PhD Thesis

Multi-Class Semi-Supervised and Online Boosting

Amir Saffari

Graz, May 2010

Thesis Supervisors:
Prof. Horst Bischof
Prof. Bernt Schiele

Institute for Computer Graphics and Vision
Graz University of Technology, Austria

To Aida, Mahvash, Jahangir, and Kaveh.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Acknowledgments

First, I would like to express my gratitude to my supervisor, Horst Bischof. Horst was not only my advisor, but also a great friend, who had a profound influence on my research. Under his supervision, I enjoyed having complete freedom on choosing my research directions, which also can be seen from the wide spectrum of topics I got involved during my studies. Thanks for the support, guidance, and simply everything. I would also like to thank my second advisor, Bernt Schiele for his comments on the thesis.

Special thanks to my metal buddy and my primary collaborator, Christian Leistner. I have had a great time sharing the office with you and it was a pleasure discussing, working, listening to music, playing football, and having fun together. Rock on dude!!!

I was fortunate enough to be in an institute where I had a chance to collaborate with many great colleagues. In particular, I would like to thank Isabelle Guyon for guiding me through many machine learning problems and challenges. It was a great pleasure working with you Thomas Pock, your perspective on optimization techniques was invaluable. I would like to thank the BOB meister, Martin Godec, for all his supports. Special thanks to Peter Roth, the Boss, for his helpful comments on many of my scientific research. Nguyen Thuy, thanks for all the help and also sharing the office with me as well. I had the pleasure of working with a GPU guru, Jakob Santner, Prost!!!

I was quite lucky to share and work interesting ideas with Gokhan Bakir, Joachim Buhmann, Gavin Cawley, Gideon Dror, Hugo Escalante, Helmut Grabner, Andreas Juffinger, Inayatullah Khan, Michael Pfeiffer, and Bernhard Zeisl.

Most importantly, I want to thank my family. Aida, you are the greatest love of my life. Thanks for all the support, happiness, and love all along the way. I cannot imagine this work, or anything else, being done without you. My parents, Mahvash and Jahangir, you are always the greatest support and friends I ever had, thanks for everything. Kaveh, you know I cannot describe in words how much I appreciate your help, support, and your friendship. G^3 forever.

This thesis was created at the Institute for Computer Graphics and Vision at Graz University of Technology between 2006 and 2010. During this time, the work was supported by the the Austrian Joint Research Project Cognitive Vision under projects S9103-N04 and S9104-N04, the EVis (813399) and Outlier (820923) FFG projects under the FIT-program and the Austrian Science Fund (FWF) under the doctoral program Confluence of Vision and Graphics W1209.

Abstract

In this thesis, we develop multi-class boosting algorithms for supervised, semi-supervised, and online learning problems. First, we present our supervised algorithm which is based on maximizing the true multi-class classification margin. This algorithm is versatile enough to use many different loss functions. As a result, the proposed multi-class boosting algorithm can be used in many different applications and circumstances where the desirable behavior can be tuned by choosing the proper loss function.

Based on the concept of learning from prior knowledge, we build a multi-class semi-supervised boosting algorithm which is able to use unlabeled data to improve its performance. We show that our algorithm can be applied to large-scale problems with many unlabeled samples. This algorithm is flexible enough to operate on a wide-range of semi-supervised learning problems by incorporating both the cluster and manifold assumptions. We take the proposed semi-supervised boosting algorithm further and show that it can be used for learning from multiple views of the data. In this approach, different views combine their beliefs regarding the output label of an unlabeled sample into a set of prior knowledge and train each other using the same principles introduced for the semi-supervised boosting model.

We proceed to develop an online multi-class learning algorithm which uses online convex programming tools to solve linear programs in the context of the boosting. The resulting algorithm can use any other online learner as its base functions and compared to other online learning algorithms, its performance is considerably higher.

We apply the proposed methods to a wide range of applications, such as pattern recognition, object category recognition, and visual object tracking. Our extensive set of evaluations demonstrates the competitiveness of the proposed algorithms with respect to the state-of-the-art methods.

Kurzfassung

Das Ziel dieser Dissertation ist die Entwicklung von Mehrklassen-Boosting Algorithmen für berwachte, halb-überwachte und online Lernverfahren. Hierführ presentieren wir zuerst einen überwachten Lernalgorithmus, der auf der Maximierung der echten Mehr-klassen Differenzspanne (Margin) besteht. Dieser Algorithmus ist vielseitig einsetzbar, da er beliebige jedoch typische Lernfunktionen verwenden kann. Daher kann der von uns vorgeschlagene Mehrklassen Algorithmus in vielen Anwendungen zum Einsatz kommen, wobei für jede Anwendung eine dementsprechend angepasste Lernfunktion verwendet werden kann.

Weiters zeigen wir wie Lernalgorithmen vom Einsatz von Vorwissen profitieren können und presentieren, basierend auf diesem Konzept, einen Mehrklassen Algorithmus, der sowohl von gelabelten als auch ungelabelten Daten lernen kann. Der von uns vorgeschlagene Algorithmus ist flexibel einsetzbar und eignet sich daher für alle möglichen Varianten von halb-überwachtem Lernen, wo für das jeweilige Problem unterschiedliches Vorwissen in den Ansatz miteingebracht werden kann, zum Beispiel, in Form von Cluster oder Manifold Annahmen. Ausserdem zeigen wir wie man mit unserem Algorithmus auch von unterschiedlichen Blickwinkeln auf das Lernproblem profieren kann, indem man für den jeweiligen Blickwinkel einen eigenen Klassifizierer trainiert, und die unterschiedlichen Klassifizierer sich dann gegenseitig auf ungelabelten Daten trainieren. Im speziellen enkodieren wir die Vorhersagen der unterschiedlichen Klassifizierer in eine Menge von Vorwissen. Dieses Vorwissen wird dann verwendet um mit dem zuvor vorgestelltem Prinzip zu lernen.

Danach presentieren wir einen online Mehrklassen Algorithmus. Die Lernmethode verbindet das Prinzip der online konvexen Programmierung, die üblicherweise für das lösen linerarer Programme verwendet wird, und Boosting. Der resultierende Algorithmus ist in der Lage beliebige online schwache Klassifikoren zu verwenden. Wir vergleichen den von uns vorgestellten Algorithmus mit anderen state-of-the-art Methoden und zeigen, dass wir signifikant bessere Ergebnisse erreichen.

Wir wenden die von uns vorgeschlagenen Methoden auf eine weite Palette von Applikationen an, zum Beispiel, Mustererkennung, Objektkategorisierung und Objektverfolgung. In all den von uns durchgeführten Experimenten zeigen wir, dass wir in der Lage sind zumindest state-of-the-art Ergebnisse zu erzielen und vielfach bessere Ergebnisse liefern als andere Methoden.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Learning is the ability to gather, acquire, or infer new knowledge, skills, and behaviors through a set of information processing steps. This ability is possessed by some animals and is most pronounced in humans[1], where learning happens through various autonomous, educational, or self-motivated processes. Learning is the necessary tool for humans to develop certain skills and acquire intelligence, which then affects how we plan, reason, act, and more importantly, think. Learning is a continuous process which happens over the course of a human life span. It is often thought to be a process which involves various high-level cognitive tasks, such as collecting the data and facts (*e.g.* through observing the world), interpretation, abstraction, generalization, imprinting, imitation, or association of the data to form certain useful intelligence.

In this thesis, we focus on developing *Machine Learning* algorithms. Machine learning is a field of computer science which is concerned with designing and developing computer algorithms and systems that allow machines to *learn* from the empirical data collected through a set of sensors. This process is usually task-oriented and can be used to solve real-world problems. Therefore, we can think of machine learning as a scientific discipline with the goal of imitating the learning abilities of human and providing the machines with tools to adapt to their environment and abstract high-level information into intelligence.

Humans can learn from a wide spectrum of information: some can be seen as *supervised* data where the learning involves a teacher which can provide additional information about the facts and the data we observe, and can correct the mistakes we make. This kind of learning constitutes our educational system, and depending on the age or the task, can be seen as one of the most effective kinds of learning. The majority of machine learning techniques rely on such a supervised system, where the learning data is usually

[1]In this thesis, we mainly refer to humans when we draw an example. However, the arguments can be used for other higher-level animals and mammals as well.

accompanied by a teacher or supervisor. We are also able to learn without any supervisor in an autonomous manner. From early childhood, we can differentiate between various concepts and use the sensory information without supervision. Equivalently, in machine learning we have algorithms which also focus on learning from the data without a need for a supervisor.

There are also intermediate levels of supervision, where we usually receive a small amount of information from a teacher and then can use all the rest of the data to improve our knowledge with respect to the concept the teacher was referring to. This kind of learning can be referred to as *semi-supervised* where some data is conveyed with the supervisory signal and some others are simply the plain sensory data. Machine learning community has developed algorithms for this kind of learning as well.

We, as humans, possess the ability to learn from a wide range of data, continuously and autonomously. Our learning process never ceases over our life-time. The ability to continuously learn and update our intelligence allows us to acquire new knowledge or refine and improve the ones we have collected before. We can easily incorporate, either consciously or subconsciously, the new facts and data into what we know already about the world. *Online Learning* is a field of machine learning which is concerned with developing algorithms for continuous learning from incoming data. It can be used to imitate the ability of humans to learn and acquire new knowledge through the course of analyzing the data over time.

In this thesis, we have a special focus on the online and semi-supervised machine learning topics, and their applications to various computer vision tasks. Therefore, in the following sections we make an informal introduction to these topics, where more formal description and discussion is presented in later chapters.

1.1 Machine Learning

In humans, the learning happens via specific processes and changes in the central nervous system. The learning act usually involves a set of observations made about a phenomenon and a process for transforming those information into knowledge.

Drawing a parallel between the human and machines, the artificial systems learn via alterations they make into their internal representations, data, models, and/or structures (both at the level of software or hardware). These machines are built in order to solve certain tasks and aim of the learning is to improve their performance over those tasks by learning from the environment, teachers, or their own experiences in achieving certain goals. The machine learning techniques can be helpful for tasks where the complexity of the problem in terms of the varieties of inputs and outputs are too high to be explicitly encoded by a programmer. Therefore, the machine learning algorithms can be thought of programs that configure the main program responsible for accomplishing a given goal via experimentation with inputs and outputs.

1.1.1 Supervised Learning

Machine learning methods are usually categorized by their requirements over the type of data they can use. Fully supervised algorithms always require a teacher to reveal the true interpretation of the data (such interpretation is usually referred to the *label* of a data). The goal for these algorithms is to learn a mapping from the input data to the outputs given by the teacher. Depending on the nature of the output, these algorithms can be split further into *classification* problems where the output space can be represented in discrete form, or the *regression* problems where the output space is continuous.

1.1.2 Unsupervised Learning

Unsupervised learning methods, on the other hand, work solely over the data samples themselves and do not require a teacher. They usually try to identify similar patterns in their inputs and identify clusters of data samples. These algorithms can be used to recognize the structure of data and how they are organized.

1.1.3 Semi-Supervised Learning

In many applications, the requirement for a teacher in the fully supervised learning methods can not be fulfilled. For example, it might be that identifying the true interpretation of the data requires special measurements or devices which are costly or time consuming. Or it could be that such a task is tedious for a human to be performed repeatedly. However, collecting a large set of unlabeled data samples is usually effortless, or in many applications cheap to obtain. Therefore, it is desirable to have methods which require only a minimal supervision and are able to learn from both labeled and unlabeled data. *Semi-supervised learning* deals with developing algorithms which can use both labeled samples and unlabeled samples. These algorithms can mix the supervised and unsupervised learning principles to accomplish this task.

1.1.4 Reinforcement Learning

Reinforcement learning deals with designing algorithms which perform actions based on their inputs with a goal to maximize a reward in long run. These methods usually try to identify a set of policies for taking certain actions based on the sensory data they collect and the feedback signal from the environment, which indicates the availability of a reward (or a punishment which is a negative reward).

1.1.5 Offline and Online Learning

It is possible to categorize machine learning algorithms based on how the learning takes place over time. *Offline* algorithms require all the training data to be available from the beginning of learning process. These kind of algorithms try to produce results which are consistent with all the collected data samples. On the other hand, in *online* learning the training data arrives sequentially over time and additionally, it is required that the learner is able to produce results at any time during its learning process. Online learning is more consistent with how the human learns and provides the machine with the ability to learn continuously and adapt all the time to its inputs.

1.1.6 Meta Learning and Ensemble Methods

Meta learning algorithms are special kinds of learning methods which are able to incorporate other learners as their base functions and produce a model that is a combination of them. This model often has better performance compared to each of the individual constructing base learners. Since these algorithms work on top of other learning methods, they are referred to as meta learners as well as *ensemble methods*. In this thesis, we mainly work with special kinds of ensemble methods, known as *boosting* models. Boosting, as its name suggest, is a method which is able to boost the performance of base learning algorithms by repeatedly training it over modified versions of the input data.

1.1.7 Binary and Multi-Class Classification

For classification problems, if the outputs are described by only two binary decisions (such as yes or no answers), then we call it a *binary classification* problem. Similarly, if the output space consists of more than two outcomes, the problem is called a *multi-class classification* task. It is clear that binary problems are a specific kind of multi-class problems, and hence, multi-class methods can be easily used for binary problems. However, the reverse is not true and in order to use binary algorithms for a multi-class problem, one has to convert the original problem into a set of equivalent binary tasks.

1.2 Computer Vision

Computer Vision is a scientific field concerned with extracting information from visual data (*e.g.*, images and videos), and subsequently, enabling the machines to observe and *see* the world. Though vision and the ability to comprehend the surrounding visual environment is trivial for humans, understanding the images and videos for machines is still a difficult task, mainly due to the high variability and dynamics of the visual data. Because of these challenges, learning plays a central role in many computer vision applications.

1.3 Main Contributions

1.3.1 Motivations

In this dissertation, we address the development of boosting-based semi-supervised and online learning algorithms for multi-class classification tasks. Many interesting problems in computer vision and other domains where machine learning is applied, are multi-class by nature. While there are methods developed to use a binary classifier for multi-class tasks, they usually have their own limitations. Therefore, in this thesis we have a special focus on developing inherently multi-class algorithms. Due to the success of boosting based methods, specially in computer vision applications, we concentrate on developing boosting methods.

In many classification problems, a large amount of unlabeled data is available, while it is costly to obtain labeled data. This is especially true for applications in computer vision such as object recognition and categorization. Current supervised approaches obtain high recognition rates if sufficient labeled training data is available. However, for most practical problems there is simply not enough labeled data available, whereas hand-labeling is tedious and expensive, and in some cases not even feasible. Nowadays, the Internet offers a huge amount of data in form of unlabeled data (or labeled with a high degree of uncertainty), and learning from Internet is becoming more and more widespread in computer vision. Therefore, investigating in methods which are efficient enough to use a large amount of unlabeled samples is becoming more and more important. Hence, one of our goals in designing semi-supervised boosting algorithms is the efficiency of the method and its scalability to large scale problems.

Furthermore, in many computer vision applications, the images and videos can be represented with several kind of features, and learning from multiple representations has been shown to be effective. Therefore, we set a goal to design a semi-supervised method which can be used easily for multi-view learning problems.

Online learning is another interesting research area with a huge application domain in computer vision field. Many recent research in the online learning domain only focus on binary problems. Because of the existence of many multi-class problems in computer vision, in this work, we also conduct research in the design of robust and powerful online multi-class boosting techniques.

1.3.2 Contributions

In this thesis, first, we develop a multi-class boosting algorithm based on the true multi-class classification margin. This algorithm is based on the functional gradient descent approach to boosting and is versatile enough to use many different loss functions. A loss function defines how much penalty an algorithm receives based on the decision it makes on an input data. Therefore, different loss functions produce different learning behaviors. As a result, the proposed multi-class boosting algorithm can be used in many different applications and circumstances where the desirable behavior can be tuned by choosing the proper loss function.

In many applications, there exists a prior knowledge about the problem at hand, which can be represented in terms of prior probabilities. Based on the concept of learning from prior knowledge, we build a multi-class semi-supervised boosting algorithm which is able to use unlabeled data to improve its performance. We show that our algorithm can be applied to large-scale problems with many unlabeled samples. This algorithm is flexible enough to operate on a wide-range of semi-supervised learning problems by incorporating both the cluster and manifold assumptions. We propose the *Jensen-Shannon* loss function to be used for measuring the penalty for the unlabeled samples and show that is more robust to noise than other choices for the unlabeled loss function.

We take the proposed semi-supervised boosting algorithm further and show that it can be used for learning from *multiple views* of data. In this approach, different views combine their beliefs regarding the output label of an unlabeled sample into a set of prior knowledge and train each other using the same principles introduced for the semi-supervised boosting model.

Boosting can be seen as a linear classifier working in the feature space built by its base learners. Therefore, there are boosting algorithms which use linear programming techniques to solve the classification problem. These algorithms are one of the most powerful boosting algorithms available. We proceed to develop an online multi-class learning algorithm which uses online convex programming tools to solve linear programs in the context of these algorithms. The resulting algorithm can use any other online learner as its base functions and its performance compared to other online learning algorithms is considerably higher. Since this algorithm is multi-class, we extend the tracking-by-detection type of algorithms to incorporate *virtual classes*, which are a set of background regions which lie close to the decision boundary of the target object.

