been adopted, where each super-peer will replicate the cascaded models to r successive peers. Since the identifier assignment of the peers is very random, it is very likely that peers reside in different geographical locations; thus, speeding up the access of neighboring peers. In addition, with multiple peers having the cascaded models, the load of the super-peer for both communication and computation can be reduced substantially. Moreover, it is unlikely that all the replicas from the same group fail simultaneously (c.f. Proposition 3.1).

Proposition 3.1 *Given an initially stable network where every super-peer has propagated its cascaded model to its r successive peers (replicas) and assume that every peer fails or leaves with probability $1/2$, there is a high probability that a prediction is made based on all the g cascaded models (e.g., $\geq 99\%$ when $r \geq 6$).*

Proof: Since each super-peer has already propagated its cascaded models to its r successive peers, then in order for a prediction to be not based on all the g cascaded models, it would require the failure of at least one entire group's super-peer and replicas. This implies that all the $r + 1$ peers will have to fail where the probability is $(1/2)^{r+1}$, and with $r \geq 6$, the probability will be less than 0.01, hence it is with high probability a prediction would be based on all the g cascaded models.

Replication is only needed when new data are collected or when there is a change in the neighboring peers (i.e., when a new peer joins to become one of the r successive peers or when a replica has left), and such events will be notified by Chord. When peer p 's predecessor has left or failed, peer p will send an election message to the group's super-peer. In the event that the failed or left peer is a super-peer, its immediate successor (in the identifier key space) will receive the election message and become the new super-peer of that group. In order to minimize the amount of data replicated, before replicating, the super-peer will first check with each replica to see what data they are holding. After which, the super-peer will then send the missing data to the replicas.

Since Chord already maintains a list of sibling nodes (successive in the key values) with their physical address, tracking the changes of these successive peers will be trivial by retrieving the list from Chord (which does the actual job). Replication to the new successive peers (either newly joined or to replace a peer that has left) will be handled by the super-peer. In addition, note that peers will always retain their collected data to reduce data propagation in the event of any changes in the super-peer or replicas. All communications are done in an efficient manner using SendMsg as the physical addresses of all replicas are known.

3.3.5 Complexity Analysis

Next, an analysis of the computation and communication costs of the proposed algorithms is presented. The following assumptions are made to simplify the analysis process: 1) the local training dataset size ℓ_i of all peers are equal and 2) the percentage of data $s \ll 1$ (s defaults to 0.01 in the experiments) used by each peer for training the local RSVM model is the same. Given these assumptions, note that the maximum size of the support vector set resulting from the RSVM training is at most $m_i = s\ell_i$ for every peer $p_i, i \in \{1, \dots, N\}$ and the maximum size of the training set for the cascade model is $m = \sum_{i=1}^N m_i$.

Time Complexity For AllCascade, the cost of performing local RSVM training for each peer p_i is $O(\ell_i m_i^2)$ [LM01]. As the maximum size of the cascaded model is m , assuming the use of a traditional SVM solution, the cost of training the cascaded model for each peer is then $O(m^3)$. Since $\ell_i m_i^2 \ll m^3$ the time complexity of training for AllCascade is $O(m^3)$. For prediction, the time complexity is $O(mt)$ where t is the number of testing data points.

For RandBag, the cost of performing local RSVM training for each peer p_i is also $O(\ell_i m_i^2)$. As each peer only collect k RSVM models, the maximum size of training data for cascading in RandBag is km/N . Assuming the use of a traditional SVM solution, the cost of training the cascaded model for each peer is then $O((km/N)^3)$. Since $\ell_i m_i^2 \ll (km/N)^3$, the time complexity of training for RandBag is $O((km/N)^3)$. For prediction, the time complexity is $O(kmt/N)$.

From the above time complexity, it can be seen that RandBag incurs less computational cost, $(k)/N$ of AllCascade.

Communication Cost Analysis As the proposed approaches are independent of the underlying communication protocol, the only concern lies with the cost of propagating the data points. Assuming reliable delivery of the data, only the communication cost associated with data propagation is computed.

For AllCascade, since every peer will perform a full propagation (sending of models to all other peers), the communication cost of training is $O(Nm)$. For the prediction phase, as only the local cascaded model is used, no communication cost is required.

For RandBag, since every peer only collects k peers' models, the communication of training is $O(km)$. For the prediction phase, as $v - 1$ votes are required from other peers, RandBag needs to send the t testing data points to $v - 1$ randomly selected peers. Therefore the communication cost of prediction is $O(vt)$.

Note that with the reduction of communication cost during training phase for RandBag (k and compared to N for AllCascade, where $k < N$), this saving of communication cost is compensated during the prediction phase where AllCascade does not incur any communication overhead.

3.4 P2P Adaptive Classification Ensemble (PACE) Framework

This section presents the proposed novel P2P classification approach, the P2P Adaptive Classification Ensemble (PACE) framework. PACE is an improvement of the typical voting ensemble, which has several advantages for classification in P2P networks. Firstly, voting ensemble is a loosely coupled algorithm, which means that it does not require high-level synchronization. Secondly, as it does not have the constraint of requiring all models to participate in the voting, it is able to provide a partial solution anytime [LXLS07]. This also means that it is *tolerant to peer dynamism* where failures of a few peers will only slightly affect the final prediction. Hence, voting ensemble is very suitable for classification in the P2P environment. However, the regular voting ensemble approaches also have several critical drawbacks. As discussed in the introduction section, they often suffer from several critical scenarios, including *disjoint data distribution*, *skewed class distribution*, and *disjoint class distribution*, etc.

This section presents PACE, which aims to maintain the advantages of the regular voting ensemble solution meanwhile effectively overcoming their drawbacks. In particular, it is noticed that the regular voting ensemble is often *static* in nature, i.e., the same voting scheme is applied *universally* for every test data sample which does not exploit test data distribution. In addition, they often engage all classifiers or peers for prediction which may not be efficient especially when the number of peers is huge. Unlike conventional ensemble approaches, PACE is novel in that it is **aware** of test data distribution, and employs a **dynamic** voting scheme, which only chooses a subset of important classifiers or peers for making prediction on a test data sample. To facilitate the selection of important classifiers with respect to test data distribution, a cluster-driven approach is proposed, which provides an efficient way to examine how a test data sample is close to a specific peer. Furthermore, to combine the outputs from the subset of the selected classifiers, a k nearest neighbor weighted voting approach is adopted, which exploits

strategy, the number of weight vectors is the number of classes minus one), which will significantly reduce communication cost incurred for model propagation.

As noted earlier, simply combining the ensemble of classifiers by majority voting is insufficient to guarantee satisfactory classification accuracy. However, without additional data it is not possible to perform model selection or use advanced model combination techniques. Unfortunately, considering the size of the P2P network and the communication cost, it is not possible to manipulate the data as required by existing ensemble model selection or advance combination techniques. Assuming that the classifier's accuracy correlates to the distance between the testing and training data, the distance between the testing and training data or the locality of the training data have to be somehow captured. Hence, the use of clustering is proposed to capture the locality of the training data of each peer.

Clustering will be performed on the training data D_i of peer p_i to generate the set of centroids C_i of size c which will be representative of the training data of the classifier. The centroid serves as a summarization of a group of data examples. By using only a small number of centroids, the additional overheads on the communication cost will be reduced substantially. Empirical results show that the inclusion of centroids will ensure robust ensemble performance. Here, the simple and efficient k -means clustering algorithm is employed, in which the number of clusters can be specified, which is directly proportional to the communication cost.

Note that the training and the clustering steps can be performed either concurrently or sequentially. However, in addition to the locality information, it is also desirable to capture the classifier's classification accuracy on the particular centroid. This mainly aims to address the problem of *skewed class distribution*. Note that a classifier trained on an imbalanced dataset will often have a high error rate. Hence, even if the classifier is trained on data near to the test data, it might be possible that the classifier's accuracy is not at an acceptable level. Hence, the accuracy of the centroid can be used as the balancing parameter. In order to obtain the error rate of the classifier on the cluster, both the training and clustering have to be completed.

Algorithm 3.7: Training and Clustering Phase.

input : Local data D_i , number of cluster c
output: classification model M_i , centroids C_i , model error rate Err_i

- 1 $M_i \leftarrow trainClassifier(D_i)$;
- 2 **Clusters** $\leftarrow clusterData(D_i, c)$;
- 3 $C_i \leftarrow computeCentroids(Clusters)$;
- 4 $Err_i \leftarrow predictionTest(M_i, D_i)$;

Once the cluster testing is completed, the model together with the centroids and their error rates are sent to other peers. Note that the collection of other peers' models and centroids is a progressive process since new peers may join anytime. As the underlying prediction mechanism of the prediction phase is voting, addition of new models will not be a problem. The training and clustering phase is summarized in Algorithm 3.7.

3.4.2 Prediction Phase

Assuming that not all classifiers will fulfil the accuracy criteria required for ensemble classifiers, there would be a need to filter out those irrelevant classifiers to prevent them from adversely affecting the ensemble's accuracy. Figure 1.1 illustrates **scenario 1**. Given a test instance represented by *, note that a regular ensemble consisting of all the classifiers $M1$, $M2$ and $M3$ will incorrectly classify the test instance. However, if a model (subset of all models) which is trained on data nearest to the test instance was selected, e.g. $L3$ in this case, it would be more likely to be able to correctly predict the test instance.

Let $dist(D_i, T)$ denote the distance from test instance T to the training data of peer p_i in the feature space, and $P_{err}(M_i, T)$ denote the confidence probability of a classifier M_i wrongly classifying T . The following lemma is introduced to show the relationship between two models on a test instance as follows:

Lemma 3.2 *When $dist(D_i, T) > dist(D_j, T)$, then $P_{err}(M_i, T) > P_{err}(M_j, T)$ if $Err_{emp}(M_i) = Err_{emp}(M_j)$.*

Proof: Note that when $dist(D_i, T) = 0$, T is located within the distribution of the training data D_i , hence $P_{err}(M_i, T) = Err_{emp}(M_i)$. However as $dist(D_i, T)$ increases, T moves further away from the distribution of the training data D_i , i.e., T becomes part of the unseen / validation set; $P_{err}(M_i, T)$ approaches $Err_{exp}(M_i)$ as $Err_{exp}(M_i)$ increases. Therefore $dist(D_i, T) > dist(D_j, T)$ implies $Err_{exp}(M_i) > Err_{exp}(M_j)$ and $Err_{exp}(M_i) > Err_{exp}(M_j)$ implies $P_{err}(M_i, T) > P_{err}(M_j, T)$. ■

Therefore, using Lemma 3.2 as the basis, a better estimation on the expected error of the test instance T can be provided. Given that the criteria for the classifier of the ensemble is based on Lemma 3.2, issues 1, 2 and 3 can be addressed. However, Lemma 3.2 assumes equal empirical error among the classifiers, which is noted earlier as an incorrect assumption. In addition, the distance should be factored in as an important component for ensemble model selection. Hence, it is proposed to combine the empirical error $Err_{emp}(M_i)$ and the distance $dist(D_i, T)$ as the weight for the classifier in the ensemble:

$$\begin{aligned} W(M_i, T) &= 1 - P_{err}(M_i, T) \\ &= (1 - Err_{emp}(M_i)) * w(dist(D_i, T)) \end{aligned} \tag{3.1}$$

where $w(\cdot)$ is an inverse distance function. However, it is not possible to perform the distance measurement between all test data example and all peers' training data. Hence, the set of centroids C_i that were generated from the training data of peer p_i are used to represent the training dataset D_i and the distance is computed as $dist_{min}(C_i, T) = \min_{cd \in C_i} dist(cd, T)$, where T represents the test instance and cd represents the individual centroid.

Observe that the weighting for a classifier is based on Equation 3.2, which comprises an inverse distance function $w(\cdot)$ and the minimum centroid distance $dist_{min}(C_i, T)$. Depending on the inverse distance function $w(\cdot)$, the effect of the minimum centroid distance on the final weighting varies. Moreover, since centroids were used instead of the actual training data, there will be some loss in the accuracy of the distance measure. It is obvious that $W(M_i, T)$ is not equivalent to $Err_{exp}(M_i)$. Hence, it is not possible to determine which classifier fulfils the criteria of having $Err_{exp}(M, T) < 0.5$. Therefore, instead of choosing classifiers that meet the accuracy criteria, ranking is performed with the top v classifiers chosen. Since every classifier C_i minimizes $Err_{emp}(M_i)$, using the error estimate as the ranking criteria may create unnecessary bias. Hence, all classifiers M are ranked according to their distance to the test instance ($dist_{min}(C, T)$). To minimize of adverse effects of erroneous classifiers, only the the top v models are chosen. Since it is not possible to determine Err_{exp} , the choice of v will only be served as an estimation. Hence, the choice of v and determination of its effect on the classification accuracy are examined empirically.

Note that the ranking of models has to be performed for each test instance since their distribution will vary so will their v nearest neighbors. A naïve approach is to compute the distance between all classifiers' centroids each and every time a new instance arrives. However, this will be too costly as each ranking will incur gN distance computation. Hence, to maintain high efficiency of PACE, the use of a distance aware indexing algorithm such as k-d tree [BEK⁺98] or locality-sensitive hashing (LSH) [AI06] is proposed. However, due to time constraint, indexing is not implemented here. Nevertheless, the rest of this work shall be presented based on the assumption of the presence of an index. Note that given an index, the indexing is only performed once and thereafter perform retrieval based on the index. On the arrival of a model, the index will be updated using the centroids which will allow the retrieval of the v -nearest classifier.

With consideration to the ranking of the models and the modified distance measure, Equation 3.1 is modified as:

$$W(M_i, T) = (1 - Err_{emp}(M_i, c_{min})) * w(d') \quad (3.2)$$

where $d' = dist_{min}(C_i, T)$ or $d' = rank$.

One can observe that the inverse distance function $w(\cdot)$ is yet to be defined. Note that the main purpose of $w(\cdot)$ is to increase the weight of the “closer” models (*research problem 2*). As such, the following k NN based weighting schemes are examined:

- $w_0(d') = 1$
- $w_1(d') = 1/d$
- $w_2(d') = k - modelrank$
- $w_3(d') = \frac{dist_{min}(C_{last}, T) - dist_{min}(C_{current}, T)}{dist_{min}(C_{last}, T) - dist_{min}(C_{first}, T)}$
- $w_4(d') = e^{-\lambda rank}$

Algorithm 3.8: Prediction.

input : test instance T , set of all peers' classifiers $\mathbf{M}$, centroids of all peers' training data $\mathbf{C}$, prediction errors of all peers' models $\mathbf{Err}$, size of ensemble v ;
output: prediction y ;
1 weighted votes counts $\mathbf{VC}$;
2 list of v classifier ;
3 $\mathbf{TopV} \leftarrow$ retrieve the set of nearest v models;
4 **for** $i \leftarrow 1$ **to** $|\mathbf{TopV}|$ **do**
5 $y_i \leftarrow \text{predict}(\mathbf{TopV}_i, T)$;
6 increase $\mathbf{VC}_{y_i}$ by $((1 - \mathbf{Err}_i) * w(\text{rank}(\mathbf{TopV}_i) | \text{dist}(\mathbf{TopV}_i, T)))$
7 $y \leftarrow \text{getClassWithMaxVote}(\mathbf{VC})$;

A brief summary of the preprocessing and prediction phase is provided as follows. Firstly, when a new classifier and its centroids are received, the centroids are indexed and the classifier stored. This indexing step is critical to maintaining high efficiency for the prediction phase. Next, when a test instance T arrives, the v nearest models will be retrieved from the index. Next, compute the centroid distance to the test instance or simply record the rank. Then the prediction of the classifier, multiplied by its training error and nearest neighbor weight $w(\cdot)$ is stored. Finally, compute the largest voted class, which is the prediction of the ensemble. The pseudocode of the prediction phase is presented in Algorithm 3.8.

3.4.3 Complexity Analysis

Here, a time complexity analysis of PACE is provided. In the training phase, each peer will build a LIBLINEAR SVM classifier. The cost of building the linear SVM model is $O(\log(1/\epsilon)\ell_i d)$ for an ϵ -accurate solution where d is the dataset dimension [HCL⁺08]. Other than the model construction, peers will also cluster the local data which cost $O(\ell_i c d)$ [AV07]. Once both model construction and clustering are completed, the training data is evaluated with costing $O(\ell_i d)$. In addition, upon the arrival of a peer's model and centroids, the distance indexes for the models are updated. Since this is not a part of the prediction, it shall be computed as a part of the training cost. As the datasets used in the experiments are all high dimensional, LSH [AI06] is used as the indexing algorithm. Given a $(1, c', p1, p2)$ -sensitive hash function for $\mathbb{R}^d$, the cost of constructing the index is $O((d + \tau)(gN)^{1/c'^2 + o(1)} \log_{1/p2}(gN))$, where τ is the time to compute the hash function. Hence, assuming that $k \ll \log(1/\epsilon)$, the worst case time complexity for training and clustering is $O(\log(1/\epsilon)\ell_i d)$.

In the prediction phase, when an instance arrives, the pre-computed index is used to retrieve the top v nearest neighbors. Retrieval of these v nearest neighbors cost $O(d(cN)^{1/c'^2 + o(1)})$. Finally, prediction is performed for the top v models costing $O(vd)$. Hence, the worst case time complexity for prediction is $O(d(cN)^{1/c'^2 + o(1)})$.

Table 3.1: Description of symbols.

Param.	Description
N	number of peers in P2P network
ℓ_i, ℓ	number of instances in peer p_i , ($= N\ell_i$)
s	percentage of training dataset to build RSVM
m_i, m	size of RSVM model in peer p_i ($= s\ell_i$), entire network
M_{sv}	size of SVM model
k	number of models collected (typically 10%)
v	number of peers voting (typically 10%)
g	number of groups (typically 10)
c	number of clusters (typically 10%)
t	number of test instances
ϵ	Stopping criteria for linear SVM

Table 3.2: Cost comparison.

Approach	Training		Prediction	
	Time	Comm.	Time	Comm.
Centralized Approaches				
SVM	$O(\ell^3)$	-	$O(M_{sv}t)$	-
RSVM	$O(\ell m^2)$	-	$O(mt)$	-
LinSVM	$O(\log(1/\epsilon)\ell)$	-	$O(t)$	-
P2P Approaches				
SVME	$O(\ell_i^3)$	-	$O(M_{sv}t)$	$O(tN)$
LinSVME	$O(\log(1/\epsilon)\ell_i)$	$O(N)$	$O(tN)$	-
AllCascade	$O(m^3)$	$O(Nm)$	$O(mt)$	-
RandBag	$O((km/N)^3)$	$O(km)$	$O(kmt/N)$	$O(vt)$
CEMPaR	$O((m/g)^3)$	$O((r+1)m)$	$O(mt/g)$	$O(gt)$
PACE	$O(\log(1/\epsilon)\ell_i)$	$O(Nc + N)$	$O((cN)^{1/c'^2 + o(1)})$	-

Next, a brief overview of the communication cost incurred by each peer is provided. After the model construction, clustering and cluster validation, each peer will propagate its model, centroids and centroids' accuracy to all other peers. There are c centroids for each peer, which are all n -dimensional vectors. The model for a two class problem is in n -dimensional space. Including the centroids' accuracy, then the communication cost will be $O(Ncd)$ bytes. Since all models are available at every peer, the prediction phase does not require any communication.

As a summary, the computation and communication cost of the relevant approaches are provided in Table 3.2 and the corresponding description of symbols in Table 3.1. For simplicity, all notation of SVM will represent non-linear SVM while LinSVM will represent linear SVM, hence SVME and LinSVME represents SVM Ensemble and linear SVM ensemble respectively. Note that for LinSVME and PACE, due to the small model size, it is possible for these algorithms to be used as either model propagation or test instance propagation approaches. Hence, Table 3.2 presents both the communication cost for training and prediction but note that only one cost is incurred.