Following the open source philosophy, the majority of the code that I have developed for experimenting the algorithms proposed within this dissertation, is released as open source software. This provides the community with the possibility of experimenting with these algorithms freely and giving them a unique chance to develop them further.

In order to make the thesis coherent, I omit from this manuscript some works that I have developed or had a chance to be a part of its research team. For example, I did research in unsupervised boosting based clustering algorithms [82, 83], an online version of the Random Forests algorithm [88], semi-supervised extension of Random Forests [58], online robust boosting and the online semi-supervised multiple-instance boosting [57, 109, 56], visual object tracking [81, 89, 38], learning for GPU-based segmentation [90], Internet-based object category recognition algorithm by learning from visual and textual features [54], predicting text relevance from eye movements [73], and model selection [47]. I was also a co-organizer of some machine learning challenges [44, 45, 46]. The list of all the publications I did during my PhD studies can be found in Appendix C.

1.4 Structure of the Thesis

In Chapter 2, we introduce the preliminaries and the works related to the topics covered in this thesis. We start discussing the multi-class boosting principles in Chapter 3. Chapter 4 encompasses the semi-supervised learning algorithms, while in Chapter 5 we introduce the online multi-class boosting algorithm. We conclude the thesis with a summary and some notes on future research directions regarding the methods we introduced in this thesis in Chapter 6.

Chapter 2

Preliminaries and Related Work

In this chapter, we introduce the related concepts to the main material covered within this thesis. We first introduce the notations and then proceed to discuss the ensemble and boosting methods. We continue with describing what semi-supervised learning is and what kind of assumptions they make regarding the relation of the data and labels. Finally, we discuss the online learning and online boosting approaches.

2.1 Notations

Throughout this thesis, we use lower case letters (*e.g.* x, λ) for scalar variables and vectors are denoted by bold face (*e.g.* $\mathbf{x}$, $\boldsymbol{\lambda}$). When working with vectors and multi-valued functions, the k-th element of it is represented by a subscript (*e.g.* $\mathbf{x}_k$). Sets are represented by calligraphy letters (*e.g.* $\mathcal{X}$). We use $\|\mathbf{x}\|_p$ to denote the p norm of a vector. Also $\nabla f(\mathbf{x})$ represents the gradient vector of a function at point $\mathbf{x}$. $\mathbb{R}$ denotes the real numbers.

We use $\mathcal{X}_l = \{(\mathbf{x}, y) | \mathbf{x} \in \mathbb{R}^d, y \in \{1, \dots, K\}\}$ for a set of labeled samples, where each sample $\mathbf{x}$ comes from a d dimensional feature space, and y is its label for a K class classification problem. In order to be consistent with the common notation in machine learning, when addressing the binary problems, the labels are drawn from $y \in \{-1, +1\}$ set. $\mathcal{X}_u = \{\mathbf{x} | \mathbf{x} \in \mathbb{R}^d\}$ represents a set of unlabeled samples. When working in online settings, we order the samples as $\cdots, (\mathbf{x}_t, y_t), \cdots$ where t denotes the time.

We use

$$c(\mathbf{x}) : \mathbb{R}^d \to \mathcal{Y} = \{1, \dots, K\} \tag{2.1}$$

to represent a multi-class classifier. For multi-class problems ($K > 2$), it is common that the classifier $c(\mathbf{x})$ is constructed from another learner $f(\mathbf{x})$

$$f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^K \tag{2.2}$$

by using the following decision rule

$$c(\mathbf{x}) = \arg\max_{k \in \mathcal{Y}} f_k(\mathbf{x}), \tag{2.3}$$

where $f(\mathbf{x}) = [f_1(\mathbf{x}), \cdots, f_K(\mathbf{x})]^T$ is a multi-valued mapping and $f_k(\mathbf{x})$ represents the confidence of the learner for classifying $\mathbf{x}$ to k-th class.

2.2 Ensemble Methods and Boosting

For a classification problem, a classifier is a function $c(\mathbf{x})$ mapping the input features $\mathbf{x}$ to output labels y. Since designing such a function by hand is very difficult and tedious for a human (because of the complexity of the problem), we use learning algorithms to construct such a function based on the training data we supply to it. Many learning algorithms require an optimization step to be performed. Different learning algorithms can be designed based on the desirable characteristics of the classifier and the underlying optimization step. In the following, we focus on ensemble methods and their variants.

Ensemble method is a generic learning mechanism which accepts as its input another learning algorithm. Assume $\mathcal{E} = \{g_1(\mathbf{x}), \cdots, g_M(\mathbf{x})\}$ is a collection of classifiers in an ensemble. These classifiers are usually called as *base learners*, *ensemble members*, or because of their simplicity as *weak learners*. An ensemble method then generates a weighted combination of these classifiers as its final output by

$$f(\mathbf{x}) = \sum_{m=1} w_m g_m(\mathbf{x}), \tag{2.4}$$

where w_m is the weight of m-th base learner. This can also be seen as the *weighted voting* as the m-th ensemble member votes by weight w_m for the final decision.

The intuition behind the combination is that the final output could perform better than each of the individual ensemble members. For this combination to be successful, the ensemble members should be diverse: if all of them are the same functions, then the combination will not help and improve. Therefore, ensemble methods differ in how they introduce the diversity and how they find the proper combination weights. Dietterich [28] provides three reasons why an ensemble of diverse classifiers should work better than individual ones:

Statistical — Let us assume that there is a hidden but perfect classifier for the task we are addressing. The goal of learning is of course to uncover this perfect classifier. Therefore, we can view a learning algorithm as a search method in the space of classifiers to find the best possible one. However, due to the limited amount of training data, it is possible that this search finds many equivalently good solutions which have the same accuracy over the training set. In such a case, an ensemble method averages the output of these classifiers and hence reduces the risk of choosing a sub-optimal classifier. Therefore, the averaging helps to get a closer solution to the perfect classifier.

Computational	Many algorithms use various optimization or search strategies to identify the optimal classifier. Due to local minima (if the search objective is non-convex), randomness, or numerical issues it is possible that the learning algorithm reaches a solution which is not the optimal one. Therefore, by averaging the results of multiple runs of the same algorithm, one can get closer to the best possible solution.
Representational	Due to the hidden nature of the perfect classifier, it is not ensured that the class of functions where we search for the optimal solution can represent the perfect classifier. The weighted averaging in fact expands the class of functions a classifier can represent and hence, provides a richer function space which might be suitable to find solutions closer to the perfect classifier.

Early versions of ensemble methods, such as *Composite Classifier Systems* [26], used different class of base learners to introduce this diversity. However, the majority of ensemble learning algorithms focus on building a model based on only a single class of classifiers as their base learners.

In order to introduce the diversity and reduce the variance of the classifiers, Breiman introduced the concept of learning by *bootstrap aggregating* which became known as *Bagging* [14]. In this approach, each base learner is trained repeatedly using a bootstrapped replica of the training set. Since each base learner sees a different training set, they become diverse in nature. Later on, Brieman introduced the *Random Forests* (RFs) [15] which uses decision trees as their base learners. RFs not only use bagging, but also during the training of each decision tree, each decision node only works on a random subset of all the features. This results in the increased diversity among the *random* decision trees of the ensemble (forest). This class of ensemble learning algorithms are *parallel* by their nature as the training and testing of each of the base learners is independent from others. Therefore, by utilizing parallel programming and multi-core CPUs (GPUs), it is possible to achieve very efficient learning machines.

Another class of ensemble methods work by assigning weights to training samples. This way they can introduce diversity by training each base learner on an altered version of the training set. Schapire [91] introduced the *Boosting* algorithms. They use a combination of many base classifiers which are very *weak* on their own (*i.e.* their accuracy is only slightly better than random guessing), and boosting in fact *boosts* their performance. Boosting is a *serial* ensemble learning method, where the base learners are trained one after each other. The major intuition behind boosting-like algorithms is that one can provide the most informative training set for each of the base learners by looking into what previously trained learners are able to achieve. Many boosting algorithms modify the weight for each training sample based on the error the previous weak learners make, by increasing the weight if there is a classification mistake for a sample. Likewise, they reduce the weight of a sample if the current ensemble is able to correctly classify a sample. By interpreting the weights as importance, at each boosting iteration the base learners focus on solving harder examples. Many boosting algorithms exists, such as AdaBoost [32], LogitBoost and GentleBoost [35], and they mainly differ on how they modify the weights of the samples and how they determine the weights of the base learners (*i.e.* w_m).

There exists a few boosting methods which bridges the gap between Support Vector Machines (SVMs) and boosting-like algorithms by solving an optimization problem which uses objective functions similar to those found in SVMs literature. LPBoost is a boosting algorithm where in each iteration of boosting, a linear program is solved in order to obtain the best weights for the samples. Then a search is conducted to find the best weak learner according to these weights. Theoretically, it has been shown that LPBoost [27] and regularized versions of it, such as SoftBoost [103] and Entropy Regularized LPBoost [104] can solve the classification tasks in smaller boosting iterations and experimentally are often superior to other boosting methods.

Overall, ensemble methods are considered to be one of the most successful machine learning techniques. They have been shown to be superior to many other algorithms on various large-scale and very high dimensional problems [17]. Due to their simple nature, these algorithms enjoy computational efficiencies and are used in many real world applications.

2.2.1 Multi-Class Boosting

In this thesis, we work with multi-class classification problems. Many machine learning algorithms are designed to solve binary classification problems, and there exists generic techniques to apply such methods to multi-class problems. These techniques often work by decomposing the original multi-class task into several binary problems and apply the binary learning method to each of these sub-problems. Their results are collected and the decisions are made in a post-processing step.

Typical approaches are the 1-vs-all, 1-vs-1, and error correcting output codes [2]. However, these algorithms often introduce additional problems. First, by considering only the binary sub-problems, the algorithms often fail to completely capture the true structures and relations between the classes in the feature space. In online learning tasks, this problem is even more severe because the learner has access only to a limited amount of data. For example, since each of the binary classifiers do not have access to the information available in other classifiers, their responses or confidences might not be comparable. This problem is usually referred to as the *Calibration* problem. Another problem with these methods is that one has to repeat the same algorithm many times, *e.g.*, in 1-vs-all there will be K classifiers trained where K is the number of classes. If the underlying algorithm has a high computational complexity, these repetitions are not an suitable option for solving problems with many classes and samples. Also, in the case of the 1-vs-all approach, one introduces unbalanced datasets to the classification task, therefore there is always a need to have a balancing mechanism.

Due to these problems, there is an increasing interest in developing algorithms which are inherently multi-class, *i.e.*, they deal with the problem as a whole without a need for decompositions. There exist previous approaches to multi-class boosting, most notably the recent work of Zou *et al.* [115]. Other methods such as [35, 101, 110] still decompose the multi-class problem to binary tasks. Therefore, our main focus in this thesis is to develop inherently multi-class boosting algorithms. We present various multi-class boosting algorithms in Chapter 3 which work directly with the classification margin and as it will be shown produce state-of-the-art results.

2.2.2 Semi-Supervised Learning and Boosting

Semi-supervised classification has been of growing interest mainly in the machine learning community over the past few years and many methods have been proposed. The methods try to give an answer to the question: "How to improve classification accuracy using unlabeled data together with labeled data?". Supervised learning algorithms require a huge amount of labeled data which are often hard or costly to obtain. Semi-supervised methods offer an interesting solution to this requirement by learning from both labeled and unlabeled data. In many classification problems, a large amount of unlabeled data is available, while it is costly to obtain labeled data. This is especially true for applications in computer vision like object recognition and categorization. Current supervised approaches obtain high recognition rates if enough labeled training data is available. However, for most practical problems there is simply not enough labeled data available, whereas hand-labeling is tedious and expensive, in some cases not even feasible.

The key-idea of semi-supervised learning is to exploit labeled samples as well as a large number of unlabeled samples for obtaining an accurate decision border (see Chapelle *et al.* [20] Zhu [113] for a recent overview of approaches). This differs from the conventional "missing data" problem in that the size of the unlabeled data exceeds that of the labeled by far. The central issue of semi-supervised learning is how to exploit this huge amount of information.

Many semi-supervised learning algorithms use the unlabeled samples to regularize the supervised loss function in the form of

$$\sum_{(\mathbf{x},y)\in\mathcal{X}_l} \ell(y, f(\mathbf{x})) + \lambda \sum_{\mathbf{x}\in\mathcal{X}_u} \ell_u(f(\mathbf{x})), \tag{2.5}$$

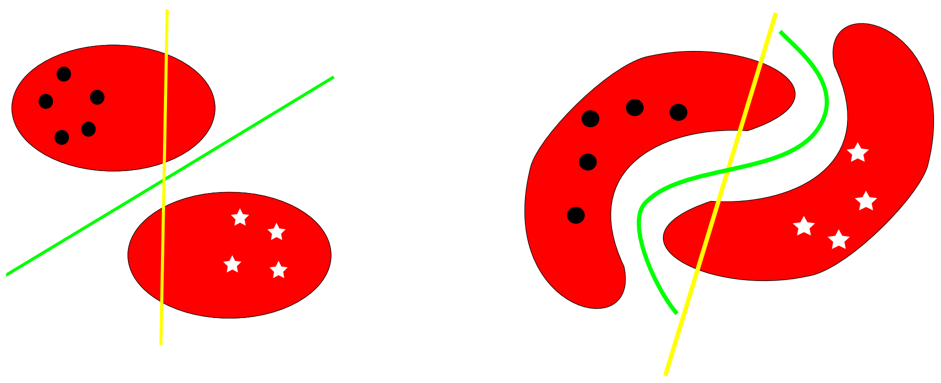
where $f(\cdot)$ is a binary classifier, and $\ell_u(\cdot)$ encodes the penalty related to the unlabeled samples. In the literature, one can find various semi-supervised learning paradigms. In the following sections, we briefly describe each of these ideas.

Self-Training

Self-training [108, 77] can be thought of the simplest way to learn from unlabeled data. First the labeled data is used to train a classifier of choice. Then this classifier checks the unlabeled samples and chooses a subset of them to be labeled and included in the training set (usually the samples which classifier is the most confident). This process continues for some iterations. These algorithms are very sensitive on the accuracy of the starting classifier and usually does not work when the number of labeled samples is low.

Generative Models

Generative models work by using the unlabeled samples to get a better estimate of the conditional density of the classes. Assume a model is trying to estimate the joint



(a) Cluster assumption (b) Manifold assumption

Figure 2.1: These plots visualize how (a) cluster and (b) manifold assumptions can be used. The circles and stars show labeled samples from two different classes. The red regions indicate the unlabeled samples. The yellow decision boundary is obtained by only training on labeled samples. However, as it can be seen it usually crosses the dense regions of the feature space. By the help of unlabeled samples, these decision boundaries can be moved to regions with lower density (green lines).

distribution $p(\mathbf{x}, y)$. By the Bayes rule, we know that one can write it as $p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y)$. When using mixture models, the unlabeled data can be used to identify the mixture distributions $p(\mathbf{x}|y)$. Nigam *et al.* [70] used an EM-based algorithm to learn mixture components. These models usually work when the data can be represented accurately by mixture models.

Cluster Assumption

Some methods, such as *Transductive Support Vector Machines* (TSVM) [52, 95], try to maximize the margin of the unlabeled samples by pushing away the decision boundary from dense regions of the feature space. The intuition behind these models is that the data samples living in very dense regions of the feature space are likely to be from the same class. In other words, if the feature space exhibits clusters, then it is more likely that each of these clusters belong to the same class. Translating this into the decision boundaries, it is likely that the best decision boundary for discriminating the classes is located at the low density regions of the feature space. Such a decision boundary can be built by maximizing not only the margin of the labeled samples, but also the unlabeled samples. This way one makes sure that the decision boundary does not cut through the unlabeled samples in the dense regions. Figure 2.1(a) illustrates this concept for a toy example.

For binary problems, variants of TSVMs [52] maximize the margin for the unlabeled samples by using

$$\ell_u(f(\mathbf{x})) = \max(0, 1 - |f(\mathbf{x})|). \tag{2.6}$$

Note that for a binary task, the classification margin of a sample can be represented as $|f(\mathbf{x})|$. Roughly, this loss function indicates that no matter whether the classifier thinks the unlabeled sample is positive or negative, it has to maximize its margin in either case.

Another approach to use the cluster assumption is to enhance the quality of a kernel by making sure the samples that are in the same cluster of data are represented more similar to each other in the kernel matrix. This approach is successfully used in *cluster kernels* [21]. The RMSBoost algorithm introduced in Chapter 4 uses both a *cluster prior* and the maximum margin principles to regularize the learning process of a multi-class boosting model.

Manifold Assumption

Some algorithms learn the manifold structure of the feature space with unlabeled samples and use it as an additional cue for the supervised learning process, for example, *label propagation* [112], *Laplacian SVMs* [7]. The idea here is that if the data lies on manifolds in the features space, it is more likely that each manifold represents a unique class. Therefore, by making sure that the classifier has a smooth prediction over the manifold where the data lies, we can fulfill this assumption. For binary problems, these algorithms usually work on a loss function which can be represented by

$$\ell_u(f(\mathbf{x})) = \sum_{\substack{\mathbf{x}' \in \mathcal{X}_u \\ \mathbf{x}' \neq \mathbf{x}}} s(\mathbf{x}, \mathbf{x}') \|f(\mathbf{x}) - f(\mathbf{x}')\|^2, \tag{2.7}$$

where $s(\mathbf{x}, \mathbf{x}')$ is a similarity function. Using this penalty term, one can enforce the classifier to predict similar labels if the samples are similar. While the graph-based methods are powerful, the pair-wise terms increase their computational complexity. This concept is depicted in Figure 2.1(b) for a simple two class example. Our GPMBoost algorithm from Chapter 4 also sports a manifold regularization term to ensure smoothness over the manifolds.

Generalized Expectation and Priors

Mann and McCallum proposed a method called Expectation Regularization [67] which, was later on extended to the Generalized Expectation [69] concept. This method relies on having access to an external information source (also know as *priors*). Then the task of the classifier over the unlabeled samples is set to find a model which, on average satisfies what the prior indicates. The following loss function can be thought of a generic penalty term that these class of algorithms use over the unlabeled samples

$$\ell_u(f(\mathbf{x})) = \jmath(\mathbb{E}[f(\mathbf{x})], q), \tag{2.8}$$

where q is the prior, and $\jmath$ is a function measuring the deviations of the expectation of the classifier $\mathbb{E}[f(\mathbf{x})]$ from the prior. Depending on the prior being global (*i.e.* q is defined over the entire unlabeled set) or is local (*i.e.* q depends on local regions of the feature space), one can derive different learning algorithms. For example, for the SERBoost [85] and RMSBoost [87] methods introduced in Chapter 4 we use local priors to regularize the learning process.

Co-Training and Multi-View Learning

In multi-view learning (MVL), the data is expressed by several views or multiple features. For each view a classifier is trained on some labeled data. Then the classifiers iteratively train each other on the unlabeled data. The underlying assumption of MVL is that the unlabeled data can be exploited in a way that enforces the selection of hypotheses that lead to an agreement among the classifiers on the unlabeled data while minimizing the training error on labeled data [96, 59]. Overall, this leads to an increased classification margin and thus lower generalization errors. Such semi-supervised approaches have been previously investigated with different flavors [11, 24, 70]. In many computer vision tasks multiple views are naturally provided which makes the application of MVL interesting. For instance, in object detection and categorization, different features can be considered as different views [60, 23]. Multi-view learning can also lead to more stable tracking results [62, 100]. Also images collected from the web naturally provide different views, because in addition to the visual data, text is also frequently provided [99].

Current multi-view methods work by primarily exchanging the information via label predictions on a subset of the unlabeled data. However, this ignores the uncertainty in each estimated label as well as the information that each view has over the entire set of unlabeled data. In Chapter 4, we propose a novel multi-view boosting algorithm that, on the one hand, performs MVL in the classical sense; *i.e.*, the classifiers provide each other labels for some selected unlabeled samples. However, we additionally regularize each classifier on the rest of the unlabeled samples in a way that it encourages the agreement between the views. In our algorithm, we use an aggregated prior that is set up by the corresponding views; *i.e.*, the iteratively trained classifiers serve each other as priors in order to exploit the rest of the unlabeled samples. However, since the priors can be wrong, we also propose a robust loss function for the semi-supervised regularization which can handle noisy priors. Additionally, most previous MVL methods mainly deal with two-classifier scenarios and are thus mainly co-training variants. Our method is general enough to incorporate not only two, but even also an arbitrary number of views.

Multi-Class Problems and Computational Complexity

The computational complexity of many of state-of-the-art semi-supervised methods limits their application to large-scale problems [67]. This is specially counter-productive for computer vision tasks, such as object recognition or categorization, where a huge amount of unlabeled data is very easy to obtain, for example, via Web downloads. Therefore, it is interesting to find algorithms that can be used for very large datasets.

Most of recent researches have focused on binary classification problems, where multi-class problems are often tackled by applying the same algorithm to a set of decomposed binary tasks. However, multiple repetition of an already heavy-duty algorithm is not a suitable option for solving problems with many classes and samples.

Hence, having an inherent multi-class semi-supervised algorithm with low computational complexity is very interesting for large-scale applications. Methods addressing both of these issues are very rare to find in the literature. Xu and Schuurmans [107] introduce a multi-class extension to the TSVM, which as stated in the paper, is computationally more intensive than the original TSVM formulation. Song *et al.* [97], and Rogers and Girolami [76] propose the use of Gaussian Processes, while Azran [4] use Markov random walks over a graph for solving the multi-class semi-supervised problems. However, the computational complexity of these methods are in the order of $\mathcal{O}(n^3)$.

The RMSBoost algorithm introduced in Chapter 4 is targeted to such problems with many classes and many unlabeled samples and enjoys low computational complexity without the need for multi-class to binary decompositions.

Semi-Supervised Boosting

The above categorization of different algorithms can also be used to differentiate between different semi-supervised boosting algorithms. For example, the CoBoost [24] can be thought of as the first semi-supervised boosting algorithm that is a co-training based algorithm and tries to encourage the agreement between the weak learners over the unlabeled samples. AgreementBoost [59] also works by reducing the variance of the boosting models trained over each view. The work of Bennet *et al.* [8] and D'Alche *et al.* [16] assume the cluster structure of the feature space and maximize the margin of the unlabeled samples. There also exists manifold based approaches, such as the ManifoldBoost [63], SemiBoost [66], and MCSSB [102].

The algorithms which we propose in Chapter 4 can also be categorized into different genres of semi-supervised learning: SERBoost [85] uses expectation regularization, while RMSBoost [87] uses the cluster assumption and maximizes the margin of unlabeled samples. The GPMBoost algorithm improves the RMSBoost and equips it with manifold regularization. Therefore, it can be used for both cases where the cluster or manifold assumptions hold. On the other hand, MV-GPBoost is designed to work with multiple views and works by maximizing the agreement between the classifiers trained over different views.

2.2.3 Online Learning and Boosting

Online learning is an area of machine learning concerned with estimation problems with limited access to the entire problem domain. It is a sequential decision making task where the objectives for the learner are revealed over time. Classical online learning problems can be formulated as a game between the learner and an adversary environment (or teacher). In this repeated game, at any time t the following steps happen:

1. The environment chooses a new sample $\mathbf{x}_t \in \mathbb{R}^d$.
2. The learner responds with its prediction $\hat{y}_t$.
3. The environment reveals the label of the sample y_t.
4. The learner suffers the loss ℓ_t and updates its model.

Online learning is an essential tool for learning from dynamic environments, from very large scale datasets or from streaming data sources. It has been studied extensively in the machine learning community (for a comprehensive overview we refer to [10, 92] and references therein). In computer vision, online learning has been used in applications such as object recognition [42, 6], object detection [74, 106] and tracking [25, 50, 39].

Historically, Oza and Russell [72] were the first to extend the AdaBoost [32] to operate in an online learning scenario. Their formulation and many variants have been used in various computer vision applications [50, 39, 74, 106, 57].

Boosting with convex loss functions is proven to be sensitive to outliers and label noise [64]. This inherent problem of boosting is even more important in online learning problems where the label given by the environment might be quite noisy. Hence, training such a sensitive algorithm in noisy environments usually leads to inferior classifiers. Recently, there has been a great effort to remedy this weakness, *e.g.* by introducing more robust loss functions [64, 94, 57]. There exist theoretical evidences that many boosting algorithms are only able to maximize the *hard-margin* [75] or *average margin* [93] of data samples. Such problems are addressed in other learning methods, specially in support vector machines, by introducing the *soft-margins*. Fortunately, for offline boosting, there exist a few methods which are able to use soft-margins, notably, the Linear Programming Boosting (LPBoost) [27] and its variants [103, 104, 36]. In Chapter 5 we first extend the LPBoost to multi-class problems and then provide an algorithm for solving it in an online learning scenario.

Chapter 3

Multi-Class Boosting

In this chapter, we focus on designing multi-class boosting algorithms which use multi-class margin directly. These set of algorithms are flexible enough to accommodate different types of loss function, which result in a rich set of algorithms suitable for many real-world tasks. We first explain the concept of risk minimization in Section 3.1 and then proceed to present the foundation of the multi-class boosting algorithm in Section 3.2. Section 3.3 discusses the loss functions and in Section 3.4 we explain how learning from priors is relevant to supervised learning. We present the learning algorithms based on the functional gradient descent in Section 3.5 and finally show how the proposed multi-class boosting algorithm works in practice.

3.1 Risk Minimization

Let $\mathcal{X}_l$ denote a set of i.i.d. training samples drawn from an unknown probability distribution $P(y, \mathbf{x})$. The data sample $\mathbf{x}$ is represented as a d-dimensional feature vector and its label for a K-class problem y is coming from the set of labels $\mathcal{Y} = \{1, \ldots, K\}$.

The goal of learning is to find a mapping (which is also known as *classifier*, *model*, *decision rule*, or *learner*)

$$c(\mathbf{x}) : \mathbb{R}^d \to \mathcal{Y} \tag{3.1}$$

such that the *expected risk* defined as

$$R(c) = \int \ell(\mathbf{x}, y; c) dP(y, \mathbf{x}) \tag{3.2}$$

is minimized. In this equation, $\ell(\cdot)$ is a *risk* or *loss* function. Since the $P(y, \mathbf{x})$ is unknown, the *empirical risk* defined over a set of i.i.d. training samples $\mathcal{X}_l$

$$R_{emp}(c) = \frac{1}{|\mathcal{X}_l|} \sum_{(\mathbf{x},y) \in \mathcal{X}_l} \ell(\mathbf{x}, y; c) \tag{3.3}$$

is used in practice.

It is common that the learner is a parametric function in form of $c(\mathbf{x};\boldsymbol{\beta}) \in \mathcal{C}$, where $\boldsymbol{\beta}$ defines the parameters of the learner, and $\mathcal{C}$ denotes the class of functions parameterized by $\boldsymbol{\beta}$. Using parametric learners, it is possible to write the empirical risk in terms of the model parameters [1]

$$R_{emp}(\boldsymbol{\beta}) = \frac{1}{|\mathcal{X}_l|} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \ell(\mathbf{x}, y; \boldsymbol{\beta}), \tag{3.4}$$

and formulate the learning process as an optimization problem for finding the best parameters which minimizes the empirical risk

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} R_{emp}(\boldsymbol{\beta}). \tag{3.5}$$

This formulation is known as the *empirical risk minimization* process and is the basis of many machine learning algorithms. Many algorithms are obtained mainly by choosing: 1) a specific function class $\mathcal{C}$, 2) a specific risk function ℓ, and 3) a specific optimization procedure.

When using classifiers with confidences, we use the $f(\mathbf{x}) = [f_1(\mathbf{x}), \cdots, f_K(\mathbf{x})]^M$ as a multi-valued mapping and $f_k(\mathbf{x})$ represents the confidence of the learner for classifying $\mathbf{x}$ to k-th class. Without loss of generality, we require the following symmetry condition on $f(\mathbf{x})$

$$\forall \mathbf{x}: \sum_{k\in\mathcal{Y}} f_k(\mathbf{x}) = 0. \tag{3.6}$$

Note that for any given function, this condition can always be satisfied by subtracting $\frac{1}{K}$ of the sum of the function values from each class, without any change in the resulting decision rule given in Eq. (2.3).

3.2 Multi-Class Boosting

Boosting can be considered as a *meta learning* algorithm, which accepts another learning algorithm (often known as *base* or *weak* learner) and constructs a new function class out of it. The most commonly used form of boosting constructs additive models in form of

$$f(\mathbf{x};\boldsymbol{\beta}) = \sum_{m=1}^{M} w_m \, g(\mathbf{x};\boldsymbol{\theta}_m), \tag{3.7}$$

[1] Note that when the context is clear, we will interpret the $\ell(\mathbf{x}, y; c)$, $\ell(\mathbf{x}, y; \boldsymbol{\beta})$, and later $\ell(\mathbf{x}, y; f)$ as the same quantities.

where $\boldsymbol{\beta} = [\mathbf{w}|\boldsymbol{\theta}]$ is the collection of model parameters, $\mathbf{w}$ are the parameters of boosting algorithm, $\boldsymbol{\theta} = \{\boldsymbol{\theta}_m\}_{m=1}^{M}$, and $\boldsymbol{\theta}_m$ represents the parameters of the m-th base learner

$$g(\mathbf{x};\boldsymbol{\theta}_m) \in \mathcal{G} : \mathbb{R}^d \to \mathbb{R}^K. \tag{3.8}$$

As it can be seen, boosting uses the base classifiers as *basis functions* for creating a linear span $\mathcal{F} = \text{span}(\mathcal{G})$. In other words, boosting *expands* the original function space $\mathcal{G}$ to create a new (and possibly richer) function class $\mathcal{F}$. Therefore, the learning process of boosting tries to identify a set of basis functions and their associated linear combinations, such that the empirical risk is minimized.

Sometimes it is convenient to write the boosting formulation of Eq. (3.7) in a more compact form of a matrix-vector multiplication

$$f(\mathbf{x};\boldsymbol{\beta}) = G(\mathbf{x};\boldsymbol{\theta})\mathbf{w}, \tag{3.9}$$

where $\mathbf{w} = [w_1, \cdots, w_M]^M \in \mathbb{R}^M$ is the weight vector of all the bases and

$$G(\mathbf{x};\boldsymbol{\theta}) = [g(\mathbf{x};\boldsymbol{\theta}_1)| \dots |g(\mathbf{x};\boldsymbol{\theta}_M)] \in \mathbb{R}^K \times \mathbb{R}^M \tag{3.10}$$

is the response matrix of all base learners for all the classes.

Note that the boosting model of Eq (3.9) is a linear classifier over the feature space G built by its base learners. Since the weight vector $\mathbf{w}$ is shared between all the classes, this leads to an adaptive construction of a discriminative and nonlinear shared feature space defined by the G transformation. Hence, boosting can be also seen as a linear classifier, which not only learns the discriminative linear classifier by optimizing the $\mathbf{w}$ vector, but also constructs the feature space, defined by G mapping, where this linear classifier operates. Figure 3.1 shows the mappings generated by a boosting model with two base learners and 4 classes for a sample $(\mathbf{x}, y)$.

3.3 Loss Functions

3.3.1 Margin Based

Many loss functions used in machine learning algorithms rely on the notion of *margin*, popularized by *support vector machines* (SVMs). Margin of an example $(\mathbf{x}, y)$, roughly describes the confidence of the classifier for assigning the sample $\mathbf{x}$ to its true class y. For a K-class problem, the multi-class margin can be described as

$$m(\mathbf{x}, y; f) = f_y(\mathbf{x}) - \max_{k \neq y} f_k(\mathbf{x}). \tag{3.11}$$

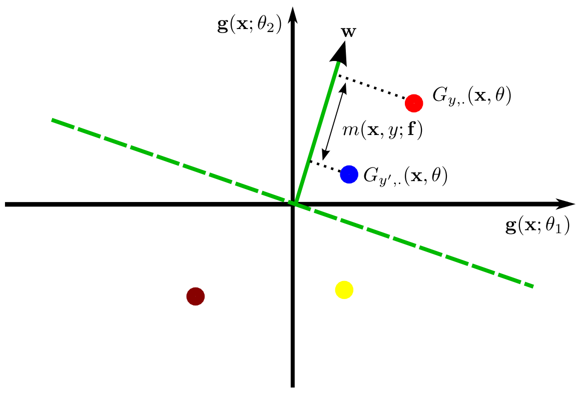

$\mathbf{g}(\mathbf{x};\theta_2)$
$\mathbf{w}$
$G_{y,\cdot}(\mathbf{x},\theta)$
$m(\mathbf{x},y;\mathbf{f})$
$G_{y',\cdot}(\mathbf{x},\theta)$
$\mathbf{g}(\mathbf{x};\theta_1)$

Figure 3.1: Adaptive shared feature space built by weak learners for a sample $(\mathbf{x}, y)$: the red dot represents the target class y while the blue dot shows the closest non-target class y'. The green arrow shows the weight vector $\mathbf{w}$ of the base learners and the margin $m(\mathbf{x}, y; f)$ is the distance between the images of the class specific representations on this vector. The other 2 classes are depicted by yellow and brown dots.

Note that for a correct classification via the decision rule of Eq. (2.3), the margin should be positive $m(\mathbf{x}, y; f) > 0$. In other words, a negative margin would result in a mis-classification.

Since the main goal in learning classifiers is to reduce the number of mis-classifications, the natural loss function to be optimized can be described as

$$\ell_{0-1}(\mathbf{x}, y; f) = \mathbb{I}(m(\mathbf{x}, y; f) \leq 0), \tag{3.12}$$

where $\mathbb{I}$ is the *indicator* function

$$\mathbb{I}(z) = \begin{cases} 1 & z \text{ is true} \\ 0 & z \text{ is false} \end{cases} . \tag{3.13}$$

Therefore, ℓ_{0-1} assigns a unit penalty for a mis-classification and none for a correct classification. While it is natural to use this loss function, due to its discontinuity, non-convexity, and invariance to the variations of the margin (except when the margin passes 0), many algorithms introduce other loss functions, which exhibit more desirable properties.