Table 3.3: Summary of the datasets.

Dataset	Instances	Attributes	Classes	Peers (N)	Validation
Binary Census Income	295,173	40	2	200	test dataset
Binary Covertypes	581,012	54	2	500	10-fold CV
Multiclass Covertypes	581,012	54	7	500	10-fold CV
Multiclass KDD cup	1,074,992	42	14	1000	10-fold CV
Multiclass MNIST	70,000	780	10	100	test dataset
Binary SCDS	1,000,000	32	2	900	10-fold CV
Multiclass Waveform	110000	21	3	100	10-fold CV

3.5 Experiments

To validate the effectiveness of the proposed approaches (AllCascade, RandBag, CEM-PaR, PACE), extensive empirical studies are conducted. Specifically, the empirical studies show that the proposed approaches

- (i) are able to achieve classification accuracy comparable to existing centralized and P2P approaches,
- (ii) are able to scale well even with a large number of peers, and
- (iii) are invariant to varying data size and class distribution.

3.5.1 Experimental Setup

The experiments were conducted on some large sized datasets to simulate real world P2P environments as follows: binary Income Census (KDD), binary Covertypes, multiclass Covertypes, multiclass KDD cup 1999 and multiclass Waveform (Synthetic) datasets from the UCI machine learning repository [BL13] and the Synthetic Classification Data Set Generator (SCDS) [Mel97].

The binary Income Census (KDD) consists of weighted census data from 1994 and 1995 population surveys conducted by the U.S. Census Bureau. The provided census dataset comes with two subsets, training and testing datasets. Training was performed on the training set while all testing was performed on the test set. The multiclass Covertypes dataset consists of cartographic data for predicting the forest cover type. The binary Covertypes dataset is created from the multiclass data by merging data from all classes except class 2 into a single class (class 2 against all other classes). The multiclass KDD cup 1999 dataset was used for the Third International Knowledge Discovery and Data Mining Tools Competition, consisting of networking data for detecting network intrusions. The multiclass Waveform dataset is generated using the Waveform Database Generator (Version 1) with 100,000 instances. Due to some issues with the random number generator, the actual classification accuracy reported here is different from a previous work [AGHN08]. However, the differences do not influence the presented

analysis. SCDS was used to generate a binary class dataset, with 32 attributes, 4 of which are relevant. In addition, it was also used to generate a 6-class dataset, with 32 attributes, 10 of which are relevant. For both SCDS datasets, noises were simulated by wrongly assigning 20 percent of the attribute values and 20 percent of class labels. The number of peers for each dataset was chosen such that the local training data of each peer will not be too small (around 1000 instances). Except for the multiclass Income Census dataset where the test data was provided, the experiments for all other datasets were conducted using 10-fold cross validation (CV). All datasets were scaled to between 0 and 1, and duplicate instances were removed. A summary of the datasets is provided in Table 3.3.

In the experiments, the proposed approaches, AllCascade, RandBag, CEMPaR and PACE, are compared with several other algorithms, viz, centralized RSVM, centralized Linear SVM, Ensemble of non-linear SVM (SVME) with weighted majority voting and Ensemble of Linear SVM (LinSVME) with weighted majority voting and P2P Ivotes [LXLS07]. Centralized RSVM and Linear SVM are used as the benchmark for accuracy achievable in a centralized environment. AllCascade, RandBag and CEMPaR which are coded in C++ and the Reduced SVM (RSVM) used is based on [LL03] implementation (least square SVM method) and the SVM used for cascading is based on LIBSVM [CL11, LL03] implementation. For all non-linear SVM construction, RBF kernel was used and the γ and C values were chosen using the model selection tool (on 1% of the training data) provided with LIBSVM [CL11, LL03]. LIBLINEAR [HCL⁺08], a linear SVM package is used as the base classifier for PACE and LinSVME which is coded in C++. In addition, Kmeans++ [AV07] is used as the clustering algorithm for PACE. P2P Ivotes was implemented in Java and uses the J48 classifier from WEKA [WF05] as the base classifier. Unless otherwise mentioned, the default value for the subset size for RSVM s is 1%, cascading models k and the number of voting peers v for RandBag is $0.1N$ and number of groups g for CEMPaR is 10. Default settings for the Linear SVM were used, and for P2P Ivotes, the bite size was set at 400 for the MNIST dataset and 800 for the rest of the other datasets and error threshold was set at 0.002.

All experiments were performed on a cluster of 16 machines, each with two Intel Dual Core Xeon 3.0GHz processors, 4-GB RAM and connected by a gigabit Ethernet.

3.5.2 Classification Accuracy

Here, a comprehensive comparison of classification accuracy is presented in Table 3.4. In these experiments, every dataset is equally distributed among all peers while the assigned class distribution is random. Comparing with centralized RSVM and SVM ensemble, the classification accuracy of AllCascade is always within 1% difference if it is not performing better. In addition, AllCascade is the solution with the highest number of best accuracies, 3 out of 7 of the datasets (Multiclass Covertypes, Multiclass KDDCup

Table 3.4: Classification accuracy, with equally partitioned data and random class distribution.

Approaches	Binary Census	Binary Coverttype	Multiclass Coverttype	Multiclass KDDCup	Multiclass MNIST	Binary SCDS	Multiclass Waveform
Centralized RSVM	94.13	75.61	75.18	99.28	96.97	91.28	86.60
Centralized LSVM	94.60	75.68	71.28	99.34	79.82	90.62	86.69
SVME	94.65	81.36	76.89	99.35	93.40	92.01	86.38
LinSVME	94.32	75.61	70.83	99.21	86.65	92.03	86.67
P2P Ivotes	94.58	78.33	73.38	98.57	89.42	95.60	85.31
AllCascade	94.22	80.35	79.35	99.83	97.85	91.85	86.27
RandBag $k = 0.1N$	93.92	77.80	76.87	99.52	95.40	91.82	83.88
RandBag $k = 0.5N$	94.29	80.07	79.04	99.80	97.36	91.96	86.10
CEMPaR	93.71	76.73	76.52	99.57	92.36	91.62	83.49
PACE $v = 10$	94.31	75.51	70.67	99.32	82.43	91.71	86.55
PACE $v = 0.1N$	94.32	75.61	70.80	99.24	82.43	92.03	86.55

and Multiclass MNIST), followed by SVM ensemble then Linear SVM ensemble and P2P Ivotes. As for the RandBag approach, with $k = 0.1N$, the difference in accuracy with the best approach is up to 4%. However, with $k = 0.5N$, the difference in accuracy with the best approach is less than 1%. Note that P2P Ivotes achieved the highest accuracy for Binary SCDS with a large gap from all other solutions. This could be caused by the how SCDS generates data, resulting in positive bias for the decision tree approaches. Here, the proposed approaches have been demonstrated to achieve accuracy comparable to existing approaches (less than 4% difference in the worst case and on average less than 1%).

3.5.3 Computational Cost for Cascading Models

To determine how the number of models cascaded affects the training time, Figure 3.7 presents the time taken to cascade different number of models for four datasets, viz., binary Coverttype, multiclass Coverttype, Binary KDD cup and Binary SCDS. Although the range of values of x-axis in Figure 3.7 corresponds to the number of models cascaded k for RandBag, the result can be generalized to present the computational cost for All-Cascade where k is mapped to N . It is observed that for all datasets, the computational cost increases in a quadratic manner, which corresponds to the time complexity of the training phase. Observe that the time required for the multiclass dataset is relatively higher than that of the corresponding binary dataset (Coverttype). This is because SVM can only solve binary class problems hence multiclass problems have to be decomposed to several binary problems resulting in addition classifiers being built. Note that the time required for the models to be cascaded is quite low even with a large number of peers (500 for the multiclass KDD cup dataset). Hence, the proposed approaches are scalable in terms of computational cost, even for a large number of peers.

3.5.4 Communication Cost

To determine how the number of peers affects the communication cost of the proposed approaches, Figure 3.8 presents the communication cost of model propagation and prediction in terms of the number of instances sent which is the main contributor of the

Figure 3.7: Effects of number of models cascaded on average training time.

communication cost (based on Binary SCDS). From Figure 3.8.a, it is observed that AllCascade incurs the highest cost (an order of magnitude higher than Randbag), followed by RandBag (slightly higher than CEMPaR with $r = 6$ and an order of magnitude higher than CEMPaR with $r = 0$), then CEMPaR (slightly higher than PACE), PACE and LinSVME. Note that although LinSVME incurs very low communication cost, it is not robust and cannot adapt to varying data distributions. SVM ensemble is not shown in Figure 3.8.a as it incurs zero communication cost during the training phase. For the prediction phase, it is observed in Figure 3.8.b that the communication cost incurred by RandBag with $v = 0.1N$ is an order of magnitude lower than SVM ensemble. Note that AllCascade, PACE and LinSVME is not presented in Figure 3.8.b as they do not incur any communication cost during prediction. Note that by adjusting k and v for RandBag, r for CEMPaR and c for PACE allow users to control the communication cost to suit their use.

3.5.5 Data Size Distribution

This experiment determines if the distribution of size of peers' local training data affects classification accuracy using the multiclass Covertypes dataset. The size of local training dataset assigned to peers follows different distributions, viz., equal, exponential, normal

Figure 3.8: Effects of number of peers on communication cost.

 Table 3.5: Effects of different local training data size $|D_p|$ distribution on accuracy (Multiclass Covertypes).

Approaches	Equal	Exponential	Normal	Uniform
SVME	81.36	81.25	81.38	81.36
LinSVME	75.61	75.61	75.62	75.62
P2P Ivotes	78.33	78.26	78.28	78.31
AllCascade	80.35	80.47	80.27	80.17
RandBag $k = v = 0.1N$	77.80	77.87	78.01	77.90
CEMPaR	76.73	76.93	76.89	76.59
PACE $k = 10$	75.51	75.39	75.59	75.54

and uniform. Assignment of class distribution to each peer is random. Care has been taken to ensure each peer has sufficient training data. The results of the experiment are presented in Table 3.5. Among all approaches, RandBag for $k = v = 0.1N$ has the smallest variance in accuracy. Another observation is that for SVM ensemble, with non-equal distribution (more common for real environments), classification accuracy achieved is slightly lower. This experiment demonstrates that the proposed approaches are invariant to the distribution of training data size.

3.5.6 Data Class Distribution

To determine how the class distribution of data affects the proposed approaches, two experiments are conducted. First, for binary datasets (binary Covertypes and binary SCDS), every peer is assigned an equal sized local training dataset, and varied the class distribution assigned by introducing additional skew in the natural class distribution. Specifically, given a dataset with natural class distribution 40/60, an additional 5% of skew introduced would result in a set of peers' training data with class distribution of 35/65 and another set of peers' training data with class distribution of 45/55. This experiment ascertains if the P2P classification approach is affected by unequal class distribution among peers and the results are presented in Figure 3.9. Although for binary

Figure 3.9: Effects of imbalance class distribution on accuracy

Table 3.6: Effects of independent class distribution on accuracy (Multiclass datasets).

Approaches	Covertypes			KDD cup		
	Random	Indep	Diff	Random	Indep	Diff
SVME	76.86	69.66	7.20	99.35	98.13	1.22
LinSVME	70.83	65.99	4.84	99.21	98.34	0.87
P2P Ivotes	73.38	67.35	6.03	98.57	97.79	0.78
AllCascade	79.35	75.81	3.54	99.83	99.73	0.10
RandBag	76.87	73.80	3.07	99.52	98.99	0.53
CEMPaR	76.52	76.73	-0.21	99.57	99.51	0.06
PACE	70.80	69.81	0.99	99.32	99.17	0.15

Covertypes dataset, all approaches are unaffected by the additional skew introduced, this is not the case for the binary SCDS dataset. For the binary SCDS dataset, both of the proposed approaches are unaffected by the additional skew introduced but for the SVM ensemble, classification accuracy decreases as the additional skew introduced increase. This demonstrates that the proposed approaches are invariant to the additional skew introduced.

To further study the effect of class distribution on the proposed approaches, another experiment is conducted where given a multiclass dataset (i.e., Covertypes and KDD cup), peers are now assigned with equal amount of data but only contain a subset of the classes (independent class distribution). For example, given a 4 class dataset, every peer will only be assigned with training data from 2 out of the 4 classes. Table 3.6 presents the results of this experiment where the *random* and *indep* columns represent the random class distribution and independent class distribution results respectively and the *diff* column represents the difference between the random and indep results. In general, classification accuracy achieved by all approaches drops when peers' training data are assigned based on independent class distribution. However, it is observed that SVM ensemble has the largest decrease in accuracy followed by P2P Ivotes and the proposed approaches have much lesser decrease in accuracy.

3.6 Summary

This chapter studied the classification problem in P2P networks and presented four different approaches, viz., AllCascade, RandBag, CEMPaR and PACE. These approaches cover the problem of learning in a simplified P2P environment, possessing desired properties such as 1) anytimeness, 2) asynchronism, 3) decentralization, 4) tolerance to peer dynamism, 5) scalability and 6) invariance to imbalanced data distributions. In addition, both the structured and unstructured P2P network are covered. The structured P2P solution, CEMPaR, was able to significantly reduce the cost of model propagation for SVM cascading. Comprehensive experiments were conducted to evaluate both the efficiency and accuracy performance. Experimental results demonstrate that the proposed approaches are scalable, achieve satisfactory accuracy without incurring high communication cost and invariant to the size and class distribution of data. However, the proposed approaches are not suitable for all types of P2P classification tasks; i.e., AllCascade are more suitable for environments where changes or updates to the training data is infrequent and prediction frequency is high and for RandBag, adjustment to the parameters are needed to adapt different situations with consideration to the communication cost and classification accuracy. In addition, cascading approaches will be preferred for applications that require higher classification accuracy at the expense of much higher communication cost. Finally, AllCascade, RandBag and PACE could be deployed in either structured or unstructured P2P networks but an efficient communication protocol should be used with unstructured P2P networks to minimize communication cost. As for CEMPaR, note that it can only be deployed in structured P2P networks.

Chapter 4

Classification in P2P Networks with Asynchronous Concept Drifts

This chapter presents two approaches that address that problem of classification in P2P networks with asynchronous concept drifts. To address the limitations of existing work discussed in Chapter 2.2.7, the use of peers' concept drifts information to pro-actively predict concept drifts for speeding up concept drift adaptation has been explored. In addition, the use of ensemble based approaches to personalize peers' models to allow them to freely adapt to their own concept drifts without adversely affecting others have also been experimented with.

The following sections will first present the Reactive and Proactive Concept Drift detection Ensemble (RePCoDE) Framework followed by the Predictive and Parameter Insensitive Ensemble (PINE) Framework. Finally, the efficacy of the proposed approach will be empirically demonstrated.

4.1 Reactive and Proactive Concept Drift detection Ensemble (RePCoDE) Framework

4.1.1 Overview

This section proposes a *Reactive and Proactive Concept Drift detection Ensemble* (RePCoDE) framework for learning drifting concepts in P2P networks.

First, some basic assumptions are stated for learning with concept drifts in a P2P environment. It is assumed that the data of all peers are drawn from the same unknown underlying probability distribution where the concept changes from time to time. However, concepts may not always drift instantaneously in all peers (some delays can occur). Also, it is assumed that for all peers, data arrive sequentially, but are grouped together and processed in chunks, each consisting of n data instances. In addition, the time taken to gather each chunk is assumed to be a single time period or step. Figure 4.1 gives an

Figure 4.1: Flowchart of the RePCoDE Framework.

overview of the processes in the proposed solution. The idea of the proposed approach is briefly described below.

First, each peer monitors its own data stream and builds a classification model for every chunk of incoming labelled data. Next, it computes statistics of the data and uses them for indexing the corresponding classifier. If concept drift has occurred since last propagation or propagation waiting time reaches zero, one then propagates its current model together with data statistics to other peers.

Further, upon receiving the models and data statistics from other peers, each peer indexes the models using the corresponding data statistics. Contrary to most existing studies, the proposed approach requires the propagation of data statistics in order to achieve the *proactive* adaptation. Due to the possibly large number of models indexed, only the most relevant models are chosen and weighed according to their performance on the most current data chunk. In addition, the local data stream with that of other peers are also matched to select models that might better represent future data.

Finally, the class labels of unseen incoming data are predicted using these selected models based on the weighted majority voting. The pseudo code of the proposed algorithm is provided in Algorithm 4.1. The remainder of this section first presents detailed descriptions of the main phases of the proposed framework; viz., (1) training phase and (2) prediction phase, and then analyses time complexity and communication cost of the proposed approach.

4.1.2 Training Phase

In a continuous manner, each peer independently gathers the data and corresponding labels until the size of the specified chunk n is reached. Each peer then constructs a local

Algorithm 4.1: RePCoDE Framework.

```

1 Last propagated model  $M_L = \emptyset$ ;
2 current data chunk  $\mathbf{D}_t = \emptyset$ ;
3 current model  $M_t = \emptyset$ ;
4 current data statistics  $S_t = \emptyset$ ;
5 propagation delay count  $P_{delay} = w$ ;
6 while stream not end do
7   if new data chunk  $\mathbf{D}_t$  arrives then
8      $M_t =$  construct classification model based on  $\mathbf{D}_t$ ;
9      $S_t =$  compute statistics on  $\mathbf{D}_t$ ;
10     $M_L =$  check propagation conditions using Algorithm 4.2;
11  if new remote peer's model  $M_r$  and statistics  $S_r$  arrives then
12    Index model  $M_r$  using  $S_r$ ;
13  if new test dataset  $\mathbf{D}_{test}$  arrives then
14    predicted class labels  $\mathbf{y} =$  predict class labels of  $\mathbf{D}_{test}$  using Algorithm 4.3;

```

classification model based on the latest data chunk. As RePCoDE is an ensemble based approach, it is possible to use any type of classification algorithms such as decision tree, neural networks, Support Vector Machine, etc, for the model construction. However, due to the high frequency and short gap of data arrival, the algorithm used must have very low time complexity. In addition, as the models may need to be propagated at a later stage, there is also a need to consider the size of the resultant classification model. Hence, to meet the above two criteria, the state-of-the-art linear SVM classification algorithm [HCL⁺08] is chosen in the experiments. A linear SVM model, consisting of only a single data vector, can be built in linear time complexity.

In addition to model construction, *statistics* of the data chunks are also computed. These statistics are concise representation of the data chunks, which are computed for the following reasons: (1) to facilitate the searching of models built using *similar* data, and (2) to reduce the communication cost required for propagation together with their respective models. Although there are many possible methods (such as Gaussian Mixture Model, clustering, etc.) to compute the statistics, RePCoDE simply considers the linear SVM model which represents the decision hyperplane due to several reasons: (1) to streamline the model construction and statistics computation process, and (2) its low time and space complexity, and (3) its good ability of representing the training data in a separating manner.

Using the data statistics, the local models are then indexed locally. The index method used here has to be distance-aware, such as locality sensitive hashing (LSH) [DWJ⁺08]. The purpose is to speed up the process of model retrieval and filtering. As the number of models can be infinite, RePCoDE limits the size of the index m to control the space complexity of the framework. RePCoDE adopts a practical and computationally efficient replacement strategy for handling index overflow—to discard the oldest models, i.e., the newest model replaces the oldest in the index. For efficiency and suitability, LSH [DWJ⁺08] is adopted for indexing in the implementation.