The following list includes some popular loss functions which are often used in various algorithms:

- Hinge loss:
$$\ell_h(\mathbf{x}, y; f) = \max(0, 1 - m(\mathbf{x}, y; f)), \tag{3.14}$$

- Exponential loss [32]:
$$\ell_e(\mathbf{x}, y; f) = e^{-m(\mathbf{x}, y; f)}, \tag{3.15}$$

- Logit loss [35]:
$$\ell_l(\mathbf{x}, y; f) = \log(1 + e^{-m(\mathbf{x}, y; f)}), \tag{3.16}$$

- Savage loss [94]:
$$\ell_s(\mathbf{x}, y; f) = \frac{1}{(1 + e^{2m(\mathbf{x}, y; f)})^2}. \tag{3.17}$$

Figure 3.2 plots the shape of these loss functions with respect to the margin of an example. As it can be seen, all of them are monotonically decreasing functions. In other words, they penalize less for larger margins, and therefore, they encourage the optimization algorithm to reach larger margins.

Between these loss functions, the exponential loss has the largest penalty for smaller margin, while hinge and Logit loss functions exhibit a linear behavior for negative margins. Savage is a non-convex loss function and shows an interesting behavior for large negative margins: its penalty becomes equivalent to that of $0 - 1$ loss.

In general, from both the theoretical and the experimental point of views, boosting is proven to be sensitive to label noise. This issue was discovered relatively early and hence, more different robust methods [65, 68, 28, 35, 33, 29, 64, 94] have been proposed. In particular, the work of Long *et al.* [64] showed that the loss function has not only a high influence on the learning behavior but also on the robustness. Especially convex loss functions (typically used in boosting) are highly susceptible to random noise. Hence, to increase the robustness the goal is to find less noise-sensitive loss functions.

Recently, Masnadi-Shirazi and Vasconcelos [94] studied the problem of loss-function design from the perspective of *probability elicitation* in statistics and, based on this, derived a non-convex loss-function for boosting. This algorithm, denoted as *SavageBoost*, has shown to be highly resistant to the presence of label noise while also converging fast in case of no noise.

From Figure 3.2 it is clear that the main difference between these loss functions is how they deal with mis-classified samples. There are two scenarios for mis-classification of a sample: (1) The sample is noise-free and it is a learning model which is not able to classify it correctly. (2) The sample has a label noise and the learning model is recovering its true (but hidden) label.

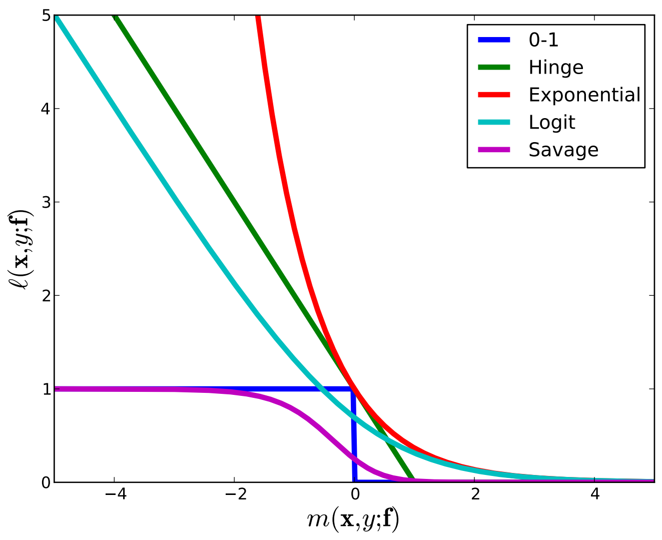


Figure 3.2: Different loss functions used in supervised machine learning methods.

For the second scenario, it can clearly be seen that different loss functions behave differently in such situations by covering different parts of the mis-classification spectrum. The exponential loss, has the most aggressive penalty for a mis-classification. This justifies why AdaBoost dramatically increases the weight of mis-classified samples. Going further, one can see that Logit and Hinge losses are less aggressive and their penalty increases linearly on the left side of the figure. In contrast, Savage follows totally different strategies: it almost gives up on putting pressure over the classifier when there is a severe mis-classification (*i.e.*, $m(\mathbf{x}, y; f)$ is a large negative value).

Summarizing, we expect that when faced with label noise issues the exponential loss to perform poorly, and Savage loss to be the most robust one. Theoretical and experimental findings reported by different authors confirms this hypothesis [35, 64, 94].

3.3.2 Loss Functions from Statistical Learning

There are some loss functions which have their roots in statistical learning methods. Many probabilistic models are trained using the *maximum likelihood* principle, where the loss function is usually the negative *log-likelihood*.

Let $p(k|\mathbf{x};\boldsymbol{\beta})$ describe the posterior probability of assigning sample $\mathbf{x}$ to k-th class, estimated by a model parametrized by $\boldsymbol{\beta}$. The negative log-likelihood is described as

$$\ell_{ll}(\mathbf{x}, y; f) = -\log p(y|\mathbf{x};\boldsymbol{\beta}). \tag{3.18}$$

For example, a *multi-nomial logistic regression* model, estimates the posteriors by

$$p(k|\mathbf{x};\boldsymbol{\beta}) = \frac{e^{f_k(\mathbf{x};\boldsymbol{\beta})}}{\sum_{j\in\mathcal{Y}} e^{f_j(\mathbf{x};\boldsymbol{\beta})}}. \tag{3.19}$$

Given such a model, the negative log-likelihood loss function has the form of

$$\ell_{ll}(\mathbf{x}, y; f) = -f_y(\mathbf{x};\boldsymbol{\beta}) + \log \sum_{j\in\mathcal{Y}} e^{f_j(\mathbf{x};\boldsymbol{\beta})}. \tag{3.20}$$

It should be noted that for a binary problem ($K = 2$), the Logit loss function presented in the previous section is related to the negative log-likelihood loss function [35].

Theorem 3.3.1. *For a binary problem ($K = 2$) and the multi-nomial logistic regression model of Eq.* (3.19)*, the Logit loss function is equivalent to the negative log-likelihood loss function [35].*

Proof. To realize this fact, note that for a binary problem represented by $\mathcal{Y} = \{+1, -1\}$ labels, we have $f_+(\mathbf{x};\boldsymbol{\beta}) = -f_-(\mathbf{x};\boldsymbol{\beta})$, which is a result of the symmetry condition of Eq. (3.6). Additionally, the margin of a sample $(\mathbf{x}, y)$ can be written as

$$m(\mathbf{x}, y; f) = f_y(\mathbf{x}) - f_{-y}(\mathbf{x}) = 2f_y(\mathbf{x}). \tag{3.21}$$

Using the Logit loss, we can see

$$\begin{aligned}\ell_l(\mathbf{x}, y; f) =& \log(1 + e^{-2f_y(\mathbf{x})}) = \log(e^{-f_y(\mathbf{x})}(e^{f_y(\mathbf{x})} + e^{-f_y(\mathbf{x})})) = \\ =& - f_y(\mathbf{x}) + \log(e^{f_y(\mathbf{x})} + e^{f_{-y}(\mathbf{x})}) = \ell_{ll}(\mathbf{x}, y; f).\end{aligned} \tag{3.22}$$

□

For multi-class problems ($K > 2$), the connection to the classification margin of Eq. (3.11) is not direct, but we prove the following theorem which shows an approximate relationship with the Logit loss.

Theorem 3.3.2. *For the multi-nomial logistic regression model of Eq.* (3.19)*, the following holds for the negative log-likelihood loss function*

$$\ell_{ll}(\mathbf{x}, y; f) = \log(1 + \sum_{k \neq y} e^{-m_k(\mathbf{x}, y; f)}), \tag{3.23}$$

where $m_k(\mathbf{x}, y; f)$ *is the classification margin between the true class* y *and* k*-th class defined as*

$$m_k(\mathbf{x}, y; f) = f_y(\mathbf{x}) - f_k(\mathbf{x}). \tag{3.24}$$

Proof. We prove this by using the definition of the multi-nomial logistic regression model as

$$\begin{aligned} \ell_{ll}(\mathbf{x}, y; f) &= -\log p(y|\mathbf{x}; \boldsymbol{\beta}) = \log(e^{-f_y(\mathbf{x}; \boldsymbol{\beta})} \sum_{k \in \mathcal{Y}} e^{f_k(\mathbf{x}; \boldsymbol{\theta})}) = \\ &= \log(\sum_{k \in \mathcal{Y}} e^{-f_y(\mathbf{x}; \boldsymbol{\beta}) + f_k(\mathbf{x}; \boldsymbol{\beta})}) = \log(1 + \sum_{k \neq y} e^{-m_k(\mathbf{x}, y; f)}). \end{aligned} \tag{3.25}$$

□

From this theorem we can see that the negative log-likelihood, approximately replaces the true classification margin in the Logit loss function (3.16) with the sum of an exponential term for the margin between each class pair.

3.3.3 Pseudo-Margins

Recently, Zou *et al.* [115] extended the concept of Fisher-consistent loss functions [61] from binary classification to the domain of multi-class problems. This concept explains the success of margin-based loss functions and their statistical characteristics.

They name $f(\mathbf{x})$ a margin vector, if

$$\forall \mathbf{x} : \sum_{k=1}^{K} f_k(\mathbf{x}) = 0. \tag{3.26}$$

The loss function $\ell(\cdot)$ is Fisher-consistent, if the minimization of the expected risk

$$\hat{f}(\mathbf{x}) = \arg\min_{f(\mathbf{x})} \int \ell(f_y(\mathbf{x})) dP(y, \mathbf{x}) \tag{3.27}$$

has a unique solution and

$$c(\mathbf{x}) = \arg\max_i \hat{f}_i(\mathbf{x}) = \arg\max_i p(y = i|\mathbf{x}), \tag{3.28}$$

where $c(\mathbf{x})$ is the learned multi-class classifier. Thus, by minimizing a Fisher-consistent margin-based loss function, one can approximate the unknown Bayes decision rule. The intuition is that the classification confidence for the k-th class, denoted as $f_k(\mathbf{x})$, is directly related to the class conditional probabilities $p(y = k|\mathbf{x})$. Therefore, because of the symmetry condition Eq.(3.6), maximizing the confidence of a class is equivalent to reducing the confidence for all other classes. In this respect, the exponential, Logit, and Hinge losses are Fisher-consistent loss functions [115]. However, their definitions and the way that the algorithms are constructed does not relate directly to the true multi-class margin of Eq. (3.11), and hence, optimization based only on the confidence of the target class does not guarantee that the sample will be classified correctly.

In order to observe this, assume that we have a 3 class problem and we have an example $(\mathbf{x}, 1)$ which belongs to the first class. Let the classifier output for this sample be $f(\mathbf{x}) = [8, 10, -18]$. As it can be seen, this classifier is a margin vector (its output sums to zero). Based on the approach of Zou *et al.* [115], if we use an exponential loss function, the penalty for this sample will be e^{-8} which is very low. Therefore, the optimization algorithm which operates over this sample will ignore it and will assume that this example is correctly solved. But, clearly this sample will be mis-classified as the class with the maximum confidence is the second class. However, the loss function is not aware of this (since it is not optimizing the exact margin which is -2 in this case), and hence, it will not be able to find a better solution. This is the reason why we call these methods only optimize the *pseudo-margin* of an example.

3.4 Supervised Learning from Priors

In the next chapters, we will employ *priors* for learning from unlabeled samples. Therefore, here we introduce the concept, and show that in case of supervised learning, the negative log-likelihood is equivalent to training from $0-1$ priors using the *Kullback-Leibler* divergence as the loss function.

Assume we are given a prior probability in the form of

$$\forall \mathbf{x}, k \in \mathcal{Y} : q(k|\mathbf{x}). \tag{3.29}$$

The goal of the learning algorithm is to produce a model, which is able to match the prior probability over the training samples. For a supervised learning problem, the prior can be easily obtained via $0-1$ prior

$$\forall (\mathbf{x}, y), k \in \mathcal{Y} : q_{0-1}(k|\mathbf{x}) = \mathbb{I}(k = y). \tag{3.30}$$

Since the goal is to reduce the differences between the model and the prior, we can employ distance functions, such as Kullback-Leibler (KL), used for measuring the divergence of two distributions. The following theorem shows that using a $0-1$ prior and Kullback-Leibler divergence is equivalent to using the negative log-likelihood loss function.

Theorem 3.4.1. *The negative log-likelihood loss function is equivalent to using a* $0-1$ *prior and Kullback-Leibler divergence as the loss function.*

Proof. The Kullback-Leibler divergence between two distributions q and p is defined as

$$D_{KL}(q\|p) = H(q,p) - H(q), \tag{3.31}$$

where $H(q,p)$ is the cross-entropy between q and p, and $H(q)$ is the entropy of q. Since for our example of Eq. (3.30), the entropy of the prior is fixed ($H(q_{0-1}) = 0$), we can drop it from the loss function. Hence, the loss function only consists of the cross-entropy term and can be written as

$$\ell_{kl}(\mathbf{x}, y; f) = -\sum_{k \in \mathcal{Y}} q_{0-1}(k|\mathbf{x}) \log p(k|\mathbf{x}; \boldsymbol{\beta}) = -\log p(y|\mathbf{x}; \boldsymbol{\beta}). \tag{3.32}$$

□

While this theorem is true for a $0-1$ prior, it should be noted that for other types of priors, this equivalence does not hold.

3.5 Learning with Functional Gradient Descent

Given a loss function ℓ, the learning process for boosting is defined as

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \sum_{(\mathbf{x},y) \in \mathcal{X}_l} \ell(\mathbf{x}, y; \boldsymbol{\beta}), \tag{3.33}$$

which requires finding the parameters of the base learners $\boldsymbol{\theta}$ together with their weights $\mathbf{w}$. Usually optimizing for a global solution for this problem is difficult, therefore, many boosting algorithms adopt an approximate solution called *stagewise additive modeling* [35]. This method is explained in Algorithm 1.

Algorithm 1 Stagewise Additive Modeling

```
Require: Training samples: $\mathcal{X}$.
Require: $M$ as the number of base models.
1: Set the model $f_0(\mathbf{x}) = 0$.
2: for $m = 1$ to $M$ do
3:     Find the solution for the optimization problem:
           $$(w_m^*, \boldsymbol{\theta}_m^*) = \arg\min_{w_m, \boldsymbol{\theta}_m} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \ell(\mathbf{x}, y; f_{m-1}(\mathbf{x}) + w_m g(\mathbf{x}; \boldsymbol{\theta}_m)). \quad (3.34)$$
4:     Update the model: $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + w_m^* g(\mathbf{x}; \boldsymbol{\theta}_m^*)$ .
5: end for
6: Output the final model: $f(\mathbf{x}; \boldsymbol{\beta}^*) = \sum_{m=1}^{M} w_m^* \ g(\mathbf{x}; \boldsymbol{\theta}_m^*)$ .
```

This algorithm alternates between two main parts: 1) Optimization part which finds a solution for Eq. (3.34) by looking for such a base function $g(\mathbf{x}; \boldsymbol{\theta}_m)$ and its weight α_m which if added to the solution of the previous stage $f_{m-1}(\mathbf{x})$, reduces the empirical risk the most. 2) Update part, which adds the best solution obtained from the previous step to the final solution.

Because of the additive nature of this algorithm, its called *additive modeling*. Additionally, because when we find a solution for the m-th stage of the algorithm, we fix those solution for the next iterations (*i.e.* we never look back to alter the previous solutions when we obtain a new base learner), it is also called a *stagewise* algorithm.

Considering the Algorithm 1, one can also see the similarity of this method to the *gradient methods* used widely in optimization techniques. To better realize this fact, we present also a generic gradient descent method in Algorithm 2 for finding a solution for

$$\theta^* = \arg\min_{\theta} J(\theta). \tag{3.35}$$

Using this similarity, one can also use a *functional gradient descent* method for learning the parameters of a boosting model [34]. Here we will present two methods to achieve this goal. Figure 3.3 shows how the traditional gradient descent (Figure 3.3(a)) can be converted to the functional gradient descent (Figure 3.3(b)). The details of derivations of the gradients for each of the loss functions discussed in this section can be found in Appendix A.