Algorithm 4.2: Check propagation conditions.

input : Current data chunk $\mathbf{D}_t$, Current model M_t , data statistics S_t , last propagated model M_L , propagation delay count P_{delay} , max propagation delay w , drift detection threshold T ;

output: Last propagated model M_L , propagation delay count P_{delay} ;

```

1 if  $P_{delay} > 0$  then
2   | predicted class labels  $y_{current} = \text{predict } \mathbf{D}_t \text{ using } M_t$ ;
3   | predicted class labels  $y_{last} = \text{predict } \mathbf{D}_t \text{ using } M_L$ ;
4   | if percentage of differences between  $y_{last}$  and  $y_{current} > T$  then
5   |   | propagate  $M_t$  and  $S_t$ ;
6   |   |  $P_{delay} = w$ ;
7   | else  $P_{delay} = P_{delay} - 1$ 
8 else
9   | propagate  $M_t$  and  $S_t$ ;
10  |  $P_{delay} = w$ ;

```

In addition to the indexing of local models, propagation conditions are also validated. Here, two criteria for propagation are proposed: (1) presence of concept drift, and (2) propagation waiting delay. The first criterion uses the detection of concept drift to decide whether the local models should be propagated to other peers. Here, concept drift is examined by measuring the differences between the prediction outputs of the last propagated model and the latest model (based on the latest data), or the differences in their classification accuracies. For instance, let $y_{last} = \{0, 1, 2, 3\}$ be the predicted labels of the last propagated models and $y_{current} = \{0, 0, 0, 0\}$ be the outputs of the current models. In this case, the differences between the two models are 3 and the percentage of differences is 0.75. However, if the criterion is based on accuracy, then the percentage of differences will be simply the differences in accuracy. Given a concept drift threshold $T \in [0, 1]$, the latest model and its data statistics are propagated to other peers if the percentage of difference is greater than T . By comparing with the last propagated model, RePCoDE can detect both the sudden and gradual concept drifts. In the presence of gradual concept drifts, the difference in accuracy will slowly build up as more models are built and will eventually trigger the model propagation. Needless to say, for sudden concept drifts, the difference in accuracy will immediately satisfy the propagation criteria. Moreover, as a backup plan to insure against failure of concept drift detection, RePCoDE imposes a propagation waiting delay condition. By assuming the absence of any model propagation for w time steps, the latest model and statistics are propagated to the network, regardless of whether concept drift has been detected or not. The propagation waiting delay condition ensures regular updates of peers' data trends and provides additional models to improve accuracy performance of the ensemble solution. An outline is provided in Algorithm 4.2.

As for the remote models, upon receiving the models and statistics from other peers, each peer indexes the remote models using the corresponding data statistics in the same manner similar to the processing of local models.

Algorithm 4.3: Prediction.

input : test dataset $\mathbf{D}_{test}$, number of nearest neighbour voters k , number of proactive voters f , length of past statistics sequence b , current data chunk $\mathbf{D}_t$, current data statistics S_t , proactive/reactive ratio λ ;
output: Predicted Labels $\mathbf{y}$;
 1 Initialize model set $\mathbf{M}_{set}$, reactive weights $W_{reactive}$, reactive model set $M_{reactive}$, proactive weights $W_{proactive}$, proactive model set $M_{proactive}$, local past data statistics sequence $\mathbf{S}_{local}$ and similarity score W_{sim} to $\emptyset$;
 2 $\mathbf{M}_{set} =$ retrieve from index $\max(k, 2f)$ models nearest to S_t , in ascending order;
 3 $\mathbf{M}_{reactive}, W_{reactive} =$ find top k most accurate models and their weights from $\mathbf{M}_{set}$ based on $\mathbf{D}_t$;
 4 $\mathbf{S}_{local} = \{S_{t-b}, S_{t-b+1}, \dots, S_t\}$, the b past data statistics of local peer;
 5 **for** $i = 1$ **to** $2f$ **do**
 6 $\mathbf{S}_{temp} =$ ordered set of data statistics corresponding to b past models received from owner of $\mathbf{M}_{set}[i]$;
 7 $W_{sim}[i] =$ compute sequence similarity between $\mathbf{S}_{local}$ and $\mathbf{S}_{temp}$;
 8 $M_{proactive}, W_{proactive} =$ find top f most similar models and their weights from $\mathbf{M}_{set}, W_{sim}$;
 9 $\mathbf{y} =$ predict labels of $\mathbf{D}_{test}$ using models in $M_{reactive}$ and $M_{proactive}$ based on weighted majority voting ($W_{reactive}, W_{proactive}$) with ratio λ ;

4.1.3 Prediction Phase

As mentioned earlier, the proposed framework adapts to concept drift using both reactive and proactive techniques. Hence, the models used for prediction are selected based on two criteria: (1) distance between the model's statistics and the statistics of the latest (local) data chunk (reactive) and (2) similarity between the sequence of statistics of the models and the sequence of statistics of the local peer's data, i.e., pattern matching of concept drift trends (proactive). This process is outlined in Algorithm 4.3.

First, a peer retrieves the statistics of the latest data. Then the statistics are used to retrieve $\max(k, f*2)$ models from the index, where k is the number of *nearest-neighbour* voters and f is the number of *proactive* voters. The models retrieved have statistics that are the nearest (out of all indexed models) in terms of Euclidean distance to the statistics of the latest data chunk. Out of all the retrieved models, we select only the top k models for the reactive prediction component. Here, the basic assumption is that if the models have statistics similar to the most current data, they are more likely to be trained on similar data and hence, achieve better accuracy for the most current data. In addition, the accuracy of each of the chosen models is validated using the most current labelled data chunk, which is then set as the weights of the model for weighted majority voting. Finally, the sum of all weights of selected models is normalized to 1 such that the weights of all models becomes a distribution. Note here that this approach is similar to previous works [KM07, TPCP08], which is a reactive technique as it is only based on what has happened; i.e., based on the latest labelled data chunk which is already observed. Hence, adaptation to the concept drift only starts after the drift is known or detected. However,

the sharing of the different peers' models will allow the ensemble to achieve much better adaptation to concept drifts compared to only using local models. An ensemble solution also performs better when more (relevant, generalization error less than 50%) models are used.

It is obvious that using the reactive approach, there is still an initial drop in accuracy when concept drift first starts. However, if it is assumed that the same concept drift occurs at different time steps for different peers, then it may be possible to learn from the concept drift patterns of peers' whose concept drift has already occurred. Hence, the following proactive technique is proposed to select models that may be representative of the future data. First, backtrack b time steps and retrieve the statistics of the data up until the most current time step. This is defined as a sequence of the local data statistics. Next, for each model retrieved that is not constructed locally, check the existence of a *later* model received from the same peer. Then, in a similar manner, backtrack to b *earlier* models received from the same peer and create a sequence of remote data statistics. Then, compute the similarity of the two sequences to obtain a similarity score. In this implementation, dynamic time warping (DTW) algorithm is used to perform similarity matching [Lem09]. Once the similarity matching has been performed for all models retrieved from the index, select f *later* models whose sequence of statistics is the most similar to the local sequence. These later models are termed as *proactive* models as they are deemed to be indicative of future data. Here, assume that peers experience concept drifts in the same patterns and the sequence similarity matching searches for peers who had similar concept drifts experience. The similarity score obtained is used to weigh the importance of the *proactive* models. Similar to the reactive approach, the sum of the weights of all proactive models are also normalized such that the weights of all models becomes a distribution.

Finally, to fuse the the reactive and proactive approaches, RePCoDE allows users to set a ratio parameter $\lambda \in [0, 1]$ that determines the importance of the reactive and proactive approaches. With $\lambda = 0$, only the reactive approach is used, and on the contrary, with $\lambda = 1$, only the proactive approach is used. As such, the class labels of unlabelled data are obtained by performing weighted majority voting based on both the selected reactive and proactive models, where the balance between the two approach is determined by λ .

4.1.4 Complexity Analysis

For the model construction phase, the time complexity is analysed with respect to only a single data chunk since data are possibly infinite. For each model construction (Linear SVM), the time complexity is $O(\log(1/\epsilon)nd)$ for an ϵ -accurate solution where n is the size of the data chunk [HCL⁺08]. For computation of data statistics, no additional cost is incurred. For indexing the model based on LSH, suppose there are L hash tables and

τ is the cost of computing one function, then the cost is $O(L\tau)$ for each model. Hence, the total cost for model construction is $O(\log(1/\epsilon)nd + L\tau)$.

The time complexity to predict a dataset D_{test} of size n_t is as follows (c.f., Algorithm 4.3). First, retrieve $\max(k, 2f)$ models from the LSH index, which cost $O(dm^{1/c^2})$, where m is the size of the index and c is the approximation factor. To evaluate the accuracy of the models, the cost is $O(\max(k, 2f)nd)$. Next, the past data statistics are retrieved and sequence similarity computed, which cost $O(2fb)$ and $O(2fdb^2)$ respectively. The cost of retrieving the top k and f models is $O(k)$ and $O(f)$ respectively. Finally, the cost of predicting with $k + f$ models is $O((k + f)dn_t)$. Hence, the total cost for prediction is $O(dm^{1/c^2} + \max(k, 2f)nd + 2fb + fdb^2 + k + f + (k + f)dn_t)$.

4.1.5 Communication Cost

The only communication cost incurred is the cost of propagating the data statistics and models. Hence, the cost of propagation for each model (including its data statistics) is $O(d)$, as the Linear SVM model consist of only a single vector and since the Linear SVM model is used as the data statistics to represent the data, no additional cost is incurred. However, note that factors such as concept drift detection threshold and propagation waiting delay can affect the frequency of model propagation and hence increase the communication cost.

4.2 Predictive and Parameter Insensitive Ensemble (PINE) Framework

This section proposes a novel framework called *Predictive and parameter INsensitive Ensemble* (PINE) for handling asynchronous drifting concepts in distributed networks.

This section starts with a brief overview of the proposed framework followed by detailed discussions on each component of the proposed framework.

4.2.1 Overview

The sequence diagram of the proposed framework from one peer p perspective is presented in Figure 4.2. The peer p in PINE monitors its own data stream $\mathbf{D}_p$, and maintains its own ensemble of reactive predictive models denoted as $\mathbf{REM}_p$ and an ensemble of proactive predictive models denoted as $\mathbf{PEM}_p$. The reactive models are the classifiers that represent peer p 's current data distribution, and the proactive models are the classifiers that may represent peer p 's future data distribution which is different from the current distribution. Whenever a new unseen data instance $\mathbf{x}_{new}$ arrives, peer p consults both $\mathbf{REM}_p$ and $\mathbf{PEM}_p$ and from a combination of their outputs predicts the class label $\hat{y}_{new}$ (cf. Section 4.2.7).

The idea and concepts of the proactive and reactive models is illustrated in Figures 4.3 and 4.4 respectively, each of which follows the example shown in Figure 1.2.

Figure 4.2: The proposed framework as a sequence diagram.

Figures 4.3 and 4.4 illustrate the change in concepts of peers (change in y axis value), the concept drift detection and adaptation (vertical dotted line) and propagation of the newly trained models to other remote peers (labelled arrows). For instance, peer p_1 's concept drifted at time t_3 (drop in y axis value), was detected and adapted at time t_4 (vertical dotted line).

In addition, Figure 4.3 shows the optimal model that should be used before a peer's concept drift is detected and Figure 4.4 shows the optimal model(s) that should be used after a peer's concept drift is detected, which is indicated in both figures by the labels in the shaded areas. The example for proactive model presented in Figure 4.3 is as follows. Peer p_1 's concept has drifted at time t_3 and was detected at time t_4 . Hence, between time t_3 and t_4 , peer p_1 's current model is not suitable for predicting its own data since the model is made outdated by the concept drift. Similarly, peer p_3 is also not suitable. As such, from time t_3 to t_4 , peer p_2 's model should be the most accurate (optimal) for predicting p_1 's unlabelled data.

Similarly, the example for reactive model presented in Figure 4.3 is as follows. Peer p_1 concept drift was detected in time t_4 and hence, from time t_4 to t_5 , the best models for predicting peer p_1 unlabelled data are the current models of peers p_1 and p_2 . At time t_5 , peer p_3 detected its concept drift and updated its model which it propagated

Figure 4.3: An illustration of optimal *proactive* concept drifts for two peers. The shaded areas indicate non detected concept drift periods and the labels m_i indicate the optimal proactive models, where i is the peer number from Figure 1.2.

Figure 4.4: An illustration of optimal *reactive* concept drifts for two peers. The shaded areas and their labels indicate optimal reactive models.

to peer p_1 . Hence, the optimal models for p_1 's unlabelled data are those of peers p_1, p_2 and p_3 . However, at time t_7 , peer p_2 adapted to a new concept which is different from that of peer p_1 . Hence, peer p_2 's current model at time t_7 is no longer suitable for peer p_1 and the optimal models for p_1 's unlabelled data are models of p_1 and p_3 .

Therefore, the aim of PINE is to ensure timely supply of the optimal models for each peer either locally or 'borrowed' from other peers (combined of Figures 4.3 and 4.4).

A walk-through of the various components in PINE illustrated in Figure 4.2 is as follows.

Methods 1 & 2. First, whenever an unlabelled data point $\mathbf{x}_{new}$ is received (*receiveNewData*) by peer p , its label y_{new} is predicted using the models in $\mathbf{PEM}_p$ and $\mathbf{REM}_p$.

Methods 3 to 7. Once the true label y_{new}^* of $\mathbf{x}_{new}$ arrives (*receiveDataLabel*), peer p then checks if has have been a concept drift for the received training exam-

ples (*checkConceptDrift*). If the concept has not drifted, p then updates its training dataset D_p that represents the current concept and updates the local classifier m_p (*updateClassifier*). Otherwise, if the concept has drifted, peer p then updates D_p by removing data of the old concept and moving them to $\hat{D}_p$ which represents the old concept. A new local classifier m_p is then built from D_p (*constructClassifier*). Next, peer p notifies all other peers r_i of the time its concept drifted and sends the new local classifier m_p to them (*notifyDriftPushClassifier*). The details of training local classifiers and handling concept drift are presented in Section 4.2.2.

Method 8. To exploit the proactive models from other peers, peer p mines the temporal association rules from the sequence of concept drift occurrences (*mineTempAssoc*), which are used for concept drift prediction. These rules consist of a sequence of peers whose concept drift occurrences were followed by the local concept drift at peer p (cf. Section 4.2.3).

Methods 9 & 10. Then peer p tests all remote classifiers m_{r_i} using $\mathbf{D}_p$ and $\hat{\mathbf{D}}_p$ to verify if there are accurate proactive classifiers (*verifyProactive*). If accurate, the remote classifiers are flagged as potential proactive models (cf. Section 4.2.4). In addition, it also verifies if the remote classifiers m_{r_i} are reactive classifiers (*verifyReactive*) and adds to $\mathbf{REM}_p$ if accurate (cf. Section 4.2.5). This marks the end of all tasks required to be performed after the local concept drift occurrence.

Methods 11 to 13. Next, whenever a remote peer r_i receives a notification of concept drift from peer p and its new classifier m_p , r_i then tests m_p using $\mathbf{D}_{r_i}$ to verify if it is a reactive classifier (*verifyReactive*) and adds a copy to its $\mathbf{REM}_{r_i}$ if accurate. In addition, peer p 's time of concept drift is recorded and checked against the existing temporal association rules to predict if local concept drift is likely to occur in peer r_i (*matchTempAssoc*). When a rule is fully matched, peers listed in the rule sequence are marked as potential proactive classifiers and time-stamped at time concept drift of r_i is predicted (cf. Section 4.2.6).

At the time point when a local concept drift at peer p is predicted by a temporal association rule (*updatePEM&REM*), remote classifiers m_{r_i} of peers listed in the rule are then added to $\mathbf{PEM}_p$ if they are valid proactive classifiers and removed from $\mathbf{REM}_p$ (cf. Section 4.2.6).

4.2.2 (Re-)Training Local Classifiers

This section presents details of how local classifiers are (re-)trained following arrival of every true label depending on whether a concept drift has been detected (**Methods 5 & 6**). The detection of concept drift will be discussed in the next section.

In principle, any classification algorithm can be employed at peer p . However, with consideration to the distributed network settings, the selected algorithm should have two desired properties – low time complexity to deal with the possible large amount of

data and low model propagation cost since the models need to be propagated to other peers. Hence, this study uses the state-of-the-art linear SVM classifier.

As new labelled instances arrive, the performance of a local classifier m_p is monitored for detecting concept drift. If no change is detected, then the classifier is incrementally updated with new labelled data. Otherwise, if a change is detected, the change detector points out the time when drift occurred. In such a case a new classifier is trained with the data accumulated after the occurrence of the drift. By doing so, it can be ensured that the prediction on new data will not be affected by the old concept, which will lower the prediction accuracy. This is a common practise among ensemble based solutions which deal with concept drifts [KM07, BHP⁺09, MWY10, SK01, BM08]. This classifier is sent to other remote peers that are likely to experience this drift in the (near) future.

Locally learnt base classifier at peer p can become part of REM_{r_i} at any remote peer r_i . In order to keep the communication costs low, peer p is allowed to propagate its updated classifier only when the sample size used for inducing the local base classifier increases in geometric progression as compared to the previous sample size, so that the improvements are substantial.

Detecting Concept Drift Here, the details of how concept drift detection is organized in the proposed framework is provided.

A change detector is a function that monitors the historical data and signals the occurrence of concept drift. A detector needs to see a number of instances that represent a new concept before it can detect the drift. The period from the time when the actual drift occurred to the time when it was detected is called the *detection lag*. A good change detector would detect drifts accurately with as few false positives and as small detection lag as possible. Figure 4.5 illustrates the setting of concept drift detection in the mentioned distributed classification example. A detection lag is shaded in grey. A change detector is needed to estimate the time of actual concept drift occurrence, which is the arrival time of the oldest data point in the new concept. This allows the timely deployment of the proactive models whenever concept changes as illustrated in Figure 4.3.

In the proposed framework, concept drift is detected (**Method 4**) for each peer using a modified version of the ADWIN algorithm [BG09], which has sound theoretical justifications and uses only a few parameters. Given a sequence of data points, ADWIN splits the sequence into all possible pairs of sequential windows and compares the means of data within the windows. When the absolute differences of the means are significant, the oldest element is dropped and the comparison repeats until no significant difference is found.

More formally, suppose μ_1 and μ_2 are the means of the two sub-sequences as a result of a split. Then the criterion for signalling a change is $|\mu_1 - \mu_2| > \epsilon_{cut}$, where

$$\epsilon_{cut} = \sqrt{\frac{1}{2\mu} \log \frac{4n}{\delta}}, \quad \mu = \frac{1}{\frac{1}{n_1} + \frac{1}{n_2}}, \quad (4.1)$$

Figure 4.5: An example lag in drift detection. $CDD(B)$ represents the detection of concept drift of concept B . Shaded area indicates the transition period where the concept drift has changed but has not yet been detected.

here n is the length of the full sequence, while n_1 and n_2 are lengths of the sub-sequences respectively. Note that $n = n_1 + n_2$, and $\delta \in (0, 1)$ is a hyper-parameter of the model.

In PINE's concept drift detection, ADWIN is used to detect over a stream of *accuracies* represented as ones for correct and zeros for wrong predictions). This is because classification accuracy is the main concern and evaluation criteria, besides, in some cases concept drifts may occur without any change in the distribution of the attributes (the real concept drift).

In addition, 2 modifications have been made to ADWIN and motivation behind these modifications are as follows. First, concept drift are considered to have occurred when the mean of the first window is *significantly larger* than that of the second window, i.e., $\mu_1 - \mu_2 > \epsilon_{cut}$.

Second, to speed up the detection, instead of dropping a single element from a window, *all elements* in the previous window are dropped.

Note that even though a concept drift is detected and old data may seem not necessary anymore, in PINE, every peer p maintains two sets of data $\mathbf{D}_p^{t-1}$ and $\mathbf{D}_p^t$ corresponding to the previous and the current concept. This is necessary for the proactive model verification (cf. Section 4.2.4). In the experiments, the size of the two datasets $\mathbf{D}_p^{t-1}$ and $\mathbf{D}_p^t$ are unbounded as it was not necessary to limit their sizes. The average size of $\mathbf{D}_p^{t-1}$ and $\mathbf{D}_p^t$ are less than 2000 over all the experiments that were conducted, which is not an unreasonable size to maintain. However, bounding of the dataset size should only have a negligible effect, and if the bounding window is large enough to represent the concept (e.g., 1000 to 10000), no adverse effect is expected from bounding the window size.