Algorithm 2 Gradient Descent Method.

```
1: Set the initial solution $\theta_0$.
2: for $m = 1$ to $M$ do
3:     Compute the gradient descent direction:
           $\theta^*_m = -\frac{\partial J(\theta)}{\partial \theta}|_{\theta=\theta_{m-1}}$.    (3.36)
4:     Choose a step size $w^*_m$.
5:     Update the model by a gradient descent step: $\theta_m = \theta_{m-1} + w^*_m \theta^*_m$ .
6: end for
7: Output the final model: $\theta^* = \sum_{m=1}^M w^*_m \theta^*_m$
```

3.5.1 Regression Formulation

Let us first concentrate on finding a solution to $\boldsymbol{\theta}_m$ in Eq. (3.34). At each iteration of the gradient descent, we choose the *steepest descent* direction in Eq. (3.36). Generalizing this to the space of functions, we can see that this is the equivalent of choosing the m-th base function as

$$g(\mathbf{x}; \boldsymbol{\theta}^*_m) = -\nabla \ell(\mathbf{x}, y; f_{m-1}), \tag{3.37}$$

where ∇ is the gradient operator, and g corresponds to the a function which has the steepest descent direction. However, since the loss function (and hence its gradients) are only defined over the training samples, one can not use this solution directly. Therefore, we try to find a base function which approximates the gradient descent direction the best

$$\boldsymbol{\theta}^*_m = \arg\min_{\boldsymbol{\theta}_m} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \| -\nabla \ell(\mathbf{x}, y; f_{m-1}) - g(\mathbf{x}; \boldsymbol{\theta}_m)\|_2^2. \tag{3.38}$$

Note that since the gradients are continuous quantities, we formulate the learning of the m-th base function as a least square regression problem. Now that we have the function which approximates the steepest descent direction, we perform a line-search to find the step size by solving

$$w^*_m = \arg\min_{w_m} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \ell(\mathbf{x}, y; f_{m-1}(\mathbf{x}) + w_t g(\mathbf{x}; \boldsymbol{\theta}^*_m)). \tag{3.39}$$

Combining these two steps, we present a learning approach for multi-class boosting via regression base function in Algorithm 3.

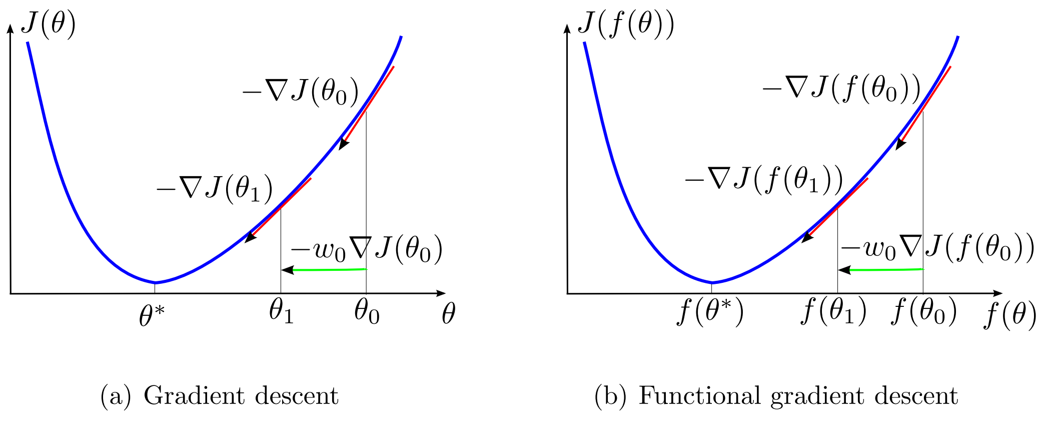


Figure 3.3: These plots show how the gradient descent can be converted to functional gradient descent.

3.5.2 Classification Formulation

Another approach to derive a functional gradient descent based boosting algorithm is to use an approximation for the problem of Eq. (3.34). Note that, we can approximate this problem by replacing the inner term with its first-order Taylor expansion around $f_{m-1}(\mathbf{x})$

$$\begin{aligned}\ell(\mathbf{x}, y; f_{m-1}(\mathbf{x}) + g(\mathbf{x}; \boldsymbol{\theta}_m)) \approx & \ell(\mathbf{x}, y; f_{m-1}) + \\ & + \nabla \ell(\mathbf{x}, y; f_{m-1})^M g(\mathbf{x}; \boldsymbol{\theta}_m).\end{aligned} \tag{3.42}$$

Since the first term in the expansion is fixed, the optimization problem of Eq. (3.34) can be approximated as

$$\boldsymbol{\theta}_m^* = \arg\max_{\boldsymbol{\theta}_m} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} -\nabla \ell(\mathbf{x}, y; f_{m-1})^M g(\mathbf{x}; \boldsymbol{\theta}_m). \tag{3.43}$$

The corresponding algorithm is presented in Algorithm 4.

Algorithm 3 Functional Gradient Descent for Multi-Class Boosting: Regression Approach.

```
Require: Training samples: $\mathcal{X}$.
Require: $T$ as the number of base models.
1: Set the model $f_0(\mathbf{x}) = 0$.
2: for $m = 1$ to $M$ do
3:     Find the solution for the optimization problem:
       $$\boldsymbol{\theta}_m^* = \arg\min_{\boldsymbol{\theta}_m} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \| -\nabla\ell(\mathbf{x}, y; f_{m-1}) - g(\mathbf{x}; \boldsymbol{\theta}_m)\|_2^2. \quad (3.40)$$
4:     Choose a step size
       $$w_m^* = \arg\min_{w_t} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \ell(\mathbf{x}, y; f_{m-1}(\mathbf{x}) + w_m g(\mathbf{x}; \boldsymbol{\theta}_m^*)). \quad (3.41)$$
5:     Update the model: $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + w_m^* g(\mathbf{x}; \boldsymbol{\theta}_m^*)$
6: end for
7: Output the final model: $f(\mathbf{x}; \boldsymbol{\beta}^*) = \sum_{m=1}^M w_m^* \, g(\mathbf{x}; \boldsymbol{\theta}_m^*)$ .
```

3.5.3 Unified View

Note that the two Algorithms 3 and 4 are closely related to each other. In order to realize that, we write the inner part of the least square problem as

$$\| -\nabla\ell(\mathbf{x}, y; f_{m-1}) - g(\mathbf{x}; \boldsymbol{\theta}_m)\|_2^2 = \|\nabla\ell(\mathbf{x}, y; f_{m-1})\|_2^2 + \|g(\mathbf{x}; \boldsymbol{\theta}_m)\|_2^2 + \\ +2\nabla\ell(\mathbf{x}, y; f_{m-1})^M g(\mathbf{x}; \boldsymbol{\theta}_m). \quad (3.46)$$

The first term in this equation is independent of the optimization variables, and hence can be dropped from the optimization. Now, if we assume that $\forall \mathbf{x}, \boldsymbol{\theta}_m : \|g(\mathbf{x}; \boldsymbol{\theta}_m)\|_2^2 = 1$, then we can also drop the second term from the optimization. Hence, the resulting optimization is equivalent to the one presented in Eq. (3.38), which means that if the norm of the base functions is fixed, then Algorithms 3 and 4 are the same.

3.5.4 Step Size and Shrinkage Factor

All these algorithms involve a procedure represented in Eq. (3.39) to find the step size for performing the functional gradient descent update. There are generally three ways

Algorithm 4 Functional Gradient Descent for Multi-Class Boosting: Classification Approach.

```
Require: Training samples: $\mathcal{X}$.
Require: $T$ as the number of base models.
1: Set the model $f_0(\mathbf{x}) = 0$.
2: for $m = 1$ to $M$ do
3:     Find the solution for the optimization problem:
           $$\boldsymbol{\theta}_m^* = \arg\max_{\boldsymbol{\theta}_m} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} -\nabla\ell(\mathbf{x}, y; f_{m-1})^M g(\mathbf{x}; \boldsymbol{\theta}_m). \quad (3.44)$$
4:     Choose a step size
           $$w_m^* = \arg\min_{w_t} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \ell(\mathbf{x}, y; f_{m-1}(\mathbf{x}) + w_m g(\mathbf{x}; \boldsymbol{\theta}_m^*)). \quad (3.45)$$
5:     Update the model: $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + w_m^* g(\mathbf{x}; \boldsymbol{\theta}_m^*)$
6: end for
7: Output the final model: $f(\mathbf{x}; \boldsymbol{\beta}^*) = \sum_{m=1}^M w_m^* \, g(\mathbf{x}; \boldsymbol{\theta}_m^*)$ .
```

to tackle this part: 1) Perform a line-search, 2) Find a closed form solution, 3) Set the step size to a fixed value.

While performing a line-search is always feasible, some forms of loss functions might allow for obtaining a closed form solution by using the fact that the gradients of the optimization function in Eq. (3.39) should vanish at any local minimum or maximum. Therefore, it might be possible to obtain a closed form solution by solving

$$\sum_{(\mathbf{x},y)\in\mathcal{X}_l} \frac{\partial \ell(\mathbf{x}, y; f_{m-1}(\mathbf{x}) + w_m g(\mathbf{x}; \boldsymbol{\theta}_m^*))}{\partial w_m} = 0, \tag{3.47}$$

and checking for the solution to be the minimizer. For example, the estimation of the w_m in the original AdaBoost algorithm [32] for a binary problem is shown to be a closed form solution to such a problem [35].

It is also possible to set the step size to a fixed value $\forall t : w_m = \nu$, where $\nu \in (0, 1]$ is known as the *shrinkage factor* [34, 79]. As several researchers [79] suggested, using a shrinkage factor usually slows down the learning of the model, but improves the classification accuracy over using a line-search to select the w_m. This approach will be mainly used in this work.

3.5.5 Practical Considerations for Base Learners

For regression based learning methods presented in Algorithm 3, we can directly use the gradient vectors $\nabla\ell(\mathbf{x}, y; f_{m-1})$. One way to solve the optimization problem presented by Eq. (3.40) is to train K regressors for each dimension of the gradient vector.

Algorithm 4 describes classification based learning for multi-class boosting. For these algorithms the goal of the base learner is to provide a function which has the highest correlation with the negative direction of the loss function. One way of solving the optimization problem of Eq. (3.44) is to train K binary classifiers by using

$$d_{\mathbf{x},k} = |\frac{\partial\ell(\mathbf{x}; f)}{\partial f_k(\mathbf{x})}| \tag{3.48}$$

as the *sample weight* for training the j-th classifier and

$$\hat{y}_{\mathbf{x},k} = \text{sign}(-\frac{\partial\ell(\mathbf{x}; f)}{\partial f_k(\mathbf{x})}) \tag{3.49}$$

as the *pseudo-label*. Using these two terms, the Eq. (3.44) can be written as

$$\boldsymbol{\theta}^*_m = \arg\max_{\boldsymbol{\theta}_m} \sum_{\mathbf{x}\in\mathcal{X}} \sum_{k\in\mathcal{Y}} d_{\mathbf{x},k}\hat{y}_{\mathbf{x},k} g_k(\mathbf{x}; \boldsymbol{\theta}_m). \tag{3.50}$$

If the base learner is a binary classifier, *i.e.* $g_k(\mathbf{x}; \boldsymbol{\theta}_m) \in \{-1, +1\}$, then this optimization problem is as simple as training K binary classifiers with using $d_{\mathbf{x},k}$ and $\hat{y}_{\mathbf{x},k}$ as the weights and labels of a sample $\mathbf{x}$.

While this method works for almost all classes of base learners, one can also benefit from training inherently multi-class classifiers, such as decision trees or random forests. Since those methods can only use one single label and weight per example, in the following theorem we show how one can select the best labels and weights for multi-class base learners.

Theorem 3.5.1. *The solution of Eq.*(3.44) *using a multi-class classifier* $c(\mathbf{x}; \boldsymbol{\theta}_m) \in \mathcal{Y}$ *can be obtained by solving*

$$\boldsymbol{\theta}^*_m = \arg\min_{\boldsymbol{\theta}_m} \sum_{\mathbf{x}\in\mathcal{X}} d_{\mathbf{x}}\mathbb{I}(c(\mathbf{x}) \neq \hat{y}) \tag{3.51}$$

where

$$d_{\mathbf{x}} = \max_{k \in \mathcal{Y}} - \frac{\partial \ell(\mathbf{x}; f)}{\partial f_k(\mathbf{x})} \tag{3.52}$$

is the weight and

$$\hat{y}_{\mathbf{x}} = \arg\max_{k \in \mathcal{Y}} - \frac{\partial \ell(\mathbf{x}; f)}{\partial f_k(\mathbf{x})} \tag{3.53}$$

is the pseudo-label for the sample $\mathbf{x}$.

Proof. Note that a multi-class classifier $c(\mathbf{x}; \boldsymbol{\theta}_m)$ can be represented as a response vector $g_k(\mathbf{x}) = \mathbb{I}(c(\mathbf{x}; \boldsymbol{\theta}_m) = k) - \frac{1}{K}$. Thus, the Eq.(3.44) becomes

$$\sum_{(\mathbf{x},y) \in \mathcal{X}_l} \sum_{k \in \mathcal{Y}} \frac{\partial \ell(\mathbf{x}; f)}{\partial f_k(\mathbf{x})} \big(\mathbb{I}(c(\mathbf{x}; \boldsymbol{\theta}_m) = k) - \frac{1}{K}\big) =$$
$$\sum_{(\mathbf{x},y) \in \mathcal{X}_l} \frac{\partial \ell(\mathbf{x}; f)}{\partial f_{c(\mathbf{x};\boldsymbol{\theta}_m)}(\mathbf{x})} + \text{const.} \tag{3.54}$$

We can see that for a multi-class classifier, the only remaining term is the gradient term, where the classifier has a prediction. Therefore, in order to minimize the loss function Eq.(3.44) the most, we choose the gradient term which has the largest magnitude and use it as the weight of an example. This corresponds to Eq. (3.52) and Eq. (3.53), respectively, which selects the largest gradient descent direction between all the known classes. □

It should be noted that for labeled samples, because of the shape of the loss functions and their derivatives presented in the two previous sections, we always have

$$\hat{y}_{\mathbf{x}} = y, \tag{3.55}$$

as only the negative of the derivatives with respect to the target class y is positive

$$-\frac{\partial \ell(\mathbf{x}, y; f)}{\partial f_y(\mathbf{x})} \geq 0, \tag{3.56}$$

and for all others it is always negative

$$\forall k \neq y : -\frac{\partial \ell(\mathbf{x}, y; f)}{\partial f_k(\mathbf{x})} < 0. \tag{3.57}$$

Since later on we will use functional gradient descent for semi-supervised and unsupervised learning problems, we formulated our algorithm for the classification based learning, regardless of the label of an example. Therefore, the derivations hold for unlabeled samples, by replacing the gradients term with the corresponding gradients of the unlabeled loss function. This means that the pseudo-labels and sample weights developed for classification based learning, can be in fact directly used by the base learners for unlabeled data samples. That was the reason to mention the pseudo-labels in the previous theorem. Of course, we showed also that if the sample is a labeled one, the pseudo-label is the same as the original class label.

3.6 Experiments

We evaluate the performance of the proposed multi-class boosting algorithm using the Savage loss function with other state-of-the-art methods on a set of multi-class machine learning benchmark datasets obtained from the UCI repository. By comparing the Savage loss with other loss functions mentioned earlier, we found that there is not much performance difference when the labels are not noisy. However, as soon as we have noisy labels, the Savage is consistently better than all other algorithms. Note that similar results have been observed by Shirazi and Vasconcelos [94]. Therefore, in the following experiments we only use the Savage loss, which works reasonably well in all of these situations. The code implementing these algorithms can be obtained from the following link[2].

3.6.1 UCI Benchmarks

We compare with the following multi-class classifiers: Random Forests (RF) [15], four multi-class formulations of AdaBoost namely AdaBoost.ML [115], SAMME [111], AdaBoost.ECC [43], and the recent algorithm of AdaBoost.SIP [110]. As the last algorithm, we also compare with the multi-class support vector machine algorithm. For Random Forests, we train 250 randomized trees. For the SVM, we use the RBF kernel and perform model selection by a grid search for selecting the kernel width σ and capacity parameter C. For our GBoost algorithm, we use 5 extremely randomized trees as weak learners, and set the number of weak learners $T = 50$ and fix the shrinkage factor to $\nu = 0.05$ for all the experiments. We repeat the experiments for 5 times and report the average test error.

[2]http://www.ymer.org/amir/software/multi-class-semi-supervised-boosting/

Method/Dataset	DNA	Letter	Pendigit	USPS
GBoost	0.0582	**0.0265**	*0.0387*	*0.0524*
RF	0.0683	0.0468	0.0496	0.0610
SVM	0.0559	*0.0298*	**0.0360**	**0.0424**
AdaBoost.ML	0.0649	0.0385	0.0441	0.0558
SAMME [110]	0.1071	0.4938	0.3391	N/A
AdaBoost.ECC [110]	**0.0506**	0.2367	0.1029	N/A
AdaBoost.SIP [110]	*0.0548*	0.1945	0.0602	N/A

Table 3.1: Classification error on machine learning benchmark datasets. The best performing method is marked in **bold-face** font, while the second best is shown in *italic*.

The results over *DNA*, *Letter*, *Pendigit*, and *USPS* datasets are shown in Table 3.1. As it can be seen, our algorithm achieves results comparable to other multi-class classifiers. The best performing method, on average, is the SVM with RBF kernel. However, our algorithm achieves these results without any need for model selection (we use a fixed setting for all the experiments in this section), and is considerably faster during both training and testing. For example, for the Letter dataset with 15000 training and 4000 samples, our unoptimized Python/C++ implementation finishes the training and testing in 54 seconds, while the training of the SVM using Shogun LibSVM interface [98] takes around 156 seconds (without taking the model selection into account). Comparing the performance of the GBoost which optimizes the true multi-class margin and the AdaBoost.ML, which uses the pseudo-margin, we can see that GBoost consistently has a better performance.