4.2.3 Mining Temporal Associations

This section describes how the temporal associations of concept drifts among peers are mined in PINE (**Method 8**).

The temporal associations of the concept drift occurrences among peers can allow peers to proactively predict occurrence of drifts in peers. Hence minimize the lag between

the drift and adaptation of the models. In order to obtain updated temporal associations, the temporal association rules are mined whenever a local concept drift is detected and the old rules mined in the previous concept drift are then forgotten and replaced with the new rules mined.

With reference to the time notation, an important point to note here is that while PINE requires all peers to share a global time, no global synchronization is required among peers. They can simply make use of their local system clocks with periodic synchronization to the Coordinated Universal Time (UTC) to achieve the intended purpose (with the correct offsets). Unless the time gap between concept drift occurrences are very close, otherwise some differences between the local clocks and UTC are acceptable and peers only need to synchronize with UTC once in a while.

There is a need to find not only temporal associations but also the *expected delays* between the occurrences of concept drift at different peers (that can be seen also as the propagation time of concept drift from one peer to another). This knowledge is mined from a sequence of global concept drift occurrence events. The setting for this mining is formally defined as follows [MTIV97].

Given a set of peers E , a concept drift event is a pair (A, t) where $A \in E$ corresponds to a peer whose concept has drifted at time t_i . A concept drift event sequence $\mathbf{s}$ is a triple (s, T_s, T_e) where $s = \langle (A_1, t_1), (A_2, t_2), \dots, (A_n, t_n) \rangle$ is an ordered list of concept drift events such that $t_i \leq t_{i+1}, \forall i \in \{1, \dots, n\}$ and T_s and T_e are the starting and ending time where $T_s \leq t_i \leq T_e, \forall i \in \{1, \dots, n\}$. Hence, the following sequence can be derived from Figure 1.2: $\mathbf{s}_{global} = (s_{global}, t1, t20)$ where $s_{global} = \langle (p_2, t1), (p_1, t3), (p_3, t4), (p_2, t6), (p_1, t8), (p_2, t9), (p_3, t10), (p_1, t11), (p_2, t15), (p_1, t17), (p_2, t18), (p_3, t19), (p_1, t20) \rangle$.

This problem is closely related to the frequent episode mining problem [HD04, LWC08, MTIV97]. Hence, from here on, the terms *episodes* and *temporal association rules* shall be used interchangeably. In general, an episode can be considered as a partially ordered collection of events occurring together [MTIV97]. Existing frequent episode mining algorithms aim to find all possible frequent episodes from the sequence. The scenario presented in this work is slightly different, as the peers are only interested in concept drift events leading to their own concept drifts. Hence, the problem is simplified by splitting the sequence into sub-sequences using the local concept drift event as the delimiter, i.e., setting $T_s + 1$ and T_e as time of two consecutive local drifts events. Table 4.1 shows the resultant sub-sequences by segmenting s_{global} where p_1 is the local peer, e.g., the set of sub-sequences is $\{(ss1, t1, t3), (ss2, t4, t8), (ss3, t9, t11), (ss4, t12, t17), (ss5, t18, t20)\}$.

In this setting, one can simply apply frequent sequence mining algorithms [Zak01] to find these frequent episodes α denoted as a pair $(V, \leq, g)$ where V is a set of nodes, $\leq$ is a partial order on V and $g : V \rightarrow E$ is a mapping associating each node with a peer. The interpretation of an episode is that the events in $g(V)$ have to occur in

Table 4.1: Peer p_1 's segmented sequences of the global sequence of concept drift occurrences.

Tid	Segmented sequences
ss1	$(p_2, t1)$
ss2	$(p_3, t4), (p_2, t6)$
ss3	$(p_2, t9), (p_3, t10)$
ss4	$(p_2, t15)$
ss5	$(p_2, t18), (p_3, t19)$

the order described by $\leq$. Formally, an episode $\alpha = (V, \leq)$ occurs in an event sequence $\mathbf{s} = (s, T_s, T_e)$, denoted as $\alpha \in \mathbf{s}$, if there exists an injective mapping $h : V \rightarrow \{1, \dots, n\}$ from nodes of α to events of $\mathbf{s}$ such that $g(i) = A_{h(i)}$ for all $i \in V$, and for all $i, j \in V$ with $i \neq j$ and $i \leq j$ then $t_{h(i)} < t_{h(j)}$. For ease of reading, the episodes in this work is presented as $e = A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n$ where $A_i \rightarrow A_j$ denotes concept drift event A_i occurs before A_j , e.g., ss2 in Table 4.1 can be represented as $p_3 \rightarrow p_2$.

When computing the *delay* from the occurrence of an episode until the local concept drift event, the interest is only in computing the delay from the smallest sub-sequence with the earliest ending time which the given episode occurs in (*earliest occurring sub-sequence*). This is because the episodes are matched in a state transition manner. Hence, given an episode α and a sequence $\mathbf{s} = (s, T_s, T_e)$, *earliest occurring sequence* is denoted as $\mathbf{s}_{earliest} = (s_{earliest}, T_s, t_e)$ where $t_e = \min_{t \leq T_e} \{\mathbf{s} = (s, T_s, t) | \alpha \in \mathbf{s}\}$. Therefore, the *delay* is computed as $t_l - t_e$ where t_l is the time the local concept drift event occurred.

For example given a subsequence $s = \langle (p_2, t1), (p_3, t2), (p_2, t4), (p_3, t5), (p_1, t6) \rangle$ and a frequent episode $e0 = p_2 \rightarrow p_3$ of peer p_1 , $\langle (p_2, t1), (p_3, t2) \rangle$ is the first occurrence of $e0$ and the delay of an episode $delay(e0)$ is computed from the last event of the first occurrence of the episode to the local concept drift event, e.g., $delay(e0) = t6 - t2$.

In this setting, based on the definition of serial episode [LWC08], all the mined frequent episodes will have hundred percent confidence, which is not useful at all. Hence, the standard definition is deviated to define confidence as the same value as support σ since it gives the probability a given episode leading to a local concept drift.

In addition, the following new measure is proposed: delay deviation d_{dev} which is the standard deviation of the expected delay leading to the local concept drifts given a matched frequent episode, indicating the confidence on the expected delay. A lower d_{dev} implies a higher confidence since the expected delay covers a smaller range and hence is more accurate.

For estimating the delay until the local concept drift event, the variable *expected delay* d_{med} is defined and computed as the median of the delays. Median is chosen to reduce the effects of outliers. Table 4.2 provides an illustration of the frequent episode, support, delays, expected delay d_{med} and delay deviation d_{dev} , with reference to Table 4.1.

Table 4.2: Frequent episodes leading to concept drifts of peer p_1 . For simplicity, the consequent p_1 is removed.

Rid	Freq. episodes	Delays	σ	d_{med}	d_{dev}
$e1$	p_2	2,2,2,2	1	2	0
$e2$	p_3	4,1,1	0.6	1	1.732
$e3$	$p_2 \rightarrow p_3$	1,1	0.4	1	0
$e3$	$p_3 \rightarrow p_2$	2	0.2	2	0

Note that the frequent sequence mining does not compute the support, expected delay d_{med} and delay deviation d_{dev} . Hence, post processing is required and the fast counting technique presented in [LSU07] is used to match the episodes and perform the statistics computations.

Finally, the filtering of the episodes, to reduce the number of episodes mined and prevent mining of irrelevant episodes, are as follows. Two input parameters, viz., min support σ_{min} and max delay deviation $d_{max-dev}$, are specified. The min support σ_{min} is used as the first cut filter when all episodes with support σ lower than σ_{min} are discarded. Then the max delay deviation $d_{max-dev}$ is used where rules with d_{dev} larger than $d_{max-dev}$ are discarded. These (unfiltered) frequent episodes, expected delay d_{med} and delay deviation d_{dev} are then used for the temporal association rule matching (cf. Section 4.2.6).

4.2.4 The Ensemble of Proactive Models (PEM)

This section provides the details of how the ensemble of proactive models **PEM** is managed and maintained (**Method 9**).

The classifier m_{r_i} of a remote peer r_i is considered as a proactive classifier if it is learnt at r_i after a concept drift has occurred at that peer and it is expected that this drift will propagate in a near future to p . Whenever peer p detects a local concept drift, the following condition is checked for the classifier m_{r_i} of every remote peer r_i :

$$Err(m_{r_i}, \mathbf{D}_p^{t-1}) - Err(m_{r_i}, \mathbf{D}_p^t) > \epsilon_{cut} \quad (4.2)$$

Since a significant difference have been found by the local classifier, then a proactive classifier of a remote peer r_i should also be able to do so and obtain a significantly higher error on the old concept compared to the new concept, thus satisfy (4.2). If m_{r_i} satisfies (4.2), peer r is flagged as a proactive peer and the accuracy $1 - Err(m_{r_i}, \mathbf{D}_p^t)$ of m_{r_i} is stored and used later if it is used as a proactive classifier.

4.2.5 The Ensemble of Reactive Models (REM)

This section details how the ensemble of reactive models **REM** is managed and maintained (**Methods 10 & 11**).

The ensemble of reactive models for each peer p consists of one locally learnt model (i.e. from data observed by peer p) and zero or more models which were learnt remotely at other peers and validated on recent data at p .

Every peer p stores the latest classifier m_{r_i} of every remote peers r_i . Whenever a new classifier m_{r_i} arrives, the following conditions (two of which are based on ADWIN's measure) are checked:

$$Err(m_{r_i}, \mathbf{D}_p^t) < 0.5 \quad (4.3)$$

$$Err(m_{r_i}, \mathbf{D}_p^t) - Err(m_p, \mathbf{D}_p^t) \leq \epsilon_{cut} \quad (4.4)$$

$$Err(m_{r_i}, \mathbf{D}_p^t) - Err(m_{r_i}, \mathbf{D}_p^{t-1}) \leq \epsilon_{cut} \quad (4.5)$$

where n_1 and n_2 of (4.1) are the size of the respective datasets. If all three conditions are satisfied, then m_{r_i} is a reactive model and it is added to the reactive ensemble and the preceding model removed, i.e., $\mathbf{REM}_p \leftarrow m_{r_i}$ and $\mathbf{REM}_p = \mathbf{REM}_p \setminus \hat{m}_{r_i}$, where $\hat{m}_{r_i}$ is the predecessor of m_{r_i} .

In addition to the criteria of adding a classifier to an adaptive ensemble when its error rate goes below a certain threshold, e.g. 0.5 in (4.3), two more conditions specific to the distributed settings are employed, which is expected to improve the accuracy of the ensemble. In (4.4), the error rate of m_{r_i} is verified to be not significantly higher than the local classifier m_p . By doing so, it is possible to ensure that m_{r_i} is built on data from the same concept as m_p . In (4.5), the error rate of m_{r_i} on the previous concept is verified to be not significantly higher than the current concept of peer p since this might indicate that m_{r_i} is built from previous concept data and its error rate might increase in the future.

Note that whenever local concept drift (LCD) is identified, every remote model m_{r_i} is verified and added to REM if it passes the validation. Recall Figure 4.4, which provides an illustration of situations where the reactive models need to be used.

4.2.6 Balancing Reactive and Proactive Ensembles

This section specifies the details of how the Reactive and the Proactive ensembles interact with each other and details the role of the temporal association rules in this process (**Methods 12 & 13**).

Whenever the local peer is notified of a peer's concept drift, the previously mined temporal association rules will be checked. A match of peer's concept drift will transit the rule to the next state. Note that the initial state of all rules is the first item. For instance, given a temporal association rule $e4 = p_4 \rightarrow p_2$, its initial state will be p_4 , meaning that it will wait for the occurrence of peer p_4 's concept drift. Given that now p_4 concept drift has occurred, then the state will be transited to p_2 .

When the entire rule is matched, all peers identified in the rule will be made into proactive candidates until d_{med} of the rule has passed (from the point the last peer's

has concept drifted). Note that if a new rule with lower d_{dev} is matched, the d_{med} of all peers in the new rule are replaced with the new d_{med} as it has a higher confidence hence taking precedence.

Whenever the delay of a proactive candidate r_i has passed, its current classifier will be added to the proactive ensemble, i.e., $\mathbf{PEM} \leftarrow m_{r_i}$, and removed from the reactive ensemble, i.e., $\mathbf{REM} = \mathbf{REM} \setminus m_{r_i}$.

At any point in time for a given peer, there is a need to balance between the reactive and proactive predictions depending on the expectations of concept drift. The process of adding models to $\mathbf{PEM}$ and removing models from $\mathbf{REM}$ is used as the mechanism for adaptively balancing the weights between reactive and proactive prediction. As more rules are matched, that indicates that more peers have experienced concept drift and it becomes more evident that the local concept of a given peer is likely to drift. As such, there is a need to increase the importance of the proactive prediction which is achieved by increasing the number of models in $\mathbf{PEM}$ or reducing the size of $\mathbf{REM}$. This simple approach is expected to be more accurate than assigning weights for the two ensembles. Finally, note that whenever local concept drift (LCD) is identified, all models in $\mathbf{PEM}$ are removed.

4.2.7 Prediction by Reactive and Proactive Ensemble Models

All the previous section of the PINE framework presentation detailed the process of on-line training and adaptation of the system. This final section of the framework describes how to get a prediction from a trained PINE for an unseen data point (**Method 2**).

When peer p receives a new unseen data $\mathbf{x}_{new}$, its class label y_{new} is predicted using all models from the reactive and proactive ensembles, i.e., $\forall i, m_i \in \mathbf{REM}_p \cup \mathbf{PEM}_p$, using a weighted voting approach. The weights of the models in $\mathbf{REM}_p$ and $\mathbf{PEM}_p$ are computed by different ways. In particular, the weights of model in $\mathbf{REM}_p$ are computed based on their error on the training data $\mathbf{D}_p$ of the current concept, i.e., $w_i = 1 - Err(m_i, \mathbf{D}_p), m_i \in \mathbf{REM}_p$. In contrast, for the models $m_i \in \mathbf{PEM}_p$, their weights are obtained during the proactive classifier verification or in another words it is estimated from the performance of the previous classifier on current concept, e.g., $w_i = 1 - Err(\hat{m}_i, \mathbf{D}_p)$, where $\hat{m}_i$ denotes the predecessor of m_i . Finally, the class label is predicted by the following function: $y_{new} = \text{sign}(\sum_i (w_i m_i(\mathbf{x}_{new})))$.

Next, a general overview and detailed design choices of the framework for distributed classification under asynchronous concept drifts is described. The framework makes use of associations between drifts in different peers happening over time to speed up the adaptation by predicting the occurrence of drifts. The next section empirically evaluates the performance of RePCoDE and PINE.

4.3 Experiments

Extensive experiments are conducted to evaluate the performance of the proposed approach. The performance of RePCoDE and PINE are compared with existing approaches that can handle drifting concepts in distributed network classification.

The main goals of the experimental study include:

- presenting the motivation and potential of proactive handling of concept drift in distributed settings, i.e. demonstrating that the changes in different peers can be related, and that corresponding temporal relations between peers (w.r.t. occurrence of drifts) can be mined effectively;
- demonstrating that it is possible to estimate the propagation time of concept drift from one peer to another accurately enough such that this estimate can be used for proactive handling of concept drift;
- showing that RePCoDE and PINE achieve better accuracy than the state-of-the-art approaches for handling concept drift in P2P (distributed) networks;
- demonstrating that RePCoDE and PINE incur comparable communication and computational costs with existing approaches.

4.3.1 Datasets

For the experiments, the following synthetic and real world dataset have been used.

Synthetic data. The *moving hyperplane* data generator [HSD01] is employed to act as a benchmark for evaluating classifiers adapting to concept drift. The hyperplane has been used to simulate both the (incremental) gradual and sudden drifts scenarios. This data model is chosen since it has several important properties for analysing concept drift. First, it does not change the prior probabilities of classes; second, the drift is easy to quantify by the angle of rotation; third, the hyperplane (or plane in 2D) is easy to interpret.

The moving hyperplane generator is as follows. A hyperplane in d -dimensional space is expressed by the equation $\sum_{i=1}^d w_i x_i = w_0$, where w_i is the weight of the attribute x_i . Data instances satisfying $\sum_{i=1}^d w_i x_i \geq w_0$ are labelled as positive and negative otherwise. Concept drifts are introduced by changing the weights of the attributes and the direction of change. The generated data consists of 8 attributes of which 2 are drifting. The probability of noise, which means assigning a wrong class label, is set at 0.05. The weight change per attribute is set to 0.5 and 5, the probability of direction change is set to 0.1 and 1 and concepts are set to drift every 10 and 500 instances, randomized by adding a small random value, for gradual and sudden drift scenarios respectively. Peers' initial concepts are the same and are set to drift to the next concept with different delays, randomly assigned in the range of $[0, 1000)$. 10 independent runs

are conducted with every run consisting of 100 peers. Results were obtained by averaging over all peers and runs. The total number of instances each peer receives for one run are 17,000 and 19,000 for the gradual and sudden drift scenarios respectively.

Real data. To verify if peers share correlated concepts (and concept drifts), experiments are conducted on *real world weather* data from SOD¹, which consists of daily observations recorded in different meteorological stations. The task is to predict whether it is going to rain from 10 weather observations, viz., mean temperature, mean dew point, mean sea level pressure, mean station pressure, mean visibility, mean wind speed, maximum sustained wind speed, maximum wind gust, maximum temperature and minimum temperature. 500 worldwide stations are selected, each representing a peer and observations from year 2001 to 2010 consisting of 3038 days of observations that exist in all selected stations.

4.3.2 Experimental setup

RePCoDE and PINE are compared with two other approaches, first of which can be considered as the baseline and the second as the main existing competitor: 1) *Local* – uses only the latest local adaptive model (based on the modified ADWIN) of a peer to perform prediction. It represents existing centralized solutions that do not share any information. 2) *All* – uses the latest adaptive model of all peers to perform prediction. It represents both ensemble approaches and approaches that assume a global drifting distribution.

The δ value of ADWIN for all adaptive detection approaches is set as 0.25. The default *min support* σ and *max delay deviation* d_{dev} is set at 0.25 and 100 respectively. Parameters of RePCoDE are set as follows: window size 100, proactive and reactive ratio 0.5, number of nearest neighbor voters 10% of peers, number of future voters 10% of peers, propagation delay 4, concept drift threshold 0.05 and number of sequences to match 4.

The approaches are tested in a prequential manner [GSaR09], i.e., the incoming data are first used for testing and then for updating classifiers. Results are computed as the moving average (100 data points) of averaged prequential error rate of all peers.

For implementation, all algorithms were implemented in C++. LIBLINEAR [HCL⁺08] package for linear SVM is adopted as the base classifier for all approaches. Linear SVM is chosen due to its high efficiency and scalability for learning and model propagation in distributed networks.

4.3.3 Performance of the compared approaches

This section compares the four approaches on accuracy, communication and computational cost.

Figure 4.6: Moving average of prequential error rate over 100 data points.

Accuracy In Figure 4.6 the accuracies of the approaches are compared. The increase in error rates indicates that concepts in peers start to drift and the decrease indicates that the new concepts have been learnt. For the gradual hyperplane dataset (Figure 4.6.a), it is observed that the prequential error of all approaches decreases as time passes, especially for RePCoDE and PINE which become much better than Local and All, and PINE achieves the lowest error among all. This is because both RePCoDE and PINE learn from historical data and hence can reduce error as time passes. It is also found that for all the concept drifts, i.e., when the error starts to rise, PINE can detect and adapt to the concept drift faster and thus achieves lower peaks in error throughout.

For the sudden hyperplane dataset (Figure 4.6.b), it can be seen that PINE consistently achieves lower prequential error than all other approaches followed by RePCoDE, then All and finally Local. This is most likely due to the fact that in this dataset, delay among the occurrences of peers' concept drifts are fixed and the drifts are sudden. Hence PINE can consistently detect concept drifts and accurately learn the temporal association between peers' concept drift occurrences.

For the weather dataset (Figure 4.6.c), it is observed that concept drifts consistently occurs around April every year, which corresponds to spring and autumn in the northern and southern hemisphere respectively. This could indicate that the patterns of rainfall change every year, instead of quarterly when seasons change. It is also found that initially PINE is comparable to RePCoDE, but as time passes, its prequential error becomes lower than that of RePCoDE. While not shown in the figures, note that PINE has higher prequential error in all datasets initially. However, the prequential error quickly reduces to become lower than all approaches over time. This is because PINE learns associations from the history of concept drift occurrences and hence several occurrences are needed to learn the pattern of the drifts before it can perform well.

Given that RePCoDE performs better than Local and All in most datasets, it can be seen that prediction of the future concepts does indeed reduce prequential errors even though RePCoDE uses the fixed window drift detection which is in general slower in

¹<http://www.ncdc.noaa.gov/cgi-bin/res40.pl?page=gsod.html>

Table 4.3: Average communication cost per peer in terms of kilobyte (models propagated).

Approach	Gradual	Sudden	Weather
RePCoDE	634.80 (72.87)	820.99 (94.24)	654.30 (12.61)
PINE	293.51 (33.69)	609.13 (69.92)	616.73 (11.88)

Table 4.4: Average time taken (in msecs).

Data	RePCoDE	Local	All	PINE
Model Construction				
Gradual	0.227	2.813	2.813	2.813
Sudden	0.252	1.311	1.311	1.311
Weather	131.3	509.2	509.2	509.2
Pre-prediction				
Gradual	0.370	-	-	37.32
Sudden	0.394	-	-	178.0
Weather	11.48	-	-	71.56
Prediction				
Gradual	0.003	0.0003	0.018	0.009
Sudden	0.004	0.0004	0.019	0.010
Weather	0.010	0.0004	0.093	0.034

concept drift detection. Moreover, PINE achieving the lowest prequential error demonstrates that in addition to predicting future concepts, predicting when concept drifts occurs can further reduce the prequential error.

Communication and computational costs Tables 4.3 and 4.4 present communication and computational costs of the compared approaches.

The main communication cost incurred by the approaches is the sending of models, presented in Table 4.3. Note that Local does not incur any communication cost and the additional cost incurred by PINE compared to *All* is the cost needed to notify the time of drift. Hence, only the results of RePCoDE and PINE are presented. Table 4.3 shows that PINE incurs less communication cost compared to RePCoDE. The difference is most likely due to the adaptive detection and propagation of models compared to the fixed window drift detection and fixed delay model propagation.

Table 4.4 presents the average time taken by each peer to perform various tasks. Model construction refers to the time taken to build a classifier. Pre-prediction refers to the computations required before the prediction, i.e., indexing, model retrieval and sequence similarity matching for RePCoDE and temporal association mining, temporal

association rule matching and model verifications for PINE. Prediction refers to the time taken to predict on one unlabelled data point. Even though PINE incurs the most computational cost compared to other approaches, it is still acceptable as the absolute time taken is very low, i.e., all tasks can be computed in less than one second.

4.4 Summary

This chapter studied the problem of classification in P2P networks with asynchronous concept drifts and presented two different approaches, viz., RePCoDE and PINE. These approaches combine both reactive and proactive techniques to speed up the models' adaptation of concept drifts. Experimental results showed that both RePCoDE and PINE perform better than existing approaches that often can hardly handle concept drift in P2P environments. However, to achieve high accuracy, RePCoDE has to accurately detect concept drifts, which is dependent on the static drift detection threshold, and is prone to higher communication cost. On the contrary, PINE is less sensitive to the parameters and can better detect concept drifts but as it does not store historical models like RePCoDE does, PINE is less suitable for scenario where one would expect recurring concept drifts.

Chapter 5

Classification in P2P Networks with Privacy Preservation

This chapter addresses the problem of classification in P2P networks with privacy preservation. To overcome the limitation of existing approaches, the proposed approach, PRICADE, employs the secure multi-party computation together with multi-tier cascade ensemble framework. As such, it is able to fulfill all the basic properties of P2P classification algorithms and minimize the disclosure of peers' information and loss of accuracy due to privacy preservation. The following sections will introduce the proposed solution, PRICADE, and present the experimental results to demonstrate its effectiveness in addressing the problem of classification in P2P networks with privacy preservation.

5.1 P2P Privacy-Preserving SVM Framework (PRICADE)

This section presents PRICADE, an incremental cascade learning approach for privacy-preserving SVMs in P2P systems. First, ensembles of SVM models are constructed from the subsets of peers. These models are then incrementally combined into larger SVM models until a single SVM model is constructed. By constructing the ensemble of SVM, the number of peers participating in each model construction can be minimized and hence minimizing the degree of synchronization. Assuming that all participating peers are required for model construction and prediction, peer failures will only affect the subset of (minimum) peers. In addition, a (partial) solution can still be generated at any time based on the models of other subsets. However, with only the ensemble of SVM, it may not be possible to achieve high accuracy. Hence with cascading, accuracy can increase as the number of cascaded models increases (i.e., more peers/data used for construction) [AGNH09].

Figure 5.1 shows an overview of the proposed framework. This section first details the process of the proposed cascade approach for constructing the P2P privacy-preserving SVM. Next, the proposed multiparty privacy-preserving SVM algorithm, which is the basic building block of the proposed P2P approach, is described. Finally, the discussion on how secure SVM prediction is performed in the P2P network is provided.

Figure 5.1: Privacy-preserving P2P SVM construction flowchart.

5.1.1 P2P secure SVM construction

At the setup stage, peers build Reduced SVM (RSVM) [LL03] models from their local data. The resulting support vectors will then be used for subsequent P2P SVM training, which avoids engaging the entire local data set [AGNH09].

To construct and combine sub-models, a group of peers which has already constructed a (secure) SVM model will first search for other candidate groups (groups with the same number of peers) in the P2P network. This is achieved based on the distributed hash table (DHT) P2P network [BKK⁺03] by using the size of the group (individual peer's group size is one) as the identifier key to the network addresses of the corresponding peers.

Once the list of candidate groups is found, one of them will be chosen. The selection criteria is based on proximity of reducing the training and communication cost. Peers from both groups (current and the chosen group) then jointly construct a new secure SVM model using the proposed multiparty privacy-preserving SVM approach presented in Protocol 5.1. Note that individual peers will instead merge with two other peers for constructing the secure SVM. This cascading process repeats until all groups are combined to construct a single SVM model.

In the previous secure SVM model constructions, peers of the same groups have already computed the secure scalar product on their training data. Hence for the cascading, secure scalar product computations only need to be done among peers from different groups, eliminating redundancy in the SVM kernel computation that is a large portion of computation and communication cost in SVM training.

At every stage of cascading or the secure SVM model construction, the Lagrange multipliers of the constructed models will be retained by peers. By gradually increasing the size of the groups to cascade, the dependency among the peers are increased. Assuming that a stable state of the P2P network exists, a single secure SVM model will eventually be constructed via the cascading process. However, if any peer fails at some point in time, existing peers can simply be rolled back to their previous group using the cached Lagrange multipliers. In another words, the single secure SVM model will be split into several smaller secure SVM models. The rollback of secure models can be a recursive process since the departed peers may have been involved in several cascades. This caching and rollback process ensures the fault tolerant mechanism of the proposed

approach, which allows prediction to be carried out when facing peer failures. Albeit at a slight decrease in accuracy, which, however, will be improved over time as groups of peers continue to cascade.

Last, the cascading is performed in a balanced and binary manner as without any prior knowledge on how peers will depart from the network, a balanced and binary approach can minimize the number of groups affected by peer failures. In addition, it can also minimize the effort of searching for secure models for prediction.

5.1.2 P2P secure prediction

Peers in the P2P network perform prediction task by first locating the secure SVM models in the P2P network. Furthermore, if this is a need to maximize the classification accuracy, peers should make use of most if not all of the secure SVM models. Since the cascading is performed in a balanced binary tree manner, peers need at most $\lg(N)$ to locate all secure SVM models. In addition, if the search is done incrementally, the peer can immediately stop searching when no model for a given size exists, since the cascade tree is balanced. Once the participating peers of the secure SVM models are identified, prediction can start by using Protocol 8. Predictions from these secure SVM models are then combined using the majority voting scheme. Note that cost of prediction depends on the total number of peers but not on the level of the cascaded groups.

5.1.3 Secure multiparty SMO

This section describes the proposed secure multiparty SVM, which is based on Sequential Minimal Optimization (SMO) for building the singular secure SVM model, a core part of the proposed P2P privacy-preserving SVM. First, a brief introduction of the SMO is provided followed by the presentation of the basic protocols to facilitate secure SMO, and the secure multiparty SMO protocols as well as the secure SVM prediction.

The objective of SVM is to solve the following quadratic programming (QP) problem:

$$\min_{\alpha} f(\alpha) = \frac{1}{2} \alpha^T Q \alpha - \mathbf{1}^T \alpha \quad (5.1)$$

subject to $0 \leq \alpha_i \leq C, i = 1, \dots, \ell$ and $\mathbf{y}^T \alpha = 0$, where $\mathbf{1}$ is the vector of all ones, C is the upper bound of all Lagrange multipliers, Q is an $\ell \times \ell$ symmetric matrix with $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ and $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function.

SMO is a decomposition method to solve the QP problem of SVM by iteratively minimizing the smallest sub problem, i.e., a pair α . The pseudocode of SMO [KG02] is as follows:

- (i) Find α^1 as the initial feasible solution. Set $k = 1$.
- (ii) If α^k is an optimal solution, stop. Otherwise, find a two-element working set $B = \{i, j\} \subset \{1, \dots, \ell\}$. Define $N \equiv \{1, \dots, \ell\} \setminus B$ and α_B^k and α_N^k to be sub-vectors of α^k corresponding to B and N , respectively.

- (iii) Solve the QP sub-problem with variable α_B .
- (iv) Set α_B^{k+1} to be the optimal solution of the sub-problem and $\alpha_N^{k+1} \equiv \alpha_N^k$. Set $k \leftarrow k + 1$ and go to Step 2.

Note that B changes for each iteration but for simplicity B is used instead of B^k .

As each iteration only updates two components, SMO suffers from slow convergences when the problem is difficult. Hence Keerthi and Gilbert [KG02] proposed to select the working set via the *maximal violating pair*:

- (i) Select

$$\begin{aligned}
 i &\in \arg \max_t \{-y_t \nabla f(\alpha)_t \mid t \in I_{\text{up}}(\alpha^k)\}, \\
 j &\in \arg \min_t \{-y_t \nabla f(\alpha)_t \mid t \in I_{\text{low}}(\alpha^k)\}, \\
 \text{where } I_{\text{up}}(\alpha^k) &\equiv \{t \mid \alpha_t < C, y_t = 1 \text{ or } \alpha_t > 0, y_t = -1\}, \\
 I_{\text{low}}(\alpha^k) &\equiv \{t \mid \alpha_t < C, y_t = -1 \text{ or } \alpha_t > 0, y_t = 1\}, \\
 \text{and } \nabla f(\alpha) &\equiv Q\alpha - \mathbf{1} \text{ is the gradient of } f(\alpha).
 \end{aligned} \tag{5.2}$$

- (ii) Return $B = i, j$

This selection algorithm can be derived through the Karush-Kuhn-Tucker (KKT) optimality condition of the quadratic optimization problem, such that a feasible α is a stationary point of the optimization problem if and only if:

$$m(\alpha) \leq M(\alpha) \tag{5.3}$$

where $m(\alpha) \equiv \max_{i \in I_{\text{up}}(\alpha)} -y_i \nabla f(\alpha)_i$, and

$$M(\alpha) \equiv \min_{i \in I_{\text{low}}(\alpha)} -y_i \nabla f(\alpha)_i.$$

The convergence of SMO based on the above working set selection is proved by Lin [Lin01]. For more in-depth details on SMO, please refer to [Lin01] or [KG02].

The decision function for predicting the class label is

$$\text{sgn} \left(\sum_{i=1}^{\ell} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) - \rho \right), \tag{5.4}$$

where the bias term ρ is

$$\rho = \sum_{0 < \alpha_t < C} y_t \nabla f(\alpha)_t / \sum_{0 < \alpha_t < C} 1, \tag{5.5}$$

if there exists α_t such that $0 < \alpha_t < C$; otherwise

$$\rho = 1/2 [\max_{t \in \hat{I}_{\text{low}}} y_t \nabla f(\alpha)_t + \min_{t \in \hat{I}_{\text{up}}} y_t \nabla f(\alpha)_t], \tag{5.6}$$

$$\begin{aligned}
 \text{where } \hat{I}_{\text{low}} &= \{t \mid \alpha_t = 0, y_t = -1 \text{ or } \alpha_t = C, y_t = 1\}, \\
 \hat{I}_{\text{up}} &= \{t \mid \alpha_t = 0, y_t = 1 \text{ or } \alpha_t = C, y_t = -1\}.
 \end{aligned}$$

Basic protocols In the above section, computations of max and min for $\nabla f(\alpha)$ are required but to preserve privacy, these values are shared between peers. Hence, the following basic protocols are introduced for computing max (min) of shared values which facilitate the remaining protocols for secure SMO computation. PRICADE's secure computations are based on the Paillier cryptosystem [Pai99]¹. For instance, let $e =$

¹Homomorphic (additive) properties of Paillier cryptosystem: $\text{Enc}(a + b) = \text{Enc}(a) \times \text{Enc}(b)$ and $\text{Enc}(a * b) = \text{Enc}(a)^b$.

$Enc(v, P_H)$ denote the encryption of value v with the public key of peer P_H and $Dec(e)$ denote the decryption of e (the use of private key of peer P_H is implicit).

Local Max of Shares (SLMS) SLMS securely finds the maximum of the sum of two vectors of shares of two parties.

Input. Parties P_1 and P_2 each holding a corresponding vector of shares $\mathbf{v1}$ and $\mathbf{v2}$.

Output. Index of the max value $m \in \arg \max_t \mathbf{v1}_t + \mathbf{v2}_t$.

- (i) Initially, parties P_1 and P_2 generate a current vector $LC = \{1, \dots, |\mathbf{v1}|\}$ and an empty vector $LT = \emptyset$.
- (ii) For every sequential pair of $\mathbf{v1} + \mathbf{v2}$ values (e.g., $\mathbf{v1}_i + \mathbf{v2}_i$ and $\mathbf{v1}_j + \mathbf{v2}_j$ where $i < j$), parties P_1 and P_2 jointly and securely determine which is larger by computing $a + b \geq 0$ using Yao's circuit evaluation protocol [MNPS04, Yao86], where party P_1 's input is $a = \mathbf{v1}_i - \mathbf{v1}_j$ and P_2 's input is $b = \mathbf{v2}_i - \mathbf{v2}_j$. Hence, if $a + b \geq 0$ is true, then $\mathbf{v1}_i + \mathbf{v2}_i$ is larger, so append i to LT . Otherwise $\mathbf{v1}_j + \mathbf{v2}_j$ is larger, so append j to LT .
- (iii) When all pairs are evaluated, set $LC \leftarrow LT$ and $LT \leftarrow \emptyset$. If only one index is left in LC , the element with the max value is found, return $m = LC$. Otherwise, go to step 2.

Secure Global Max of Shares (SGMS) SGMS securely finds the global maximum from N pairs of corresponding vector of shares held by N parties.

Input. Every party P_H ($1 \leq H \leq N$) holds a vector of local shares $\mathbf{v1}_H$ and a vector of remote shares $\mathbf{v2}_{\Omega H}$.

Output. Index of the peer and data with the global maximum value $M \in \arg \max_H \mathbf{v1}_{Ht} + \mathbf{v2}_{Ht}$.

- (i) Every party P_H uses SLMS (with $P_{\hat{H}}$) to find $m_H \in \arg \max_t \mathbf{v1}_{Ht} + \mathbf{v2}_{Ht}$. The respective inputs are $\mathbf{v1}_H$ and $\mathbf{v2}_H$.
- (ii) All parties generate a current vector $LC = \{1, \dots, N\}$ and an empty vector $LT = \emptyset$ individually.
- (iii) Each sequential pair of parties (P_I and $P_J, I < J$) in LC jointly and securely determine which pair of shares is larger by computing $a + b \geq 0$ ($\mathbf{v1}_{Im_I} + \mathbf{v2}_{Im_I} \geq \mathbf{v1}_{Jm_J} + \mathbf{v2}_{Jm_J} \equiv \mathbf{v1}_{Im_I} + \mathbf{v2}_{Im_I} - \mathbf{v1}_{Jm_J} - \mathbf{v2}_{Jm_J} \geq 0$) using Yao's protocol, where a and b are held by P_I and P_J respectively. $P_{\hat{I}}$ generates and sends a random number $s1_I$ to P_I and sends $s2_I = \mathbf{v2}_{Im_I} - s1_I$ to P_J . $P_{\hat{J}}$ generates and sends a random number $s1_J$ to P_I and sends $s2_J = \mathbf{v2}_{Jm_J} - s1_J$ to P_J . Hence, P_I 's input is $a = \mathbf{v1}_{Im_I} + s1_I - s1_J$ and P_J 's input is $b = -\mathbf{v1}_{Jm_J} + s2_I - s2_J$. If $a + b \geq 0$ is true, then P_I has the larger value, otherwise P_J has the larger value. P_I and P_J broadcast the results to parties in LC and append the party's index to LT .

Protocol 5.1: Secure SMO optimization.

-
- 1 All peers perform the initialization based on Protocol 2;
 - 2 **while** true **do**
 - 3 All peers jointly select working set and find the corresponding peers using Protocol 3;
 - 4 Working set peers jointly perform optimality check using Protocol 4;
 - 5 **if** solution is optimal **then** break;
 - 6 Working set peers jointly update Lagrange multipliers of working set using Protocol 5;
 - 7 All peers jointly update the local and remote gradients of all data points with the working set peers using Protocol 6;
 - 8 All peers jointly compute bias using Protocol 7;
-

Figure 5.2: Dependency diagram of secure multiparty SMO.

- (iv) When all pairs are evaluated, set $LC \leftarrow LT$ and $LT \leftarrow \emptyset$. If only one index is left in LC , the party with the maximum variable m is found. Then broadcast $M = LC$ to all parties and index m_M of the largest value is known only to parties P_M and $P_{\hat{M}}$. Otherwise, go to step 2.

Secure Multiparty SMO Construction This section presents the proposed secure multiparty SMO, cf. Protocol 5.1, which closely follows the original. Section 5.1.3 describes the initialization process necessary to perform the secure model construction. Section 5.1.3 describes the secure working set selection (5.2). Section 5.1.3 describes the secure optimality check (5.3). Section 5.1.3 describes the secure update of the Lagrange multipliers [Lin01] and Section 5.1.3 describes the secure gradient update (5.2). Finally, Section 5.1.3 describes the secure bias ρ computation (5.5). Dependency among the protocols is presented in Figure 5.2, and a list of commonly used notations is provided in Table 5.1.

Here, assume that peers are semi-honest, i.e., they will follow the protocol and perform correct computations but may retain records of the intermediate computation results which may be used later to derive the other peers' data. Moreover, it is assumed

Table 5.1: Commonly used notations.