3.6.2 SSL Benchmarks

In Chapter 4, we will conduct experiments over the semi-supervised benchmark datasets from the book of Chapelle *et al.* [20]. Here we only report the performance of the supervised algorithms. These datasets are split into 2 categories with training sets consisting of 10 and 100 labeled samples. We compare with the following supervised algorithms: Nearest Neighbor (NN), SVM with RBF kernel, and our boosting algorithm (GBoost).

The results are shown in Table 3.2 and Table 3.3 for 10 and 100 labeled samples, by computing the average test classification error over 12 different splits. As it can be seen, for the case where the number of training samples is very low (10), the NN classifier

Methods-Dataset	g241c	g241d	Digit1	USPS	COIL	BCI
1-NN [20]	**44.05**	**43.22**	**23.47**	**19.82**	**65.91**	**48.74**
SVM [20]	47.32	*46.66*	*30.60*	*20.03*	*68.36*	49.85
RF (weak)	47.51	48.44	42.42	22.90	75.72	49.35
GBoost-RF	*46.77*	*46.61*	38.70	20.89	69.85	*49.12*

Table 3.2: Classification error on semi-supervised learning benchmark datasets for 10 labeled samples. The best performing method is marked in **bold-face** font, while the second best is shown in *italic*.

Methods-Dataset	g241c	g241d	Digit1	USPS	COIL	BCI
1-NN [20]	40.28	37.49	*06.12*	**07.64**	23.27	44.83
SVM [20]	**23.11**	**24.64**	**05.53**	*09.75*	*22.93*	**34.31**
RF (weak)	44.23	45.02	17.80	16.73	34.26	43.78
GBoost-RF	*31.84*	*32.38*	*06.24*	13.81	**21.88**	*40.08*

Table 3.3: Classification error on semi-supervised learning benchmark datasets for 100 labeled samples. The best performing method is marked in **bold-face** font, while the second best is shown in *italic*.

performs the best, while SVM and GBoost deliver acceptable results. However, when the number of training samples is increased to 100, the results are mixed. SVM is generally performing the best, while the GBoost is the second best method. It should be noted that for the multi-class dataset (COIL), the GBoost performs the best.

3.7 Discussion

In this chapter, we introduced GBoost, our multi-class boosting method. This algorithm works directly with the multi-class classification margin. GBoost is a versatile algorithm and can handle different loss functions. We experimented with Savage loss function due to its robustness and showed that GBoost is competitive with other multi-class boosting algorithms. Another desirable property of GBoost is that it can be used for many tasks without a need for model selection and parameter tuning. Assume that the computational complexity of the base learning algorithm is $\mathcal{O}(n_b)$. Since our boosting model needs only one pass over the training data to compute the weights, the computational complexity of the training phase is $\mathcal{O}(Mn_b)$.

In all of our experiments, we used small Random Forests as the weak learners of boosting. RFs are inherently multi-class learners which again makes them suitable for our boosting procedure. The computational complexity of RFs does not depend on the size of the feature space (as it does the random feature selection), and they scale very well with respect to the number of training samples. The maximum depth of a tree is dependent on the number of samples, because the tree stops growing from a node if the number of samples in that node falls bellow a certain threshold. In the worst case scenario, the growing will stop if there is only one sample in a node. Let us assume that the tree grows in a balanced manner. Therefore, the maximum depth that a tree can reach with n training samples is $d \approx \log_2 n$. As a result, we can think of the computational cost of the RFs as $\mathcal{O}(n \log_2 n)$, and hence, the complexity of the GBoost can be computed as $\mathcal{O}(n \log_2 n)$ as well, since usually $M \ll n$. By taking the advantages of RFs and combining them with the learning power of boosting, we are able to produce an algorithm which can be easily applied to large-scale problems without an issue.

Chapter 4

Semi-Supervised Learning

In this chapter, we focus on developing multi-class semi-supervised learning algorithms based on the concept of learning from priors. In Section 3.4, we introduced this idea for the supervised learning case, which in this section we will extend it for the purpose of semi-supervised learning. We call our algorithm GPMBoost, which stands for Gradient, Prior, and Manifold-based Boosting. We first present the general settings for the GPM-Boost algorithm in Section 4.1. In Section 4.2 we introduce how different kinds of priors can be used and in Section 4.3, we propose different loss functions which can be used for unlabeled regularization. We use the functional gradient descent for creating our learning algorithms, which is discussed in Section 4.4. In order to apply our semi-supervised learning model to multi-view data, in Section 4.5, we extend our algorithm for this kind of problems. In Section 4.6 we briefly review our previously proposed algorithms, SER-Boost [85] and RMSBoost [87], and compare it with the GPMBoost method. Finally, we present an extensive set of experiments evaluating these algorithms on various machine learning and computer vision tasks.

4.1 GPMBoost

Let $\mathcal{X}_l$ and $\mathcal{X}_u$ be the set of i.i.d. training labeled and unlabeled examples drawn randomly from an unknown probability distribution $P(y, \mathbf{x})$. Assume we are given a prior probability in form of

$$\forall \mathbf{x}_u \in \mathcal{X}, k \in \mathcal{Y} : q(k|\mathbf{x}). \tag{4.1}$$

We formulate the semi-supervised learning as an optimization problem with three goals: 1) the model should attain low mis-classification errors on the labeled data (*e.g.* by means of maximizing the margin), 2) the model should be able to be consistent with the prior probability over the unlabeled training samples, 3) the model should have smooth posterior estimates over the manifold where the data lies. These goals can be written as

$$\begin{aligned}\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} R_{emp}(\boldsymbol{\beta}) =& \frac{1}{|\mathcal{X}_l|} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \ell(\mathbf{x}, y; \boldsymbol{\beta}) + \\ &+ \frac{\gamma}{|\mathcal{X}_u|} \sum_{\mathbf{x}\in\mathcal{X}_u} \Big(\lambda \jmath^p(\mathbf{x}, q; \boldsymbol{\beta}) + (1-\lambda) \sum_{\substack{\mathbf{x}'\in\mathcal{X}_u \\ \mathbf{x}'\neq\mathbf{x}}} \frac{s(\mathbf{x},\mathbf{x}')}{z(\mathbf{x})} \jmath^m(\mathbf{x}, \mathbf{x}'; \boldsymbol{\beta})\Big),\end{aligned} \tag{4.2}$$

where $\jmath^p$ is a loss function which measures the deviations of the model from the prior for a given sample $\mathbf{x}$, $\jmath^m$ measures the deviations of the posterior estimate for $\mathbf{x}$ compared to its neighbor $\mathbf{x}'$, $s(\mathbf{x}, \mathbf{x}')$ is a similarity measurement, $z(\mathbf{x}) = \sum_{\substack{\mathbf{x}'\in\mathcal{X}_u \\ \mathbf{x}'\neq\mathbf{x}}} s(\mathbf{x}, \mathbf{x}')$ is a normalization factor, γ tunes the effect of the semi-supervised training, and λ balances the influence of the prior and manifold regularization terms.

As it can be seen, from the multi-class boosting learning point of view, this optimization problem can also be easily solved with the techniques and algorithms presented in the previous chapter. The only problems to be tackled are: 1) How to get priors for unlabeled samples, 2) Which loss functions $\jmath$ to use, 3) How the functional gradient descent is computed? The remainder of this chapter will try to answer these questions.

4.2 Priors

Priors are usually obtained by utilizing external sources of information, and need not to be completely accurate. We emphasize that the model itself is not used in order to obtain the prior, therefore, the prior is fixed and does not depend on the learner. The cases where one wants to include the model into the prior, can be seen as a different form of the manifold regularization term we defined in Eq. (4.2).

In this section we will focus on different ways of obtaining priors without involving the current algorithm directly. Technically, this means that the prior q is independent of the model parameters $\boldsymbol{\beta}$ which we are optimizing. As it will be seen later, when we define loss functions for the unlabeled samples, this sometimes leads to simplifications in the training process. For example, if the loss function consists of an entropy term over the prior, we can simply ignore it in the optimization process, since the entropy will be a fixed value, independent of the model we are optimizing.

Maximum Entropy

If we do not possess any information on the relations of the samples and labels, we can use the maximum entropy prior as

$$\forall \mathbf{x} \in \mathcal{X}_u, k \in \mathcal{Y} : q_{me}(k|\mathbf{x}) = \frac{1}{K}. \tag{4.3}$$

While this prior might not be as informative as other priors which we introduce later, it makes it possible to link the boosting algorithm with the well-known *maximum entropy learning principle* [9]. Quoting from the famous work by Jaynes[51] which first stated the correspondence between the information theory and statistical mechanics

> Information theory provides a constructive criterion for setting up probability distributions on the basis of partial knowledge, and leads to a type of statistical inference which is called the maximum entropy estimate. It is least biased estimate possible on the given information; *i.e.*, it is maximally noncommittal with regard to missing information.

Knowledge Transfer, Co-Training, and Multi-View Learning

If we want to transfer the knowledge of a classifier (or a human), we can simply transform that knowledge into priors and use it in our model. By using probabilities, we can easily encode the underlying uncertainties with respect to the relation of samples and labels, without a need for explicit labeling of the samples.

As it will be discussed later in more details in Section 4.5, for a multi-view learning scenario where the data is represented by different views (or classifiers), one can use the information available in other views as a prior for training our model.

Cluster Priors

One way of applying the cluster/manifold assumption is to create a prior which encodes these assumptions directly. For example, we can use the unlabeled data to identify groups of similar samples (clusters) with respect to a similarity function $s(\mathbf{x}, \mathbf{x}')$. Then, we relate the clusters with possible labellings of them, by estimating the label density in each of these regions. This results in a set of prior conditional probabilities in form of $q(k|\mathbf{x})$.

Let $\mathcal{C} = \{c_1, \cdots, c_V\}$ be the clusters returned by the unsupervised clustering method using both labeled and unlabeled samples, and $s(\mathbf{x}, \mathbf{x}')$ a similarity function. We estimate the prior for all samples within a cluster as

$$\forall \mathbf{x} \in c_v : q_c(k|\mathbf{x}) = \frac{|c_v^k| + q(k)m}{\sum_{j=1}^{K} |c_v^j| + m}, \tag{4.4}$$

where $|c_v^k|$ is the number of samples from class k in cluster v, $q(k)$ is the label prior, and m is a positive number. Note that we use an M-estimation smoothing term in order to obtain a more robust estimate of the cluster priors [18]. If a cluster does not contain any labeled sample, we assign an equal probability to all classes for that partition. In practice, we can run the clustering algorithm a few times with different initial conditions and parameters, and average their results. The overall idea behind the cluster prior is shown in Figure 4.1 for a three class problem.

4.3 Loss Functions for Unlabeled Samples

Since in both regularization terms used in Eq. (4.2) the goal is to measure the deviations between two probabilities, it is natural to use loss functions which measure the divergence between two given distributions. In statistics community, there are varieties of such divergence measures.

In information theory, *entropy* measures the degree of uncertainty associated with a random variable. It can also be interpreted as the amount of average information absent, when we do not observe the value of the random variable. The entropy of a discrete random variable Z which can take a set of different values from the set $\mathcal{Z}$ is defined as

$$H(p) = -\sum_{z \in \mathcal{Z}} p(z) \log p(z), \tag{4.5}$$

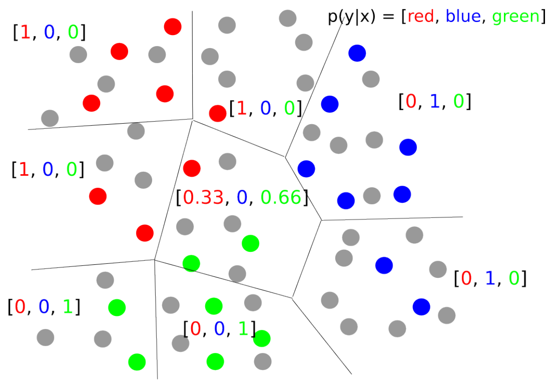
where $p(z)$ is the probability mass function over the random variable z. The entropy of a random variable is maximum, if $p(z)$ is a uniform distribution, *i.e.* $p(z) = \frac{1}{|\mathcal{Z}|}$.

The *cross-entropy* between two probability distributions measures the average uncertainty by using the probability distribution of $p(z)$ instead of the true distribution $q(z)$. This quantity is defined as

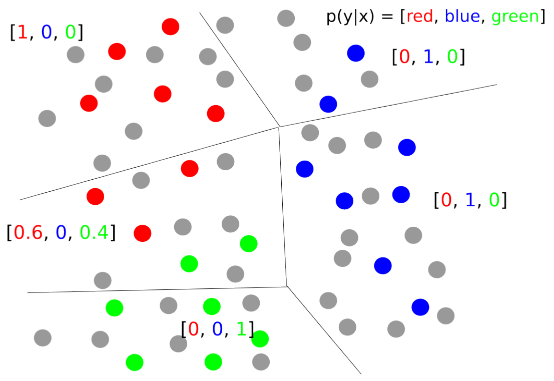
$$H(q, p) = -\sum_{z \in \mathcal{Z}} q(z) \log p(z). \tag{4.6}$$

Note that the minimum of the cross-entropy happens when two distributions q and p are the same, hence:

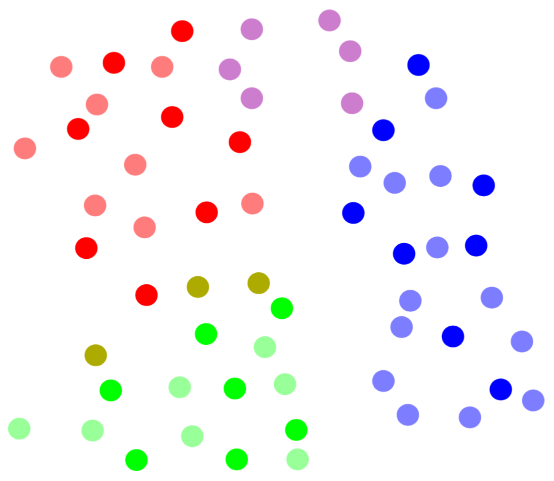
$$H(q, p) \geq H(q). \tag{4.7}$$



(a) Clustering #1



(b) Clustering #2



(c) Cluster prior

Figure 4.1: These plots show the idea behind cluster prior. This is a three class classification problem where classes are indicated by red, blue, and green colors. The grey dots are the unlabeled samples. In (a) and (b) a clustering algorithm with different number of cluster centers is applied to the data set. Depending on how many labeled samples are there in each cluster, a prior probability can be assigned to all the samples in a cluster. The final cluster prior, which is produced by smoothing the priors from all the clustering is shown in (c) where the color indicates the membership of the a sample to a class.

For measuring the divergence between two probability distributions, we use the following measures:

- Kullback-Leibler (KL) Divergence: It measures the expected amount of added uncertainty by using the probability distribution of $p(z)$ instead of the true distribution $q(z)$.
$$D_{KL}(q\|p) = H(q,p) - H(q). \tag{4.8}$$
Note that when two distributions q and p are the same, then $D_{KL}(q\|p) = 0$ since there is no added uncertainty.

- Symmertic KL Divergence: KL divergence is not symmetric with respect to p and q. However, one can use the following symmetrized version
$$D_{SKL}(q\|p) = \frac{1}{2}(D_{KL}(q\|p) + D_{KL}(p\|q)). \tag{4.9}$$

- Jensen-Shannon Divergence: This is also a symmetrized and smoothed version of KL divergence defined as
$$D_{JS}(q\|p) = \frac{1}{2}(D_{KL}(q\|m) + D_{KL}(p\|m)), \tag{4.10}$$
where $m = \frac{1}{2}(p+q)$.

Note that when we deal with priors, since q is independent of the model parameters $\boldsymbol{\beta}$, therefore, its entropy $H(q)$ is always equal to a certain quantity and hence can be safely omitted from the loss functions.

4.3.1 Prior Regularization

For the prior regularization term, we can directly use any of the loss functions we defined above, by using q as the prior and p as the posterior estimates of the model. However, there are some differences between these loss functions, which we discuss. In order to make the argumentation more clear, let us assume that we are dealing with a binary classification problem. In this case, $f_+(\mathbf{x}) = -f_-(\mathbf{x})$.