Notation	Description
N	Number of peers
$P_H, P_{\Omega_H}, P_{\hat{H}}$	The H -th peer, its preceding & succeeding peer
$\mathbf{D}_H, \ell_H$	Dataset of peer $P_H, \ell_H = \mathbf{D}_H $
$k(\mathbf{x}_i, \mathbf{x}_j)$	Kernel function of data points $\mathbf{x}_i$ and $\mathbf{x}_j$
$\mathbf{S}(i, j)$	The share of $K(\mathbf{x}_i, \mathbf{x}_j)$ held by the peer with $\mathbf{x}_i$
$\mathbf{y}$	Class label vector $([y_1, \dots, y_\ell]^T)$
α_{Hh}	Lagrange multiplier of data point $\mathbf{x}_h$ of peer P_H
$\mathbf{GL}, \mathbf{GR}$	Local and remote share of gradient $(\nabla f(\boldsymbol{\alpha}))$
ρ	Bias of SVM model
ϵ	Optimality (KKT) tolerance threshold
C	Soft margin SVM penalization cost parameter

that peers are non-colluding, which allows the proving of the privacy-preserving of the proposed approach. These assumptions are not unrealistic and have been used in many other existing works [HNWL10, YZ09, YJV06].

Protocol 2: Initialization To perform secure computations, peers have to first broadcast their public keys which are required for performing encryptions. In addition, they also have to broadcast their class labels which are used in most of the SMO computations. Revealing the class labels only tells other peers how many data of each class they hold and nothing else. Next, peers initialize the corresponding Lagrange multipliers α to 0. To make it privacy-preserving, instead of the corresponding peer P_H holding on to the gradients of the objective function for their own data, the gradients are split into two shares where the local peer P_H has $\mathbf{GL}_H$ and the remote peer $P_{\hat{H}}$ has $\mathbf{GR}_H$. For the purpose of simplicity and load balancing, the remote peer $P_{\hat{H}}$ is fixed as the peer in the next sequence. The local and remote peers will initialize their respective local and remote shares to -0.5.

Thereafter, every pair of peers jointly compute the secure scalar products of their data. Linear kernel is used in this work which simplifies the scalar products, hence no other addition computation is necessary. However, note that other kernel functions can also be securely computed as shown by Han and Ng [HN08]. Here, the secure scalar product protocol as proposed by Goethals *et al.* [GLLM05] is used. Let $\mathbf{S}(i, j) + \mathbf{S}(j, i) = \mathbf{x}_i \cdot \mathbf{x}_j$ denote the output of the secure scalar product, where $\mathbf{S}(i, j)$ and $\mathbf{S}(j, i)$ are held by peers holding $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively. This ends the initialization phase and next is the working set selection.

Protocol 3: Working set selection As shown in Algorithm 5.1, each iteration of SMO requires the selection of a working set. Below gives the protocol for all peers to jointly and securely select the working set.

Input. Every peer P_H ($1 \leq H \leq N$) has its local gradients $\mathbf{GL}_H$ and the remote gradients $\mathbf{GR}_{\Omega H}$ of $P_{\Omega H}$.

Output. Indexes I and J of the peers holding the *maximal violating pair* (5.2) i and j respectively.

- (i) Every peer P_H locally computes and sends I_{upH} and I_{lowH} of its own data to $P_{\hat{H}}$.
- (ii) All peers jointly and securely find P_I and P_J holding the *maximal violating pair* i and j (5.2) using SGMS. The inputs of each peer P_H are $\{-\mathbf{y}_{Ht}\mathbf{GL}_{Ht} \mid t \in I_{upH}\}$ and $\{-\mathbf{y}_{(\Omega H)t}\mathbf{GR}_{(\Omega H)t} \mid t \in I_{up(\Omega H)}\}$ for computing i and $\{\mathbf{y}_{Ht}\mathbf{GL}_{Ht} \mid t \in I_{lowH}\}$ and $\{\mathbf{y}_{(\Omega H)t}\mathbf{GR}_{(\Omega H)t} \mid t \in I_{low(\Omega H)}\}$ for computing j . SGMS outputs indexes I and J of peers holding the *maximal violating pair* to all peers while the index i is known only by P_I and $P_{\hat{I}}$ and the index j only by P_J and $P_{\hat{J}}$.

Note that specifically for the first iteration, the working set is restricted to come from the first peer, where computations are done locally. This is to prevent the revelation of the private data when computing δ of Protocol 5 as all initial gradients are of the same values.

Protocol 4: Optimality check Once the working set is selected, peers P_I , P_J and their corresponding remote peers jointly check if the optimal solution is obtained (5.3). As SMO only asymptotically approaches an optimum, instead of using (5.3), a small tolerance $\epsilon > 0$ is defined where the optimization can terminate when the maximal violation is small enough:

$$m(\alpha) - M(\alpha) < \epsilon. \quad (5.7)$$

Input. Peers P_I , $P_{\hat{I}}$, P_J and $P_{\hat{J}}$ holding $\mathbf{GL}_{Ii}$, $\mathbf{GR}_{Ii}$, $\mathbf{GL}_{Jj}$ and $\mathbf{GR}_{Jj}$ respectively and the constant ϵ , where $\nabla f(\alpha)_t = \mathbf{GL}_i + \mathbf{GR}_t$.

Output. Whether (5.7) is satisfied.

- (i) P_I and P_J jointly check if (5.7) is satisfied by computing $a+b \geq 0$ ($m(\alpha) - M(\alpha) < \epsilon \equiv \neg(m(\alpha) - M(\alpha) - \epsilon \geq 0)$) using Yao's protocol [Yao86] where a and b are held by P_I and P_J respectively. $P_{\hat{I}}$ generates a random number $s1_I$ and sends $s1_I$ to P_I and $s2_I = -\mathbf{y}_{Ii}\mathbf{GR}_{Ii} - s1_I$ to P_J . $P_{\hat{J}}$ generates a random number $s1_J$ and sends $s1_J$ to P_I and $s2_J = \mathbf{y}_{Jj}\mathbf{GR}_{Jj} - s1_J$ to P_J . Hence, P_I 's input is $a = -\mathbf{y}_{Ii}\mathbf{GL}_{Ii} + s1_I + s1_J - \epsilon/2$ and P_J 's input is $b = \mathbf{y}_{Jj}\mathbf{GL}_{Jj} + s2_I + s2_J - \epsilon/2$.
- (ii) If $a + b \geq 0$ is true, then (5.7) is not satisfied and P_i and P_J broadcast indexes i and j of the *maximal violating pair* to all peers. Otherwise, broadcast the signal to terminate training and all peers jointly and securely compute bias ρ , cf., Section 5.1.3.

Protocol 5: Lagrange multipliers update Since the solution is not optimal, peers P_I and P_J proceed to update the Lagrange multipliers of the working set. This update includes computing the initial change in the multipliers, projection back to the feasible region and the final change computation. Only the change computation is performed securely as it risks revealing private information. Note that peers P_I and P_J need to share the Lagrange multipliers of i and j in order to update their values and project back to the feasible region. However, this does not reveal any private information except the number of support vectors held, since Lagrange multipliers are bounded by zero and C and only support vectors have nonzero values.

Input. Peers $P_I, P_{\hat{I}}, P_J$ and $P_{\hat{J}}$ holding $\mathbf{GL}_{Ii}, \mathbf{GR}_{Ii}, \mathbf{GL}_{Jj}$ and $\mathbf{GR}_{Jj}$. In addition, peer P_I holds $K(\mathbf{x}_i, \mathbf{x}_i)$ and $S(i, j)$ and peer P_J holds $K(\mathbf{x}_j, \mathbf{x}_j)$ and $S(j, i)$.

Output. Updated α_i and α_j and final change in Lagrange multipliers $\Delta\alpha_i$ and $\Delta\alpha_j$ known to both P_I and P_J .

- (i) $P_I, P_{\hat{I}}, P_J$ and $P_{\hat{J}}$ jointly compute the initial change in the multipliers $\delta = [-\mathbf{y}_{Ii}(\mathbf{GL}_{Ii} + \mathbf{GR}_{Ii}) + \mathbf{y}_{Jj}(\mathbf{GL}_{Jj} + \mathbf{GR}_{Jj})] / [K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j)]$.
 - (a) P_I and P_J jointly and securely compute the inverse $K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j) \equiv K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2S(i, j) - 2S(j, i)$ according to the secure function of sum protocol ($\varphi(a + b)$) [HNWL10] where in this case φ is the inverse function and $a = K(\mathbf{x}_i, \mathbf{x}_i) - 2S(i, j)$ and $b = K(\mathbf{x}_j, \mathbf{x}_j) - 2S(j, i)$ provided by P_I and P_J respectively. The outputs of the protocol are two shares k_1 and k_2 ($k_1 + k_2 = \varphi(a + b)$) held by P_I and P_J respectively.
 - (b) Using $s1_I, s2_I, s1_J$ and $s2_J$ computed from Protocol 4, P_I and P_J individually compute $g_1 = -\mathbf{y}_{Ii}\mathbf{GL}_{Ii} + s1_I + s1_J$ and $g_2 = \mathbf{y}_{Jj}\mathbf{GL}_{Jj} + s2_I + s2_J$ respectively.
 - (c) P_I and P_J securely and jointly compute $\delta = (g_1 + g_2) \times (k_1 + k_2) = R_1 + R_2$ using Random Shares Protocol [HN08] where R_1 and R_2 are held by P_I and P_J respectively. P_I then sends R_1 to P_J and P_J sends R_2 to P_I and they individually compute $\delta = R_1 + R_2$.
- (ii) P_I and P_J individually use Algorithm 5.1 to update the Lagrange multipliers and obtain $\Delta\alpha_i$ and $\Delta\alpha_j$.

Protocol 6: Gradient update Once the Lagrange multipliers of the working set are updated, all peers including P_I and P_J jointly update their gradients.

Input. Peer P_I has $\Delta\alpha_i$, its portion of kernel shares of $\mathbf{x}_i$ corresponding to all other data points ($\forall t, S(i, t)$) and its portion of kernel share of its own data corresponding to $\mathbf{x}_j$, i.e., $S(t, j), \forall \mathbf{x}_t \in \mathbf{D}_I$. Similarly for peer P_J . Every other peer P_H has its portion of kernel shares corresponding to $\mathbf{x}_i$ and $\mathbf{x}_j$, i.e., $S(t, i)$ and $S(t, j), \forall \mathbf{x}_t \in \mathbf{D}_H$.

Output. Updated gradients of every peer P_H split into $\mathbf{GL}_H$ held by P_H and $\mathbf{GR}_H$

Algorithm 5.1: Lagrange multiplier projection.

input : $\alpha_i, \alpha_j, \delta$ and $\mathbf{y}$;
output: Updated α_i, α_j , final change $\Delta\alpha_i, \Delta\alpha_j$
1 Set $\alpha'_i \leftarrow \alpha_i$ and $\alpha'_j \leftarrow \alpha_j$;
2 Set $\text{sum} \leftarrow \mathbf{y}_i\alpha_i + \mathbf{y}_j\alpha_j$;
3 Set $\alpha_i \leftarrow \alpha_i + \mathbf{y}_i\delta$ and $\alpha_j \leftarrow \alpha_j - \mathbf{y}_i\delta$;
4 **if** $\alpha_i > C$ **then** Set $\alpha_i \leftarrow C$;
5 **if** $\alpha_i < 0$ **then** Set $\alpha_i \leftarrow 0$;
6 Set $\alpha_j \leftarrow \mathbf{y}_j(\text{sum} - \mathbf{y}_i\alpha_i)$;
7 **if** $\alpha_j > C$ **then** Set $\alpha_j \leftarrow C$;
8 **if** $\alpha_j < 0$ **then** Set $\alpha_j \leftarrow 0$;
9 Set $\alpha_i \leftarrow \mathbf{y}_i(\text{sum} - \mathbf{y}_j\alpha_j)$;
10 Set $\Delta\alpha_i \leftarrow \alpha_i - \alpha'_i$ and $\Delta\alpha_j \leftarrow \alpha_j - \alpha'_j$;

held by $P_{\hat{H}}$.

Every peer P_H securely and jointly update its local and remote gradient with P_I, P_J and $P_{\hat{H}}$. Due to space limitations, only the case where $P_H \neq P_I \neq P_J \neq P_{\hat{H}}$ is shown. Other cases can be computed similarly.

- (i) $\forall \mathbf{x}_t \in \mathbf{D}_H$, P_H sends $si_t = \text{Enc}(\mathbf{y}_i\mathbf{y}_tS(t, i), P_H)$ to P_I and $sj_t = \text{Enc}(\mathbf{y}_j\mathbf{y}_tS(t, j), P_H)$ to P_J .
- (ii) P_I sets $gi_t \leftarrow (si_t \text{Enc}(\mathbf{y}_i\mathbf{y}_tS(i, t), P_H))^{\Delta\alpha_i}$, randomly generates ui_t and send $gi'_t = gi_t \text{Enc}(-ui_t, P_H)$ to P_H and ui_t to $P_{\hat{H}}$.
 P_J sets $gj_t \leftarrow (sj_t \text{Enc}(\mathbf{y}_j\mathbf{y}_tS(j, t), P_H))^{\Delta\alpha_j}$, randomly generates uj_t and send $gj'_t = gj_t \text{Enc}(-uj_t, P_H)$ to P_H and uj_t to $P_{\hat{H}}$.
- (iii) Peer P_H sets $\mathbf{GL}_{Ht} \leftarrow \mathbf{GL}_{Ht} + \text{Dec}(gi'_t) + \text{Dec}(gj'_t)$.
Peer $P_{\hat{H}}$ sets $\mathbf{GR}_{Ht} \leftarrow \mathbf{GR}_{Ht} + ui_t + uj_t$.

Protocol 7: Bias computation Once the optimal α is obtained, the final step is to compute the bias term ρ in the SVM decision function (5.4).

Input. Every peer P_H has its Lagrange multipliers α_H , its local gradients $\mathbf{GL}_H$ and the remote gradients $\mathbf{GR}_{\Omega H}$ of $P_{\Omega H}$.

Output. Bias term ρ .

- (i) All peers jointly compute (5.5)
 - (a) Every peer P_H locally computes and sends $LF_H = \{t \mid 0 < \alpha_t < C\}$ to peer $P_{\hat{H}}$.
 - (b) Every peer P_H broadcast $|LF_H|$ and $\rho_H = \sum_{t \in LF_H} \mathbf{y}_{Ht} \mathbf{GL}_{Ht} + \sum_{t \in LF_{\Omega H}} \mathbf{y}_{\Omega Ht} \mathbf{GR}_{\Omega Ht}$ to all peers.
 - (c) Every peer computes $\rho = (\sum_H \rho_H) / (\sum |LF_H|)$ individually. If $\rho \neq 0$, return ρ . Otherwise, go to step 2.

(ii) All peers jointly compute (5.6)

(a) Every peer P_H locally computes and sends $\hat{I}_{lowE}$ and $\hat{I}_{upE}$ of (5.6) to peer $P_{\hat{H}}$.

(b) All peers jointly find the global

$$\rho_i \in \arg \max_t \{y_t \nabla f(\boldsymbol{\alpha})_t \mid t \in \hat{I}_{low}\} \text{ and}$$

$$\rho_j \in \arg \max_t \{-y_t \nabla f(\boldsymbol{\alpha})_t \mid t \in \hat{I}_{up}\} \text{ using SGMS. The inputs of each peer } P_H$$

are $\{\mathbf{y}_{Ht} \mathbf{GL}_{Ht} \mid t \in \hat{I}_{lowE}\}$ and $\{\mathbf{y}_{\Omega Ht} \mathbf{GR}_{\Omega Ht} \mid t \in \hat{I}_{low\Omega H}\}$ for computing ρ_i and $\{-\mathbf{y}_{Ht} \mathbf{GL}_{Ht} \mid t \in \hat{I}_{upE}\}$ and

$$\{-\mathbf{y}_{\Omega Ht} \mathbf{GR}_{\Omega Ht} \mid t \in \hat{I}_{up\Omega H}\} \text{ for computing } \rho_j.$$

(c) Peers holding ρ_i (P_I) and ρ_j (P_J) and their corresponding remote peers $P_{\hat{I}}$ and $P_{\hat{J}}$ jointly compute ρ .

i. $P_I, P_{\hat{I}+1}, P_J$ and $P_{\hat{J}}$ randomly generate u_1, u_2, u_3 and u_4 resp.

ii. P_I sends $e_1 = \mathbf{y}_{I\rho_i} \mathbf{GL}_{I\rho_i} - u_1$ to $P_{\hat{I}}$.

$P_{\hat{I}}$ sends $e_2 = \mathbf{y}_{I\rho_i} \mathbf{GR}_{I\rho_i} - u_2$ to P_J .

P_J sends $e_3 = \mathbf{y}_{J\rho_j} \mathbf{GL}_{J\rho_j} - u_3$ to $P_{\hat{J}}$.

$P_{\hat{J}}$ sends $e_4 = \mathbf{y}_{J\rho_j} \mathbf{GR}_{J\rho_j} - u_4$ to P_I .

iii. P_I sends $\rho_1 = e_4 + u_1$ to all peers. $P_{\hat{I}}$ sends $\rho_3 = e_1 + u_2$ to all peers. P_J sends $\rho_3 = e_2 + u_3$ to all peers. $P_{\hat{J}}$ sends $\rho_4 = e_3 + u_4$ to all peers.

iv. All peers compute $\rho = (\rho_1 + \rho_2 + \rho_3 + \rho_4)/2$ individually.

Secure prediction To obtain the class labels of unlabelled data, the testing peer computes (5.4) jointly with all peers of the secure model.

Input. Test data points $\mathbf{D}_T$ of peer P_T , Lagrange multipliers $\boldsymbol{\alpha}_H$ and data points $\mathbf{D}_H$ of peer P_H ($1 \leq H \leq N$).

Output. Predicted class labels $\mathbf{y}$.

(i) $\forall \mathbf{x}_h \in \mathbf{D}_T, P_T$ sets $\mathbf{e}_h = \{Enc(\mathbf{x}_{h,i}, P_T) \mid i \in \{1, \dots, d\}\}$. P_T sends $\mathbf{DE} = \{\mathbf{e}_h \mid h \in \{1, \dots, |\mathbf{D}_T|\}\}$ to all peers.

(ii) $\forall \mathbf{e}_h \in \mathbf{DE}$, every peer P_H randomly generate $\mathbf{u}_{Hh}$, sets $\mathbf{e}'_h = \prod_{t \in \{t \mid \alpha_{Ht} > 0\}} (\prod_{i=1}^d e_i^{\mathbf{x}_{Ht,i}}) \boldsymbol{\alpha}_{Ht}$ and send $\mathbf{v}_{Hh} = (\prod \mathbf{e}'_h) Enc(-\mathbf{u}_{Hh}, P_T)$ to $P_{\hat{H}}$.

(iii) Every peer P_H sends $\mathbf{y}_{Hh} = \mathbf{v}_{\Omega Hh} Enc(\mathbf{u}_{Hh}, P_T)$ to P_T .

(iv) Peer P_T computes $\mathbf{y}_h = \text{sgn}((\sum_{H=1}^N Dec(\mathbf{y}_{Hh})) - \rho)$.

5.1.4 Analysis

This section analyses the proposed protocols showing that they are privacy-preserving and provides the computational complexity and communication cost for the proposed framework.

5.1.5 Security Analysis

Here, the security of all protocols using the simulation paradigm and Composition Theorem [Gol04] is proven. In general, the security of an algorithm can be proven by the simulator paradigm when the messages received from other peers can be simulated using probabilistic polynomial-time algorithms. In addition, the Composition Theorem [Gol04] states that assuming a function g is privately reducible to function f and there exists a protocol for privately computing f , then there must exist a protocol for privately computing g .

Security Proof of SLMS: Step 2 is privately computed by Yao's secure protocol [Yao86].

Hence, SLMS is secure.