Figure 4.2 shows the plots of the loss functions defined in the previous section with respect to the confidence of the classifier for the positive class. As it can be seen, the KL and SKL divergences form a convex loss function, while the JS divergence is non-convex. By comparing these loss functions to the supervised loss functions we discussed

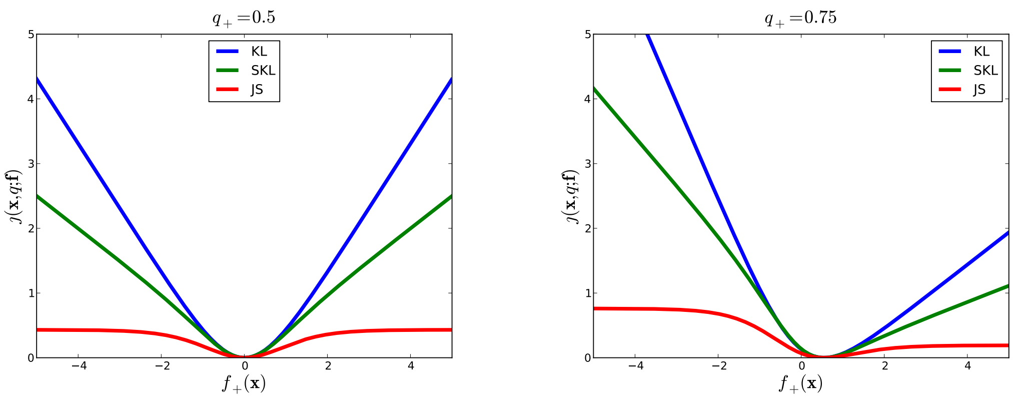


Figure 4.2: Different loss functions used for prior regularization with two different priors: (left) $q_+ = 0.5$ and (right) $q_+ = 0.75$.

in Section 3.3, one can see that KL and SKL resemble a behavior similar to the Logit loss, while JS is very similar to the Savage loss function.

Using this analogy, we expect that when the prior is noisy (which in most of the cases it is), the JS loss function to be more robust compared to KL and SKL divergences. In fact, one can visualize a scenario where the prior is relatively confident that a sample has a positive label (for example, $q = 0.75$ in the bottom plot of Figure 4.2). It can be seen that KL and SKL loss functions will force the classifier to move its prediction towards a point where they completely agree with the prior. However, the JS loss function has flat regions outside of the central valley. This means that the gradients in these regions will be very small or zero. Hence, if the classifier is not in agreement with the prior (for example, if its confidence is $f_+(\mathbf{x}) = -3$), then the algorithm will not force the classifier to move towards the prior. Effectively, if the prior has noise over this sample, and the classifier was confident enough regarding its prediction against what the prior claims, this loss function will give the chance for the classifier to stay confident about its prediction.

4.3.2 Manifold Regularization

We use the divergences discussed in previous section as the loss functions for the manifold regularization term as well. In details, let $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$ to be a neighboring sample of $\mathbf{x}$

using the $s(\mathbf{x}, \mathbf{x}')$ as the similarity measure. We define the manifold loss function as

$$\jmath_m(\mathbf{x}, \mathbf{x}'; \boldsymbol{\beta}) = D(p_{\mathbf{x}} \| p_{\mathbf{x}'}), \tag{4.11}$$

where D is a divergence measure, and $p_{\mathbf{x}}$ and $p_{\mathbf{x}'}$ are the posterior estimates for these two points. Note that depending on which measure we use, the same discussion presented in the previous section regarding the agreement and disagreement of the model and prior holds here as well.

4.4 Learning with Functional Gradient Descent

In order to incorporate these loss functions into the learning schemes presented in the previous chapter, we need to develop the individual gradient terms. Because the prior and manifold regularization terms result in different loss functions, we divide this section into two parts by computing the gradients for each of these terms. The details of the derivations can be found in Appendix B.

4.4.1 Learning with Multi-Class Classifiers

For the classification based learning using the multi-class base learners, we proved that the best choice for the label and the weight of a sample is given by Eq. (3.53) and Eq. (3.52) respectively. In the following theorem, we show that the weight of an example is positive or equals to zero. Algorithm 5 represents the learning procedure of the GPMBoost method.

Theorem 4.4.1. *For classification based semi-supervised boosting method of Algorithm 4, by utilizing a multi-class base learner with sample weights computed by the functional gradient descent, we have*

$$d_{\mathbf{x}} \geq 0. \tag{4.12}$$

Proof. Note that

$$d_{\mathbf{x}} = \max_{k \in \mathcal{Y}} d_{\mathbf{x},k}. \tag{4.13}$$

Since the sum of the weights $d_{\mathbf{x},k}$ is shown to be zero (refer to Appendix.B), therefore, either all weights $d_{\mathbf{x},k}$ are equal to zero, or if there are some non-zero weights, then their maximum is positive, as it is not possible that the sum of a set of negative terms is zero. □

```
Algorithm 5 Multi-Class Gradient, Prior, and Manifold-based Boosting (GPMBoost)
Require: Training labeled samples: $\mathcal{X}_l$.
Require: Training unlabeled samples: $\mathcal{X}_u$.
Require: Base learning algorithm: $g$.
Require: Prior for unlabeled samples: $q$.
Require: Number of boosting iterations: $M$.
Require: Shrinkage factor: $\nu$.
Require: Regularization factor: $\gamma$ and prior-manifold balancing factor: $\lambda$.
1: Set the initial model: $f_0(\mathbf{x}) = 0$.
2: for $m = 1$ to $M$ do
3:     Update the weights by Eq. (3.52).
4:     Update the pseudo-labels for unlabeled samples by Eq. (3.53).
5:     Train the $m$-th weak learner:
       $\boldsymbol{\theta}_m^* = \arg\min_{\boldsymbol{\theta}_m} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} w_{\mathbf{x}}\mathbb{I}(c(\mathbf{x}) \neq y) + \sum_{(\mathbf{x},\hat{y})\in\mathcal{X}_u} w_{\mathbf{x}}\mathbb{I}(c(\mathbf{x}) \neq \hat{y})$.
6:     Update the boosting model: $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu g(\mathbf{x}; \boldsymbol{\theta}_m^*)$
7: end for
8: Output the final classifier: $f(\mathbf{x}; \boldsymbol{\beta}_m^*) = \sum_{m=1}^{M} g(\mathbf{x}; \boldsymbol{\theta}_m^*)$.
```

4.5 Multi-View Learning

In this section, we focus on extending the algorithms presented in the previous chapter to multi-view learning scenarios. In multi-view settings, the data is represented in different views [1].

Based on our prior-based formulation, the *Co-training* style of semi-supervised learning is a special case of our model. In co-training, two classifiers are trained over different views of the same data samples. At the first iteration, they are only trained over the labeled samples. In each subsequent iteration of co-training, each of the classifiers labels a set of unlabeled samples, where its confidence is high and pass them to the other classifier for learning. Then each classifier is trained again over the set of the original labeled samples and those unlabeled samples which are labeled by the other classifier. This procedure of training and labeling continues for a certain amount of iterations or a suitable stopping criteria is met. Multi-view learning is an extension of the co-training where there is more than 2 classifiers (and views) involved.

[1]For clarity, we always use the co-training settings [11] where the data is represented by different views, while the algorithm can be applied to multiple-learners scenario as well [13].

We can extend the multi-view learning by using priors, instead of labels, as the medium for transferring the knowledge between different classifiers. This way, we can allow the uncertainties in labeling the unlabeled samples from each of the views to be encoded in the prior itself, hence, allowing the algorithm to be able to exploit the full extent of the multi-view learning.

Assume we have a multi-class semi-supervised classification task where the problem domain can be split into V different views. Let $\mathbf{x} = [\mathbf{x}_1^T | \cdots | \mathbf{x}_V^T]^T$ be a data sample which is constructed from V different views, each expressed by D_v-dim feature vector $\mathbf{x}_v \in \mathbb{R}^{D_v}$. In multi-view learning, we train a classifier per view $f_v(\mathbf{x}_v) : \mathbb{R}^{D_v} \to \mathbb{R}^K$ where K is the number of classes and $\mathcal{F} = \{f_v\}_{v=1}^V$ is the set of the classifiers. Let $p_v(k|\mathbf{x}_v)$ be the posterior estimate for classifying sample $\mathbf{x}_v$ in k-th class by the v-th learner. The goal of multi-view learning is to produce a set of classifiers which have low mis-classification rates over the labeled samples while having a high consensus over the unlabeled samples. One can express these goals as the following optimization problem

$$\mathcal{F}^* = \arg\min_{\mathcal{F}} \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \ell(\mathbf{x}, y; \mathcal{F}) + \gamma \sum_{\mathbf{x}\in\mathcal{X}_u} \phi(\mathbf{x}; \mathcal{F}). \tag{4.14}$$

The first term expresses the loss $\ell(\cdot)$ for the labeled samples where we have the true class label y, while the last term is a measure of the agreement of views over the unlabeled samples, and γ steers the effect of the unlabeled samples over the entire optimization problem.

Translating the traditional multi-view learning into learning with priors, the co-training style algorithms define the following $0-1$ prior

$$\forall \mathbf{x} \in \mathcal{X}_v : q_v^{0-1}(k|\mathbf{x}) = \begin{cases} 1 & \text{for} \quad k = \hat{y} \\ 0 & \text{for} \quad k \neq \hat{y} \end{cases}, \tag{4.15}$$

where $\hat{y}$ is the predicted label for the sample $\mathbf{x}$. The rest of the unlabeled samples which are not labeled are removed from the training set for this iteration. Note that based on the theorem we proved in Section 3.4, learning from such a prior will correspond to learning from labeled data by using the negative log-likelihood as the loss function.

Note that since our model is able to incorporate uncertainties, we can further extend the multi-view learning by replacing the $0-1$ priors by the posterior estimates from the previous training iteration. Therefore, we propose to use the posterior estimates for defining the loss over the unlabeled samples. Assume we have a function $\jmath(p\|q)$ for measuring the divergence between two probabilities p and q. Using this divergence

Algorithm 6 Multi-View Learning with Priors

```
1: For each view independently, optimize Eq. (4.14) by using Eq. (4.16) and using the
   Eq. (4.18) as the priors.
2: Label a subset of unlabeled samples and add them to the labeled set.
3: Compute the posteriors and update the priors for each view.
4: If stopping condition is not satisfied, proceed to the first step, otherwise stop.
```

measure, we express the unlabeled loss as $\phi(\mathbf{x};\mathcal{F}) = \sum_{v=1}^{V} d_v(\mathbf{x};\mathcal{F})$ and

$$\phi_v(\mathbf{x};\mathcal{F}) = \jmath(p_{\mathbf{x}_v} \| \frac{1}{V-1}\sum_{s\neq v} p_{\mathbf{x}_s}), \tag{4.16}$$

where $p_{\mathbf{x}_v} = [p_v(1|\mathbf{x}_v), \cdots, p_v(K|\mathbf{x}_v)]^T$. This loss function measures the divergence of the posterior estimates by computing the distance of each view to the average estimate of all other views. For example, if we use $\jmath(p\|q) = \sum_{k=1}^{K}(p(k|\mathbf{x}) - q(k|\mathbf{x}))^2$ the last term will measure the variance over different views. To see this, note that the variance of the v-th view can be computed as

$$\sum_{k=1}^{K}(\frac{1}{V}\sum_{s=1}^{V} p_s(k|\mathbf{x}_s) - p_v(k|\mathbf{x}_v))^2 = \frac{(V-1)^2}{V^2}\sum_{k=1}^{K}(\frac{1}{V-1}\sum_{s\neq v} p_s(k|\mathbf{x}_s) - p_v(k|\mathbf{x}_v))^2. \tag{4.17}$$

We use the *Jensen-Shannon Divergence* as $\jmath(p\|q)$ in our algorithm because of its robustness to noise. We will also refer to

$$q_v(k|\mathbf{x}_v) = \frac{1}{V-1}\sum_{s\neq v} p_s(k|\mathbf{x}_s),\ \forall v \in \{1,\cdots,V\}, k \in \mathcal{Y} \tag{4.18}$$

as the prior for the v-th view.

In order to observe the advantages gained by using this approach over the traditional multi-view learning, where the consensus is only encouraged by the iterative labeling of the unlabeled data, we propose the Algorithm 6.

If we set $\gamma = 0$ in this procedure, then we obtain a classical multi-view learning algorithm, similar to co-training [11]. Therefore, by observing the performance of both of these algorithms, one can see the gain obtained by incorporating the priors.

For the second step, where we label some unlabeled samples, we use the following approach: 1) Each view proposes the N highest confident samples to be included in the labeled set. 2) If there are disagreements over the label of a sample between some of the

Algorithm 7 MV-GPBoost

```
Require: Training labeled samples in multiple views: $\mathcal{X}_l = \{\mathcal{X}_l^v\}_{v=1}^V$.
Require: Training unlabeled samples in multiple views: $\mathcal{X}_u = \{\mathcal{X}_u^v\}_{v=1}^V$.
Require: Number of multi-view iterations: $N_{mv}$.
Require: Number of unlabeled samples to be labeled in each multi-view learning iterations: $N_l$.
 1: for $n_{mv} = 1$ to $N_{mv}$ do
 2:   for $v = 1$ to $V$ do
 3:     if $n_{mv} == 1$ then
 4:       Obtain cluster priors for this view: $q_v$.
 5:     end if
 6:     Train $v$-th view and obtain $f_v(\mathbf{x}_v; \boldsymbol{\beta}_v^*)$:
          $$\boldsymbol{\beta}_v^* = \arg\min_{\boldsymbol{\beta}} \sum_{(\mathbf{x}_v, y) \in \mathcal{X}_l^v} \ell(\mathbf{x}_v, y; \boldsymbol{\beta}) + \gamma \sum_{\mathbf{x}_v \in \mathcal{X}_u^v} \jmath^p(\mathbf{x}_v, q_v; \boldsymbol{\beta}). \qquad (4.19)$$
 7:     Compute the posteriors for $v$-th view: $\forall \mathbf{x}_v \in \mathcal{X}_u^v : p_v(\cdot|\mathbf{x}_v; \boldsymbol{\beta}_v^*)$.
 8:   end for
 9:   for $v = 1$ to $V$ do
10:     Update the priors: $\forall \mathbf{x}_v \in \mathcal{X}_u^v : q_{\mathbf{x}_v} = \frac{1}{V-1} \sum_{s \neq v} p_{\mathbf{x}_s}$.
11:   end for
12: end for
13: Output the final classifier: $f(\mathbf{x}) = \frac{1}{V} \sum_{v=1}^V f_v(\mathbf{x})$
```

views, we let the proposing views to vote with their confidence for the label of this sample, and we select the resulting class, which has the highest aggregated confidence. Again, if we would have only two views, this would be equivalent to the original co-training algorithm [11]. In the following sections, we will develop a semi-supervised multi-class boosting algorithm which can be used to solve the first step of this algorithm. Based on semi-supervised learning from priors, we propose the following Algorithm 6.

4.6 Remarks

The design of the semi-supervised boosting algorithm, GPMBoost, presented here was an evolutionary path consisting of a few different preliminary algorithms published over time at different scientific venues. In the following, we briefly describe the properties of these early versions and how they can be derived from the general algorithm we presented in this section.

4.6.1 SERBoost

SERBoost [85] was the first algorithm to be presented. This algorithm was designed for binary semi-supervised classification tasks (*i.e.*, $y \in \{-1, +1\}$) and only used the prior regularization part of GPMBoost: *i.e.*, $\lambda = 1$. SERBoost is an extension of the GentleBoost [35] to the semi-supervised learning domain. As the supervised loss function, it is using the exponential loss. For the unlabeled regularization term, we used the exponential version of the Kullback-Leibler divergence as

$$\jmath_{ekl}^{p}(\mathbf{x}, q; f) = e^{H(q,p)}. \tag{4.20}$$

The reason for this, was to make the computation of the labeled and unlabeled weights similar. To realize this, let $y_q = 2q(y = 1|\mathbf{x}) - 1 \in [-1, +1]$ to be the binary soft-label suggested by the prior. It can be easily shown that the Eq. (4.20) can be written as

$$\jmath_{ekl}^{p}(\mathbf{x}, q; f) = e^{-y_q f(\mathbf{x})} \cosh(f(\mathbf{x})), \tag{4.21}$$

where $f(\mathbf{x})$ is the confidence of the classifier for the positive class [2]. From this equation, one can immediately see the similarities with the exponential loss over the labeled samples $e^{-yf(\mathbf{x})}$. The derivation of the learning algorithm for SERBoost is done by the functional gradient descent technique [85].

4.6.2 RMSBoost

We extended the SERBoost algorithm to the multi-class classification in the RMSBoost algorithm [87]. The main differences, other than being applicable directly to multi-class problems, is that we used the standard Kullback-Leibler divergence for the regularization part. It was also in this work that we developed the *cluster prior* idea. Furthermore, RMSBoost has another component in its unlabeled regularization part, which encourages the maximization of the margin over the unlabeled samples [87]. Therefore, RMSBoost, which like SERBoost lacks the manifold regularization, can be considered as the second version of the GPMBoost algorithm, which we discussed in details throughout this chapter.

[2]From the symmetry condition of Eq. (3.6), we know that the $f_+(\mathbf{x}) = -f_-(\mathbf{x})$.

4.7 Experiments

We present an extensive set of experimental evaluations for the algorithms we have developed over time. We first show results for the SERBoost and RMSBoost algorithms. We continue with showing experimental results for our main algorithm, GPMBoost, where we compare with a large set of different semi-supervised methods over a large set of datasets.

4.7.1 SERBoost

We test the performance of SERBoost method on the challenging object category recognition data sets of Pascal Visual Object Class Challenge 2006 [31]. This dataset consists of 2615 training and 2686 test images coming from 10 different categories. In our experiments, we solve the multi-class problems with a 1-vs-all binary classification strategy.