Security Proof of SGMS: Steps 1 and 3 are privately computed using SLMS and Yao's protocol [Yao86] which have been proven secure. In addition, step 3 reveals $s1_I, s2_I, s1_J$ and $s2_J$. The probability of the view of message $s1_I$ received from $P_{\hat{I}}$ is the same as the probability of simulating $r = s1_I$, where r has a uniform distribution over a finite field $\mathcal{F}$. Hence, $Pr\{VIEW = s1_I\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = s1_I\}$. The probability of the view of message $s2_I$ received from $P_{\hat{I}}$ is $Pr\{VIEW = s2_I\} = P\{s2_I = \mathbf{v}2_{Im_I} - s1_I\} = Pr\{s1_I = \mathbf{v}2_{Im_I} - s2_I\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = s2_I\}$. Proofs are similar for $s1_J$ and $s2_J$. Hence, SGMS is secure.

Security Proof of Protocol 3: In step 1, each peer reveals the indices of unbounded support vectors and nothing else. Step 2 is privately computed using SGMS which has been proven secure. Hence, Protocol 3 is secure.

Security Proof of Protocol 4: In step 1, similar to above the proof, for the view of message $s1_I$ received from $P_{\hat{I}}$, hence $Pr\{VIEW = s1_I\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = s1_I\}$. The probability of the view of message $s2_I$ received from $P_{\hat{I}}$ is $Pr\{VIEW = s2_I\} = P\{s2_I = -\mathbf{y}_{I_i}\mathbf{GR}_{I_i} - s1_I\} = Pr\{s1_I = -\mathbf{y}_{I_i}\mathbf{GR}_{I_i} - s2_I\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = s2_I\}$. Proofs are similar for $s1_J$ and $s2_J$. In addition, step 1 is privately computed using Yao's protocol [Yao86]. Hence, Protocol 4 is secure.

Security Proof of Protocol 5: Step 1a is privately computed using secure function of sum protocol [HNWL10] which has been proven secure. The view of messages $s1_I, s2_I, s1_J$ and $s2_J$ in step 1b have been demonstrated in the security proof of Protocol 4. Step 1c is privately computed using Random Shares Protocol [HN08] which has been proven secure. Step 1c also reveals $R_1 = (g_1 \times k_1) + r_1$ and $R_2 = (g_2 \times k_2) + r_2$, where $r_1 + r_2 = (g_1 \times k_2) + (g_2 \times k_1)$ and r_1 and r_2 are random shares with probability density $1/|\mathcal{F}|$ [GLLM05]. Hence, similar to the above proof, the probability of the view of message R_1 received from P_I is $Pr\{VIEW = R_1\} = Pr\{R_1 = (g_1 \times k_1) + r_1\} = Pr\{r_1 = R_1 - (g_1 \times k_1)\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = R_1\}$. Proof for R_2 is similar. In step 2, P_I and P_J each reveal a Lagrange multiplier α which indicate whether

the corresponding data point is a support vector or not and cannot be used to derive other information. Hence, Protocol 5 is secure.

Security Proof of Protocol 6: In step 1, si_t and sj_t are revealed but as they are encrypted, it is considered to be secure. In step 2, similar to the above proof, the probability of the view of message ui_t received from P_I is $Pr\{VIEW = ui_t\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = ui_t\}$. The probability of the view of message gi'_t received from P_I is $Pr\{VIEW = gi'_t\} = P\{gi'_t = (y_i y_t S(t, i) + y_i y_t S(i, t))\Delta\alpha_i - ui_t\} = Pr\{ui_t = (y_i y_t S(t, i) + y_i y_t S(i, t))\Delta\alpha_i - gi'_t\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = gi'_t\}$. The proofs are similar for uj_t and gj'_t . Hence, Protocol 6 is secure.

Security Proof of Protocol 7: In step 1b, every peer broadcasts ρ_H and the number of unbounded support vectors they hold ($|LF_H|$) which does not reveal private information. Similar to the above proof, the probability of the view of message ρ_H received from P_H is $Pr\{VIEW = \rho_H\} = Pr\{\rho_H = \sum_{t \in LF_H} \mathbf{y}_{Ht} \mathbf{GL}_{Ht} + \sum_{t \in LF_{\Omega H}} y_{(\Omega H)t} \mathbf{GR}_{(\Omega H)t}\} = Pr\{\rho_H = \sum (gi' + gj') + \sum (ui + uj)\} = Pr\{\rho_H = \sum r\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = \rho_H\}$. In step 2a, every peer sends the indices of non support vectors and bounded support vectors to the subsequent peer, which does not reveal any private information. Step 2b is securely computed using SGMS, which has been proven secure. In step 2c, the values $\rho_1, \rho_2, \rho_3, \rho_4, e_1, e_2, e_3$ and e_4 are revealed. Similar to the above proof, the probability of the view of message e_1 received from P_I is $Pr\{VIEW = e_1\} = Pr\{e_1 = \mathbf{y}_{I\rho_1} \mathbf{GL}_{I\rho_1} - u_1\} = Pr\{u_1 = \mathbf{y}_{I\rho_1} \mathbf{GL}_{I\rho_1} - e_1\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = e_1\}$. The probability of the view of message ρ_1 received from P_I is $Pr\{VIEW = \rho_1\} = P\{\rho_1 = e_4 + u_1\} = Pr\{u_1 = \rho_1 - e_4\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r = \rho_1\}$. Proofs are similar for e_2, e_3 and e_4 and ρ_2, ρ_3 and ρ_4 . Hence Protocol 7 is secure.

Security Proof of Secure Prediction: In steps 1 to 4, all computations are performed on encrypted values hence they are secure. In step 2, the results of the secure scalar product with the support vectors are split into two shares and sent to the subsequent peer. When any peer only has a single (or small number of) support vectors, this secure share computation prevents peer P_T from probing its values since $Dec(\mathbf{y}_H)$ can be simulated. $Pr\{VIEW = Dec(\mathbf{y}_H)\} = P\{Dec(\mathbf{y}_H) = Dec(\mathbf{v}_H) + \mathbf{u}_{\Omega H}\} = Pr\{\mathbf{u}_{\Omega H} = Dec(\mathbf{y}_H) - Dec(\mathbf{v}_H)\} = 1/|\mathcal{F}| = Pr\{SIMULATOR\ r\}$, where r has a uniform distribution over a finite field $\mathcal{F}$.

Complexity

Computational complexity The initial RSVM construction takes $O(\ell_H^2 m_H)$, where m_H denotes the number of support vectors generated by RSVM of peer P_H , typically a small fraction of ℓ_H .

The computational complexity of the secure protocols will be computed based on the cost of encryption C_{enc} , decryption C_{dec} and exponential calculation C_{exp} which are the

dominating costs. In addition, s_p denote the security parameter for the secure protocols (e.g., encryption bits: 512, 1024, 2048, etc). The secure comparison used in SLMS, SGMS and Protocol 4 is based on Yao's protocol which has been shown to require n oblivious transfer protocols given n circuit inputs where each oblivious transfer protocols cost $O(C_{enc} + C_{exp} + C_{dec})$ [LP00]. Let $C_{cmp} = O(n(C_{enc} + C_{exp} + C_{dec}))$ denote the cost of evaluating $a + b \geq 0$, where $n = \{1539, 3075, 6146\}$ given $sp = \{512, 1024, 2048\}$. For the secure SVM construction, first, every pair of peers has to perform the secure scalar product of their data set once. Hence, cost of computing the secure scalar product for a pair of peer P_I and P_J is $O(m_I m_J (dC_{enc} + dC_{exp} + C_{dec}))$.

For each iteration of Protocol 5.1 (lines 3-7), the time complexity of the invoked protocols are as follows: SLMS incurs $O(m_H C_{cmp})$, SGMS incurs $O((\lg(N) + m_H) C_{cmp})$, Protocol 3 incurs $O((\lg N + m_H) C_{cmp})$, Protocol 4 incurs $O(C_{cmp})$, Protocol 5 incurs $O(C_{enc} + C_{exp} + C_{dec})$, Protocol 6 incurs $O(m_H (C_{enc} + C_{exp} + C_{dec}))$ and Protocol 7 incurs $O((\lg N + m_H) C_{cmp})$. Hence, the cost each iteration is can be simplified as $O((\lg N + m_H) C_{cmp})$. Though SMO is guaranteed to converge, the number of iterations required is unknown. However, empirical studies shows that the complexity of the SMO algorithm is much closer to $O(m^2)$ than $O(m^3)$ [Joa99]. The total number of levels to cascade given N peers is $\lg N - 1$, and the cost of a single cascade is simply the cost of constructing a single secure SVM.

For the secure SVM prediction, the computational complexity is $O(\ell_T \ell_{SV_H} (dC_{enc} + dC_{exp}) + 2C_{enc} + C_{dec})$, where ℓ_T and ℓ_{SV_H} denote the size of the test set, and the size of the support vectors on peer P_H respectively.

Communication cost The secure scalar product computation for the secure SVM construction incurs an $O(m d s_p)$ bits communication cost, while each iteration of the SVM construction incurs an $O((N \lg N + m) s_p^2)$ bits, given that the communication cost for secure comparison based on Yao's protocol is $O(s_p^2)$. The cost incurred for locating candidate groups is $O(\lg N)$.

The communication cost for secure prediction is $O((\ell_T d + N) s_p)$ bits and cost for locating all secure models in the network is $O(\lg N^2)$.

5.2 Experiments

This section presents experimental results to demonstrate the effects of the number of peers, cascading, and failure rate on classification accuracy.

5.2.1 Experimental Setup

All implementations are coded in C/C++. Except for local model construction which uses RSVM [LL03] implementation, all other models are constructed using secure SVM implementation. Only binary SVM is implemented but it is not difficult to extend it to

multiclass SVM using approaches such as one-versus-all or one-versus-one approaches. The cost parameter C is set to 1.

The proposed secure computations are based on libpaillier² which uses GNU Multiple Precision Arithmetic Library for large integer computation. As Paillier cryptosystem requires the use of integers for computations, all rational numbers are increased by a factor of 10 before any secure computation and decreased them by the same (or scaled up due to computations) factor after secure computation. It is easy to see that doing so results in loss of accuracy but with a sufficiently large factor, it can greatly minimize the effects. It is observed in the experiments that the numeric loss affects less than 0.1% of classification accuracy.

The binary Coverttype dataset created from the original dataset (normalized to $[0,1]$) with class 2 against other classes is used. All peers are assigned 1,000 data points and 50,000 data points are randomly selected as test set. Results are averaged over 10 runs for all experiments. The number of peers are varied from 2 to 512. Experiments were carried out in a simulated environment on single machine with two Intel Dual Core Xeon 3.0GHz processors, 4 GB RAM.

5.2.2 Classification Accuracy

Figure 5.3 illustrates how the number of peers and cascading affect accuracy. The accuracy of two approaches are plotted: One is an ensemble of secure models each consisting of 2 peers, and the other is a fully-cascaded model (single cascade). Observe that single cascade (which is the equivalent of a centralized SVM) achieves high accuracy regardless of the number of peers and becomes more accurate than the ensemble approach when the number of peers increases. This is in line with previous study [AGH⁺08], which shows that the increase in accuracy achieved by increasing the number of cascaded models is higher than that of increasing the size of the ensemble. Hence, experimental results justify the intuition to cascade groups for improving accuracy.

5.2.3 Complexity

Table 5.2 presents the total time taken by peers to construct a single secure SVM model (Init - Protocol 2 & Train - Protocols 3 to 7) and perform secure prediction using the proposed approach. Observed that the cost of Init increases linearly with the number of peers which is due to the linear increase in the training dataset size. Whereas cost of Train increases sublinearly but with an arbitrary large increase in the number of iterations. In general, SMO computes more iteration given a larger number of instances but it also depends on the complexity of the optimization problem, e.g., from 32 to 64 peers, there is only an increase in 451 iterations and from 128 to 256, there is a decrease in the number of iterations. For secure prediction, the time taken does not increase with

²<http://acsc.cs.utexas.edu/libpaillier>

Figure 5.3: Effects of the number of peers on accuracy.

Table 5.2: Total time taken (computation and communication) for varying number of peers.

No. peers	16	32	64	128	256
No. iterations	826	2649	3100	7978	7028
Init (secs)	9.310	18.74	37.29	74.63	149.3
Train/iter(mins)	2.082	2.273	2.558	3.090	3.766
Predict(secs)	0.105	0.106	0.098	0.103	0.126

the increase in the number of peers which is because predictions are done in parallel. However, the communication overhead increases linearly with the number of peers.

While the construction of secure SMO model with a large number of peers may seem infeasible and hence is not practical in the P2P environment, note that the cost needed to construct secure models with smaller number of peers (e.g., 2, 4, 8, etc) is much lower. Hence a large ensemble of secure models, each jointly constructed with a small number of peers, can still be deployed.

5.2.4 Number of Dimensions

As the cost of secure dot product used in Protocol 2 and secure prediction is affected by the number of dimensions ($\mathbf{x} = (x_1, \dots, x_d)^T$, cf. Chapter 1) of the data. The following experiments study the effects of the number of dimensions on cost. Table 5.3 shows the total time taken (includes both computation and communication) to compute a secure two party SMO and perform secure prediction under varying dimensions with the same training/testing set size. Init, Train and Predict indicate the cost of the initialization phase (cf. Protocol 2), a single iteration of the SMO optimization phase (cf. Protocols 3 to 7) and secure prediction of a single data point respectively. Note that Train is averaged over all iterations to remove the effects of the number of iterations on the cost. Observe that costs of Init and Predict increase linearly when the number of dimensions

Table 5.3: Total time taken (computation and communication) for varying number of dimensions.

Dimensions	10	50	100	500	1000
Init(secs)	2.114	6.155	11.22	51.68	102.9
Train/Iter(secs)	79.09	114.4	116.7	115.9	115.2
Predict(secs)	0.263	0.514	0.870	3.360	6.462

increases, whereas the cost for Train does not increase. This is expected as both the computations of both initialization and prediction are dependent on the number of attributes (both involve secure scalar product of the data). On the contrary, the cost of SMO optimization is not dependent on the number of dimensions but on the number of data points.

5.3 Summary

This chapter presented a cascading learning framework for privacy-preserving P2P classification in P2P networks. The proposed approach first constructs an ensemble of models from subsets of peers to achieve anytimeness, reduce synchronization, and increase tolerance to peer failures. Furthermore, the resultant models are incrementally cascaded to improve accuracy. To handle privacy issues, a novel privacy-preserving SVM protocol based on secure multiparty computations and the state-of-the-art SMO algorithm is proposed. The proposed protocol is geared towards improving the scalability and efficiency, by maximizing the distribution of computations among peers. The approach has been theoretically proved to be correct and privacy-preserving. Empirical studies also demonstrated the effectiveness of the proposed approach.

Chapter 6

Conclusions and Future Work

This thesis provided a comprehensive study on the existing work in distributed and P2P learning. In addition, the limitations of the approaches for distributed classification has been analysed, which serve as a guide for the design of our solution.

This thesis addressed the problem of learning in a basic P2P environment, which is the fundamental set of distributed classification issues present in all types of P2P networks. In addition, solutions have also been proposed for learning in P2P networks with imbalanced data distribution, asynchronous concept drift and privacy preservation. The solutions proposed for learning in a P2P network with imbalanced data distribution is AllCascade, RandBag, CEMPaR and PACE. Each of these approaches has different characteristics, hence different computation and communication cost with different accuracy tradeoffs. Therefore, depending on the P2P environment and the desired cost and accuracy trade-off, a suitable approach can be chosen.

To address the issue of asynchronous concept drift in P2P networks, RePCoDE and PINE have been proposed. While RePCoDE addresses asynchronous concept drift through the use of both reactive and proactive approaches based on learning from peers' models and concepts, PINE overcomes asynchronous concept drift by combining reactive adaptation via drift detection, and proactive handling of upcoming changes via early warning and adaptation across the peers. Experimental results have shown that both approaches are able to adapt to concept drifts faster and minimize the adverse effect on peers' whose concepts have not yet drifted.

In addition, PRICADE has been proposed to address the challenges of privacy preservation P2P classification. PRICADE overcomes the problem of privacy preservation through the use of secure multi-party computation and surpass the limitation of existing secure multi-party computation approaches with its multi-tier cascade ensemble framework to enable learning in the P2P environment. Empirical results have demonstrated its abilities to learn in the P2P environment and preserve data privacy.

The work done in this thesis fulfills our objective of developing a robust and efficient distributed classification framework that can construct accurate classification models, addressing challenges faced by P2P classification under all types of P2P environments for a diverse range of applications.

6.1 Future Work

In future, we would like to explore some of the following research problems to extend the applicability of P2P classification.

6.1.1 Multisource Learning and Data Fusion

In distributed classification, it is usually assumed that all the distributed data sources contain (training and testing) data with the same features. However, this may not always be true, as there are situations where the different data sources may contain different sets of overlapping or non-overlapping features. This is especially common for information collected at different geographical locations.

For example, in sensor networks, each sensor may be placed at different locations capturing different information that contribute to the same objective (e.g., geoscience [BBS01]). Businesses may collect different types of customers' information at different locations (e.g., physical stores, websites and service centers). In disease diagnosis, more than one medical test may be conducted on a patient for the confirmation of the diagnosis or other reasons. However, at the same time, not all medical tests may be conducted as some of the tests may be deemed unnecessary by the doctors or the medical tests are simply not available at that time. In the diagnosis, doctors will make use of the various medical tests conducted to decide whether the patient has contracted the disease and to determine the actions to be taken [PTP⁺08].

These above mentioned scenarios demonstrate how different data sources can have data with different feature subsets and this violates the implicit assumption and requirement for the traditional classification algorithm to work on all features. Moreover, different data sources could have the same attribute differently or vice versa. As such, attributes from different databases need to be matched correctly and data preprocessing have to be performed. This task of combining data from different sources for learning is commonly known as multisource learning or data fusion. Note that these issues could be resolved more easily with the centralization of data but in a distributed environment, especially where data are abundant and data sources are numerous (e.g., P2P networks), the cost of data centralization will be too costly. Hence, in recent years, an increasing amount of effort has been made to address this problem [BBS01, LP03, PP07, PTP⁺08]. However, many of these works have only been experimented with a small number of distributed sites, it remains unknown if existing algorithms are able to scale up for the P2P networks.

While much effort has been made to address the issue of learning from these arbitrary partitioned data, few have made any effort to examine the effects of prediction on these feature subsets. Since it has already been established that different data sources have different feature subsets, then it would not be strange that testing data of different sites also contains only the subset of all features used for the training. Thus, since the

classification model has combined all the features, then would it be possible to make an accurate prediction only based on the subset of features? How much accuracy would be lost if the testing data only have x number of features for prediction instead of all features?

As a part of our future work, we will investigate the learning of classification models in P2P networks from arbitrary partitioned data, which includes both the model construction and the prediction.

6.1.2 Fairness and Quality of Service (QoS)

Typically, peers in a P2P network voluntarily contribute their personal resources to the pool of shared resources, collectively improving the welfare of the P2P network. However, due to personal gains, some malicious peers may only want to use the resources of others and do not want to contribute their own resources. These kinds of acts are termed as free-riding and free-riding peers are named as free riders. Free riders affect the joint interest of the P2P network and degrade the performance of the P2P network. Free riding can create hot spots in a P2P network, which affects availability and speed of content retrieval. In addition, due to free-riding acts, a contributing peer may not get the service at the rate it expects and feels unfair, hence converting to a free rider. This vicious cycle can eventually lead to the collapse of the P2P network.

In addition to the fairness issues, quality of service is also of a concern. Note that every peer in the P2P network has his or her own data, which may or may not be similar. Classification accuracy of models built on such local data can vary widely. Given that there are two peers X and Y , and a P2P classification approach based on simple ensemble. Assuming that peer X 's classification model always predicts wrongly the test data of peer Y , then would it be better if peer X 's ensemble excludes the model of peer Y ? In addition, peers could have different bandwidth and computational power. If a peer has a time constraint for the prediction, then would it be better if prediction is made from a faster peer who is slightly less accurate as compared to a very slow peer who is slightly more accurate.

Hence, we intend to investigate as part of our future work the issues associated with the fairness and quality of service. Furthermore, we will study existing solutions such as incentives and reputation based approaches for other P2P applications like P2P file sharing and examine whether they can be applied for the P2P classification system.

6.1.3 Multi-Label Learning

In classic classification problems, each data instance is only associated with a single class label, regardless whether it is a binary or multiclass classification problem. However, in many real world classification problems, it is not uncommon that each data instance can be associated with more than one concepts (class labels) — multi-label classification

problem. For instance, a news article on funds raised in a charity soccer match could be associated with sports and social labels. Similarly, in disease diagnosis, a patient contracted with AIDS and dengue fever concurrently could be discovered in a set of tests.

There have been numerous studies on multi-label classification problem, but few are discussed on the platform of massively distributed environment. However, given the data associated with multi-label classification, it is not hard to see that these multi-label data are very common in the P2P environments. Hence, it would be natural that solutions for addressing multi-label classification in P2P networks be proposed. As part of our future work, we will study the existing multi-label classification solutions, examine how they can be mapped to the P2P environment, and propose new solutions to solve this problem.

6.1.4 Multi-Instance Learning

Unlike traditional classification where a single instance is mapped to a class label, multi-instance learning study the problem where a group of instance is mapped to a class label. In multi-instance learning, given a bag of instances, if all instances in the bag are negative, then the bag is labelled as negative; otherwise, if at least one of its instances is positive, then the bag is labelled as positive. The objective of multi-instance learning is to learn from the set of labelled bags to classify unseen bags correctly.

Multi-instance learning is not uncommon in many domains such as drug activity prediction, content-based image retrieval and classification and text categorization. In drug activity prediction, a molecule of interest may adopt many shapes, of which one may bind strongly to the target protein. As such, the molecule shapes represent the instances and the molecule represents the bag of instances. In content-based image retrieval and classification, an image may contain several different semantic concepts, of which one may be the retrieval target. However, often images are only provided with general captions that represent the whole image, making it hard to identify the different concepts within the image. This problem could be addressed with multi-instance learning to aid in the identification of the semantic concept. In text categorization, text document often contains many paragraphs and given a topic, it is not uncommon that some of the paragraphs may be classified under the topic while others do not. This problem can be mapped very well to the multi-instance learning providing fuzzy categorization of the text documents, which is more appropriate.

As a part of our future work, we would like to investigate the multi-instance learning on the P2P environment and propose efficient and accuracy solution for performing multi-instance classification.

References

- [AGH⁺08] H. H. Ang, V. Gopalkrishnan, S. C. H. Hoi, W. K. Ng, and A. Datta. Classification in P2P networks by bagging cascade RSVMs. In *DBISP2P*, pages 13–25, 2008.
- [AGHN08] H. H. Ang, V. Gopalkrishnan, S. C. Hoi, and W. K. Ng. Cascade RSVM in peer-to-peer networks. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, pages 55–70, 2008.
- [AGHN10] H. H. Ang, V. Gopalkrishnan, S. C. H. Hoi, and W. K. Ng. Adaptive ensemble classification in P2P networks. In *Proceedings of the 15th International Conference on Database Systems for Advanced Applications - Volume Part I*, pages 34–48, 2010.
- [AGHN13] H. H. Ang, V. Gopalkrishnan, S. C. H. Hoi, and W. K. Ng. Classification in P2P networks with cascade support vector machines. *ACM Transactions on Knowledge Discovery from Data*, 7(4):20:1–20:29, 2013.
- [AGNH09] H. H. Ang, V. Gopalkrishnan, W. K. Ng, and S. Hoi. Communication-efficient classification in P2P networks. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, pages 83–98, 2009.
- [AGNH10] H. H. Ang, V. Gopalkrishnan, W. K. Ng, and S. Hoi. On classifying drifting concepts in P2P networks. In *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, pages 24–39, 2010.
- [AGZ⁺13] H. H. Ang, V. Gopalkrishnan, I. Zliobaite, M. Pechenizkiy, and S. C. Hoi. Predictive handling of asynchronous concept drifts in distributed environments. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2343–2355, 2013.
- [AI06] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 459–468, 2006.

- [AS00] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 439–450, 2000.
- [AV07] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.
- [BBS01] G. J. Briem, J. A. Benediktsson, and J. R. Sveinsson. Boosting, bagging, and consensus based classification of multisource remote sensing data. In *Proceedings of the Second International Workshop on Multiple Classifier Systems*, pages 279–288, 2001.
- [BEK⁺98] S. Berchtold, B. Ertl, D. Keim, H.-P. Kriegel, and T. Seidl. Fast nearest neighbor search in high-dimensional space. In *Proceedings of 14th International Conference on Data Engineering*, pages 209–218, 1998.
- [BF01] J. P. Bradford and J. A. Fortes. Characterization and parallelization of decision-tree induction. *Journal of Parallel and Distributed Computing*, 61(3):322 – 349, 2001.
- [BG09] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis Advances in Intelligent Data Analysis VIII*, pages 249–260, 2009.
- [BHP⁺09] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 139–148, 2009.
- [BK04] J. Basak and R. Kothari. A classification paradigm for distributed vertically partitioned data. *Neural computation*, 16(7):1525–1544, 2004.
- [BKK⁺03] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, 2003.
- [BL13] K. Bache and M. Lichman. UCI machine learning repository, 2013.
- [BM08] S. Bach and M. Maloof. Paired learners for concept drift. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 23–32, 2008.
- [BOKSW05] A. Bar-Or, D. Keren, A. Schuster, and R. Wolff. Hierarchical decision tree induction in distributed genomic databases. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1138–1151, 2005.

- [Bre96] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [Bre99] L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1-2):85–103, 1999.
- [BWGK08] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta. Distributed decision-tree induction in peer-to-peer systems. *Statistical Analysis and Data Mining*, 1(2):85–103, 2008.
- [CBHK02] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
- [CEH01] N. Chawla, S. Eschrich, and L. Hall. Creating ensembles of classifiers. In *Proceedings of the IEEE International Conference on Data Mining*, pages 580–581, 2001.
- [CFPS99] P. Chan, W. Fan, A. Prodromidis, and S. Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems and their Applications*, 14(6):67–74, 1999.
- [CHB⁺02] N. V. Chawla, L. O. Hall, K. W. Bowyer, T. E. Moore, and W. P. Kegelmeyer. Distributed pasting of small votes. In *Proceedings of the Workshop on Multiple Classifier Systems*, pages 52–61, 2002.
- [CHBK04] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research*, 5:421–451, 2004.
- [CKL⁺07] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007.
- [CKO⁺06] L. Cao, S. Keerthi, C.-J. Ong, J. Zhang, U. Periyathamby, X. J. Fu, and H. Lee. Parallel sequential minimal optimization for the training of support vector machines. *IEEE Transactions on Neural Networks*, 17(4):1039–1049, 2006.
- [CL11] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [CMH⁺03] N. V. Chawla, T. E. Moore, L. O. Hall, K. W. Bowyer, W. P. Kegelmeyer, and C. Springer. Distributed learning with bagging-like performance. *Pattern Recognition Letters*, 24(1-3):455–471, January 2003.

- [CS93a] P. K. Chan and S. J. Stolfo. Experiments on multistrategy learning by meta-learning. In *Proceedings of the Second International Conference on Information and Knowledge Management*, pages 314–323, 1993.
- [CS⁺93b] P. K. Chan, S. J. Stolfo, et al. Toward parallel and distributed learning by meta-learning. In *Proceedings of the AAAI workshop in Knowledge Discovery in Databases*, pages 227–240, 1993.
- [CS97] P. Chan and S. Stolfo. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, 8(1):5–28, 1997.
- [CS98] P. K. Chan and S. J. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 164–168, 1998.
- [CSH04] D. Caragea, A. Silvescu, and V. Honavar. A framework for learning from distributed data using sufficient statistics and its application to learning decision trees. *International Journal of Hybrid Intelligent Systems*, 1(1-2):80–89, 2004.
- [CW02] V. Cho and B. Wüthrich. Distributed mining of classification rules. *Knowledge and Information Systems*, 4(1):1–30, 2002.
- [DBG⁺06] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed data mining in peer-to-peer networks. *Internet Computing, IEEE*, 10(4):18–26, July 2006.
- [DG04] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, pages 10–10, 2004.
- [DWJ⁺08] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling lsh for performance tuning. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 669–678, 2008.
- [FSZ99] W. Fan, S. J. Stolfo, and J. Zhang. The application of adaboost for distributed, scalable and on-line learning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 362–366, 1999.
- [GCB⁺05] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade SVM. In *Proceedings of the Advances in Neural Information Processing Systems 17*, pages 521–528. 2005.

- [Geu00] P. Geurts. Some enhancements of decision tree bagging. In *Proceedings of the Principles of Data Mining and Knowledge Discovery*, volume 1910, pages 136–147. 2000.
- [GKSS06] V. Gorodetskiy, O. Karsaev, V. Samoilov, and S. Serebryakov. Agent-based service-oriented intelligent P2P networks for distributed classification. In *Proceedings of the International Conference on Hybrid Information Technology*, volume 2, pages 224–233, 2006.
- [GLLM05] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *Proceedings of the 7th International Conference on Information Security and Cryptology*, pages 104–120, 2005.
- [GLOK04] C. Giannella, K. Liu, T. Olsen, and H. Kargupta. Communication efficient construction of decision trees over heterogeneously distributed data. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 67–74, 2004.
- [Gol04] O. Goldreich. *Foundations of cryptography*. Cambridge University Press, 2004.
- [GSaR09] J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 329–338, 2009.
- [HCL⁺08] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, pages 408–415, 2008.
- [HD04] S. Harms and J. Deogun. Sequential association rule mining with time lags. *Journal of Intelligent Information Systems*, 22(1):7–22, 2004.
- [HN08] S. Han and W. K. Ng. Preemptive measures against malicious party in privacy-preserving data mining. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 375–386, 2008.
- [HNWL10] S. Han, W. K. Ng, L. Wan, and V. C. S. Lee. Privacy-preserving gradient-descent methods. *IEEE Transactions on Knowledge and Data Engineering*, 22(6):884–899, 2010.

- [HSD01] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2001.
- [IGD11] E. Ikonomovska, J. Gama, and S. Džeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, 2011.
- [JA03] R. Jin and G. Agrawal. Communication and memory efficient parallel decision tree construction. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 119–129, 2003.
- [JKK98] M. Joshi, G. Karypis, and V. Kumar. Scalparc: a new scalable and efficient parallel classification algorithm for mining large datasets. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 573–579, 1998.
- [Joa99] T. Joachims. Making large-scale support vector machine learning practical. *Advances in kernel methods: support vector learning*, pages 169–184, 1999.
- [JX95] M. I. Jordan and L. Xu. Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, 8(9):1409 – 1431, 1995.
- [JYA05] R. Jin, G. Yang, and G. Agrawal. Shared memory parallelization of data mining algorithms: techniques, programming interface, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):71–89, 2005.
- [Kea98] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, November 1998.
- [KG02] S. S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46(1-3):351–360, 2002.
- [KM07] J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790, 2007.
- [KP01] H. Kargupta and B.-H. Park. Mining decision trees from data streams in a mobile environment. In *Proceedings of the IEEE International Conference on Data Mining*, pages 281–288, 2001.
- [KP04] H. Kargupta and B.-H. Park. A fourier spectrum-based approach to represent decision trees for mining data streams in mobile environments. *IEEE Transactions on Knowledge and Data Engineering*, 16(2):216–229, 2004.

- [KPD06] H. Kargupta, B.-H. Park, and H. Dutta. Orthogonal decision trees. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1028–1042, 2006.
- [KPHJ99] H. Kargupta, B.-H. Park, D. Hershberger, and E. Johnson. Collective data mining: A new perspective toward distributed data analysis. In *Proceedings of the Advances in Distributed and Parallel Knowledge Discovery*, pages 133–184, 1999.
- [KPJ⁺02] H.-C. Kim, S. Pang, H.-M. Je, D. Kim, and S.-Y. Bang. Support vector machine ensemble with bagging. In *Proceedings of the Pattern Recognition with Support Vector Machines*, volume 2388, pages 397–408. 2002.
- [Lem09] D. Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition*, 42(9):2169–2180, 2009.
- [Lin01] C.-J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, Nov 2001.
- [LKR06] K. Liu, H. Kargupta, and J. Ryan. Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):92–106, 2006.
- [LL03] K.-M. Lin and C.-J. Lin. A study on reduced support vector machines. *IEEE Transactions on Neural Networks*, 14(6):1449–1459, 2003.
- [LM01] Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, volume 1, pages 1–17, 2001.
- [LO01] A. Lazarevic and Z. Obradovic. The distributed boosting algorithm. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 311–316, 2001.
- [LO02] A. Lazarevic and Z. Obradovic. Boosting algorithms for parallel and distributed learning. *Distributed and Parallel Databases*, 11(2):203–229, 2002.
- [LP00] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 36–54, 2000.
- [LP03] M. Lewitt and R. Polikar. An ensemble approach for data fusion with learn++. In *Proceedings of the Multiple Classifier Systems*, volume 2709, pages 176–185, 2003.

- [LR05] F. Lozano and P. Rangel. Algorithms for parallel boosting. In *Proceedings of the Fourth International Conference on Machine Learning and Applications*, pages 368–373, 2005.
- [LSU07] S. Laxman, P. S. Sastry, and K. P. Unnikrishnan. A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 410–419, 2007.
- [LWC08] T.-Y. Lee, E. Wang, and A. Chen. Mining serial episode rules with time lags over multiple data streams. In *Proceedings of the Data Warehousing and Knowledge Discovery*, volume 5182, pages 227–240, 2008.
- [LWW04] B.-L. Lu, K.-A. Wang, and Y.-M. Wen. Comparison of parallel and cascade methods for training support vector machines on large-scale problems. In *Proceedings of the International Conference on Machine Learning and Cybernetics*, volume 5, pages 3056–3061, 2004.
- [XLS07] P. Luo, H. Xiong, K. Lü, and Z. Shi. Distributed classification in peer-to-peer networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 968–976, 2007.
- [LYWZ06] K. Lai, L. Yu, S. Wang, and L. Zhou. Neural network metalearning for credit scoring. In *Proceedings of the 2006 international conference on Intelligent Computing*, volume 4113, pages 403–408, 2006.
- [Mel97] G. Melli. Synthetic classification data sets (scds). Technical report, School of Computing Science, Simon Fraser University, Burnaby, British Columbia, 1997.
- [MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - a secure two-party computation system. In *Proceedings of the 13th Conference on USENIX Security Symposium*, pages 20–20, 2004.
- [MTIV97] H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
- [MWF08] O. L. Mangasarian, E. W. Wild, and G. M. Fung. Privacy-preserving classification of vertically partitioned data via random kernels. *ACM Transactions on Knowledge Discovery from Data*, 2(3):12:1–12:16, 2008.

- [MWY10] L. Minku, A. White, and X. Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742, May 2010.
- [Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the Advances in Cryptology EUROCRYPT 99*, volume 1592, pages 223–238, 1999.
- [PCS00] A. Prodromidis, P. Chan, and S. Stolfo. Meta-learning in distributed data mining systems: Issues and approaches. *Advances in distributed and parallel knowledge discovery*, 3, 2000.
- [Pla98] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Proceedings of the Advances in Kernel Methods - Support Vector Learning*, pages 185–208, 1998.
- [Pol06] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.
- [PP07] D. Parikh and R. Polikar. An ensemble-based incremental learning approach to data fusion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):437–450, 2007.
- [PS98] A. Prodromidis and S. J. Stolfo. Pruning meta-classifiers in a distributed data mining system. In *Proceedings of the First National Conference on New Information Technologies*, pages 151–160, 1998.
- [PSC99] A. L. Prodromidis, S. J. Stolfo, and P. K. Chan. Effective and efficient pruning of meta-classifiers in a distributed data mining system. Technical report, Columbia University, 1999.
- [PTP⁺08] R. Polikar, A. Topalis, D. Parikh, D. Green, J. Frymiare, J. Kounios, and C. M. Clark. An ensemble based data fusion approach for early diagnosis of alzheimers disease. *Information Fusion*, 9(1):83 – 95, 2008.
- [SAM96] J. C. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 544–555, 1996.
- [SK01] W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 377–382, 2001.

- [SM08] D. Skillicorn and S. McConnell. Distributed prediction from vertically partitioned data. *Journal of Parallel and Distributed Computing*, 68(1):16 – 36, 2008.
- [SMK⁺01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160, 2001.
- [SPT⁺97] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan. Jam: Java agents for meta-learning over distributed databases. In *Proceeding of Third International Conference on Knowledge Discovery and Data Mining*, pages 74–81, 1997.
- [SS06] S. Siersdorfer and S. Sizov. Automatic document organization in a P2P environment. In *Proceedings of the Advances in Information Retrieval*, volume 3936, pages 265–276, 2006.
- [Sta90] W. Stadje. The collector’s problem with group drawings. *Advances in applied probability*, 22(4):866–882, 1990.
- [TAV04] G. Tsoumakas, L. Angelis, and I. Vlahavas. Clustering classifiers for knowledge discovery from physically distributed databases. *Data & Knowledge Engineering*, 49(3):223 – 242, 2004.
- [TE03] A. Tveit and H. Engum. Parallelization of the incremental proximal support vector machine classifier using a heap-based tree topology. In *Proceedings of the Workshop on Parallel and Distributed Computing for Machine Learning (in conjunction with ECML’2003 and PKDD’2003)*. Citeseer, 2003.
- [TPCP08] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. Dynamic integration of classifiers for handling concept drift. *Information Fusion*, 9(1):56 – 68, 2008.
- [TV02] G. Tsoumakas and I. P. Vlahavas. Effective stacking of distributed classifiers. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 340–344, 2002.
- [VSN05] T. Volkmer, J. R. Smith, and A. P. Natsev. A web-based system for collaborative annotation of large image and video collections: An evaluation and user study. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, pages 892–901, 2005.

- [WDKS07] Z. Wang, S. K. Das, M. Kumar, and H. Shen. An efficient update propagation algorithm for P2P systems. *Computer Communications*, 30(5):1106 – 1115, 2007.
- [WF05] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition edition, 2005.
- [WP01] G. M. Weiss and F. Provost. The effect of class distribution on classifier learning: An empirical study. Technical report, Department of Computer Science, Rutgers University, 2001.
- [Yao86] A. C.-C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.
- [YJV06] H. Yu, X. Jiang, and J. Vaidya. Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 603–610, 2006.
- [YVJ06] H. Yu, J. Vaidya, and X. Jiang. Privacy-preserving SVM classification on vertically partitioned data. In *Proceedings of the Advances in Knowledge Discovery and Data Mining*, volume 3918, pages 647–656, 2006.
- [YWZ06] Y. Yang, X. Wu, and X. Zhu. Mining in anticipation for concept change: Proactive-reactive prediction in data streams. *Data Mining and Knowledge Discovery*, 13(3):261–289, 2006.
- [YZ09] X. Yi and Y. Zhang. Privacy-preserving naive bayes classification on distributed data via semi-trusted mixers. *Information Systems*, 34(3):371 – 380, 2009.
- [Zak01] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.
- [ZHA99] M. J. Zaki, C.-T. Ho, and R. Agrawal. Parallel classification for data mining on shared-memory multiprocessors. In *Proceedings of the 15th International Conference on Data Engineering*, pages 198–205, 1999.
- [ZLY05] J.-P. Zhang, Z.-W. Li, and J. Yang. A parallel SVM training algorithm on large-scale classification problems. In *Proceedings of the 2005 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1637–1641, 2005.

- [ZSZ06] L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7:1467–1492, 2006.
- [ZWB⁺08] K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, H. Cui, and E. Y. Chang. Parallelizing support vector machines on distributed computers. In *Proceedings of the Advances in Neural Information Processing Systems 20*, pages 257–264. 2008.