In order to observe the effect of including the unlabeled data into the learning process of our boosting algorithm, we randomly partition the training set into two disjoint sets of labeled and unlabeled samples. The size of the labeled partition is set to be $r = 0.01$, 0.05, 0.1, 0.25, and 0.5 times of the number of all training samples. We repeat the procedure of producing random partitions for 10 times and report the average of the area under the curve (AUC) for each model.

Supervised Methods

Table 4.1 shows the performance of the supervised models: χ^2-SVM, Lin-SVM, Random Forest (RF), and GentleBoost (GB), trained over the full training set ($r = 1$), together with the average computation time. We also provide the performance of the winner of VOC2006 challenge [31] as a reference.

Comparing the performance of the different models, it is clear that the χ^2-SVM produces the best result, followed by Lin-SVM and GentleBoost, while Random Forest does not seem to be competitive. However, looking into the timings, the χ^2-SVM has considerably larger computation burden compared to all other methods. It should be noted that for the Random Forest, GentleBoost, and SERBoost methods we use our naive and unoptimized C++ implementation, while the other packages used for χ^2-SVM, Lin-SVM, and TSVM are heavily optimized regarding computation speed.

Figure 4.3(a) shows the performance of the fully supervised models with respect to the ratio of labeled samples in the training set. As expected, the χ^2-SVM is producing the best performance by paying the price of heavier computations. It is also clear that

Method	χ^2-SVM	Lin-SVM	RF	GB	QMUL_LSPCH
AUC	0.9243	0.8911	0.8456 ± 0.0025	0.8978 ± 0.0012	0.936
Time	885	82	98	116	-

Table 4.1: First row: the average AUC for the χ^2-SVM, Lin-SVM, Random Forest (RF), GentleBoost (GB) models, and the winner of VOC2006 (QMUL_LSPCH). Second row: the average computation time for each model in minutes.

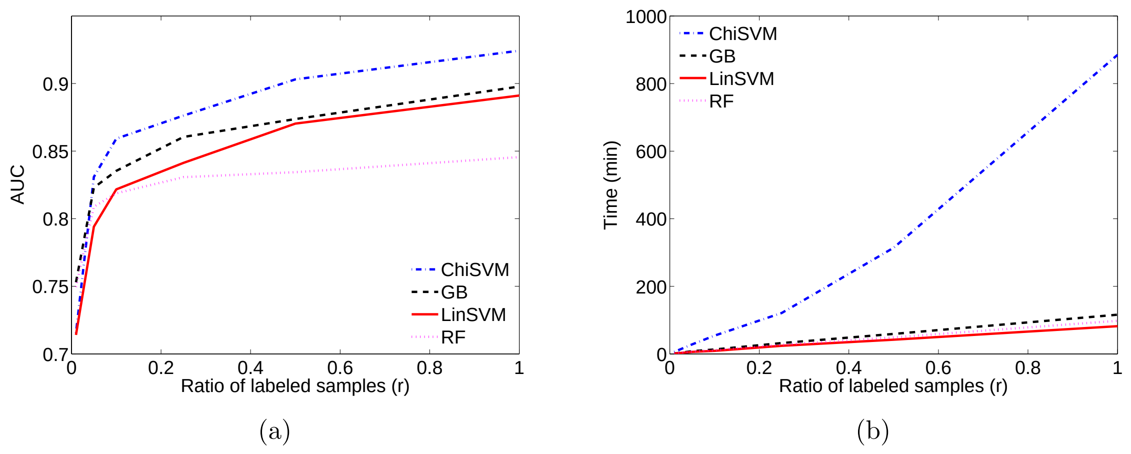
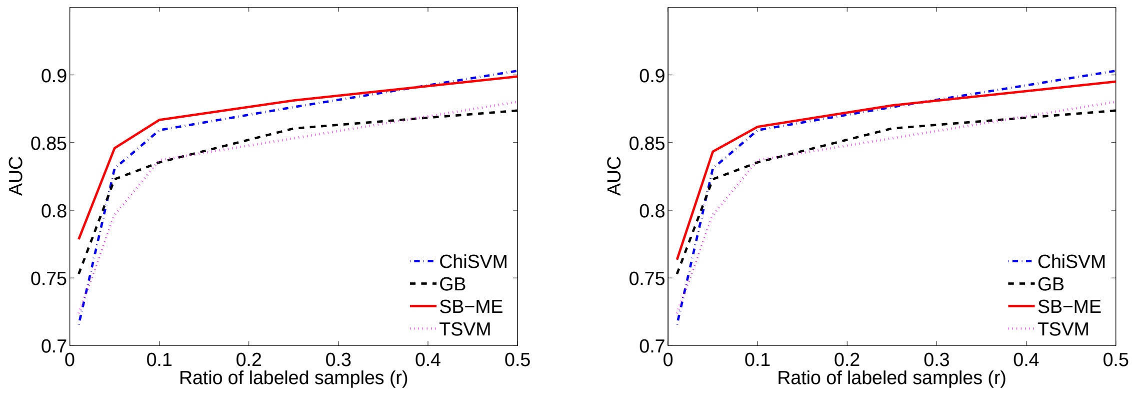


Figure 4.3: The performance (a) and computation times (b) of the χ^2-SVM, Lin-SVM, Random Forest (RF), and GentleBoost (GB) models with respect to the ratio of the labeled samples in training set.

the GentleBoost usually has a better or comparable performance compared to Lin-SVM.

Maximum Entropy Prior

We turn our attention to study the behavior of TSVM and our SERBoost models. Figure 4.4 shows the performance of SERBoost for two different values of unlabeled loss parameter $\gamma = 0.1, 0.25$ and when the maximum entropy prior is used. This figure also shows the performance of TSVM and the performance of χ^2-SVM and GentleBoost from Figure 4.3(a) as references. The first considerable observation is that SERBoost is performing better or comparable to χ^2-SVM even when there is no prior knowledge



(a) $\gamma = 0.1$ (b) $\gamma = 0.25$

Figure 4.4: The performance of SERBoost (SB) with maximum entropy prior (ME) for two different values of unlabeled loss parameter, γ.

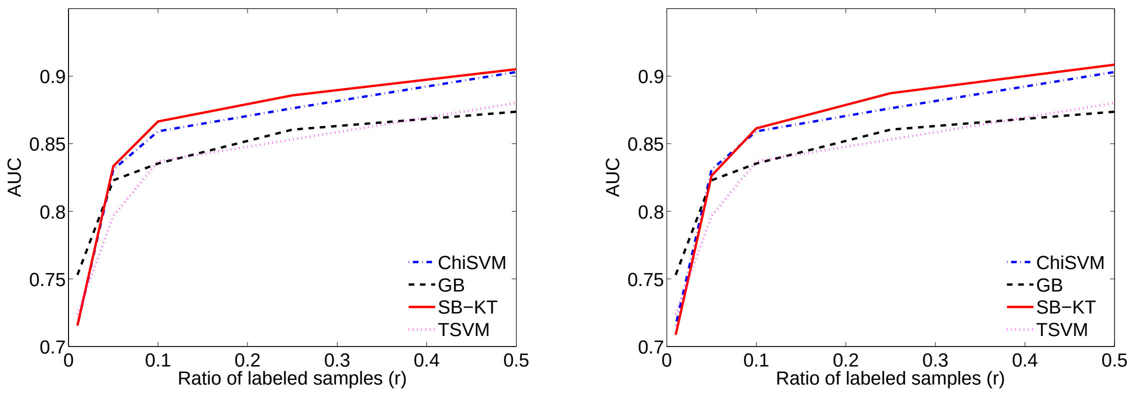
included in its learning process. As a matter of fact, SERBoost outperforms χ^2-SVM when the number of labeled images are very low, and as we continue to add more labels their performances become very close, and eventually after approximately $r = 0.5$, χ^2-SVM starts to perform better. It is also clear that TSVM is not competitive, neither in terms of performance, nor in terms of computation time with requiring 518 minutes for a single run. It should be noted that SERBoost has on average 14 minutes computation overhead compared to the GentleBoost.

Comparison to GentleBoost

From Figure 4.4, it is also clear that SERBoost opens a large gap compared to its fully supervised counter-part, GentleBoost. In order to investigate this fact, we conducted another set of experiments, where we duplicated the full training set and used the first half as the labeled set and the second half as the unlabeled partition. We applied SERBoost with maximum entropy prior to this data set and reported their results in Table 4.2. One can clearly observe that even with using the full training set with no additional information, SERBoost outperforms GentleBoost by a better margin in terms of AUC. It is also interesting to see that with this simple approach, SERBoost comes closer to the results of the winner of VOC2006.

Method	GB	SB, $\gamma = 0.01$	SB, $\gamma = 0.1$	SB, $\gamma = 0.25$
AUC	0.8978 ± 0.0012	0.9125 ± 0.0014	0.9153 ± 0.0016	0.914 ± 0.0015

Table 4.2: The performance comparison of GentleBoost (GB) and SERBoost (SB) when trained on the labeled training set.



(a) $\gamma = 0.1$ (b) $\gamma = 0.25$

Figure 4.5: The performance of SERBoost (SB) with prior knowledge (KT) for two different values of unlabeled loss parameter, γ.

Knowledge Transfer Prior

Since our method provides a principled way of including the prior knowledge as an additional source of information, we conduct experiments by training the χ^2-SVM over the labeled partition and use its predictions over the unlabeled partition as priors for SERBoost. The results are shown in Figure 4.5 for two different values of α. When the number of labeled samples are low, the predictions of the prior are mostly wrong, and therefore the performance of SERBoost is inferior to those trained with maximum entropy priors. However, as the χ^2-SVM starts to produce more reliable predictions, our method also starts to improve its performance. As it can be seen, by moving towards larger labeled sets, our method utilizes the priors well and outperforms the maximum entropy based model.

Hyperparameter Sensitivity

Another considerable fact is the robustness of SERBoost with respect to the variations of γ within a reasonable working range. If we compare the pairs of left and right plots in Figures 4.4 and 4.5, we can see that the performance changes smoothly when one varies γ. For example in Figure 4.5, one can see that when χ^2-SVM predictions are not reliable (lower values of r), having a smaller γ results in a slight performance gain, while the figure is reversed when the χ^2-SVM starts to operate reasonably. However, the overall change in performance is not significant.

4.7.2 RMSBoost

In order to evaluate the RMSBoost algorithm, we conduct experiments on two machine learning datasets as well as challenging object category recognition dataset on Pascal VOC2006 [31]. The main goal of these evaluations is to compare our method with other semi-supervised methods which are proposed for large scale datasets. Note that the VOC2006 dataset offers a challenging benchmark, where a simple representation already achieves good results without the need for complicated feature tunings. For sanity check, we also show the performance of the state-of-the-art supervised algorithms.

In these experiments, we compare to the following methods: 1) *AdaBoost.ML* [115]: a multi-class boosting algorithm based on minimizing the Logit loss function. We will refer to this method as AML in the experiments. 2) *Kernel SVM*: for machine learning datasets we use the RBF kernel, while for object category recognition, we use the pyramid χ^2 kernel. 3) *MS-TSVM* [95]: Multi-Switch TSVM is probably the fastest version of the popular TSVM. 4) *SERBoost* [85]: a semi-supervised boosting algorithm based on the expectation regularization. We will denote this method as SER. 5) *RM-Boost*: the supervised version of our method (when $\gamma = 0$). We will refer to this method as RMB.

RMSBoost has two components in the unlabeled regularization part: 1) the Kullback-Leibler regularization for using the prior, and, 2) the unlabeled margin maximization term. The effect of the first term is tuned by γ parameter, while β modifies the effect of the second term.

We apply the 1-vs.-all strategy to those methods which are not inherently multi-class. Preliminary evaluations of our method showed that setting $\beta = 0.1\gamma$ produces acceptable results. Thus, we use this setting for all experiments. We perform a 5-fold cross-validation to select the rest of the hyperparameters for all methods. We set the number of iterations T to be 10000 for all boosting methods and use extremely randomized forests [37] with 10 trees as weak learners. In order to obtain the cluster

Dataset	# Train	# Test	# Class	# Feat.
Letter	15000	5000	26	16
SensIt (com)	78823	19705	3	100

Table 4.3: Data sets for the machine learning experiments.

prior, we run the *hierarchical kmeans* clustering algorithm 10 times by setting the number of clusters to be $50K$ (where K is the number of classes) and average the prior for each sample. We also set the smoothing factor of the probability estimates to be $m = 50$.

Machine Learning Datasets

We use the *Letter* and *SensIt* datasets from the LibSVM repository [19]. A summary of these sets is presented in Table 4.3. We randomly partition the original training set into two disjoint sets of labeled and unlabeled samples. We randomly select 5% of the training set to be labeled and assign the rest (95%) to the unlabeled set. We repeat this procedure 10 times and report the average classification accuracy in Table 4.4. As can be seen from this table, our method achieves the best results over these datasets compared to all other methods. Table 4.4 also shows the average computation time for these methods. Since our method processes 20 times more unlabeled data on these datasets, it is slower than the supervised boosting methods. However, compared to the other semi-supervised methods, our method is faster in the presence of large amounts of data.

We also examined the relative contribution of each of the unlabeled regularizer terms. On the Letter dataset, using only the cluster regularizer results in 78.7% classification accuracy, while using only the margin term produces 75.1%. However, using both terms we can see that the performance is boosted to 79.9%. Similar to TSVMs, the margin term resembles a kind of self-learning strategy. Thus, its performance depends highly on the quality of the overall classifier. Therefore, in our case, the cluster prior helps to produce a boosted classifier, while the margin term helps to improve the decision margins.

VOC2006 Dataset

For the VOC2006 dataset, we follow a fairly standard bag-of-words approach by extracting the SIFT descriptors on a regular dense grid of size 8 pixels at multiple scales of 8,

Method	AML	SVM	TSVM	SER	RMB	**RMSB**
Letter	72.3	70.3	65.9	76.5	74.4	**79.9**
SensIt	79.5	80.2	79.9	81.9	79.0	**83.7**

Table 4.4: Classification accuracy (in %) for machine learning datasets. The RMSB stands for our method.

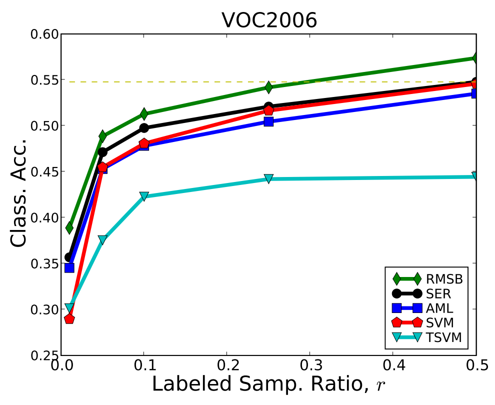
Method	AML	SVM	TSVM	SER	RMB	RMSB
Letter	22	25	74	3124	21	125
SensIt	28	195	687	1158	27	514

Table 4.5: Computation (train+test) time (in seconds) for machine learning datasets.

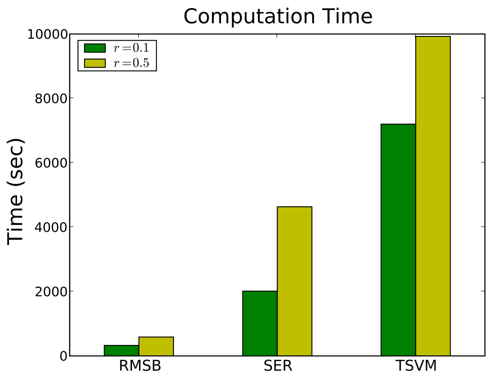
16, 24, and 32 pixels. We find the class-specific visual vocabulary by randomly selecting descriptors from 10 training images of each class, and forming 100 cluster centers using k-means. The final vocabulary is the concatenation of all class-specific cluster centers. We use the L_1-normalized 2-level spatial pyramids [55] to represent each image, and as a result, the feature space is 5000 dimensional. Note that since the VOC2006 presents a multi-label classification problem (some images contain more than one object), we duplicate the multi-label samples of the training set and assign a single label to each of them. Also during the test phase, we assign a correct classification, if at least one of the labels is predicted correctly. We should emphasize that for the VOC2006 challenge, the binary AUC was the performance measure. However, it is not trivial to extend the ROC curve analysis of binary classification to the multi-class problems directly. Therefore, we decided to use the other natural alternative which is the classification accuracy.

We randomly partition the training set of VOC2006 dataset into two disjoint sets of labeled and unlabeled samples. The size of the labeled partition is set to be $r = 0.01$, 0.05, 0.1, 0.25, and 0.5 times the number of all training samples. We repeat the procedure 10 times and measure the average classification accuracy over the test set. Note that for these experiments we do not show the results for the supervised version of our method as its performance is similar to that of Adaboost.ML.

Figure 4.6(a) shows how the classification accuracy evolves when the number of labeled samples changes in the training set. As can be seen, our model achieves the highest accuracy compared to others. The dashed yellow line shows the best results obtained by other methods (SERBoost at $r = 0.5$). We can see that our method surpasses the accuracy of the SERBoost by using only half of the labeled samples.



(a) Classification accuracy



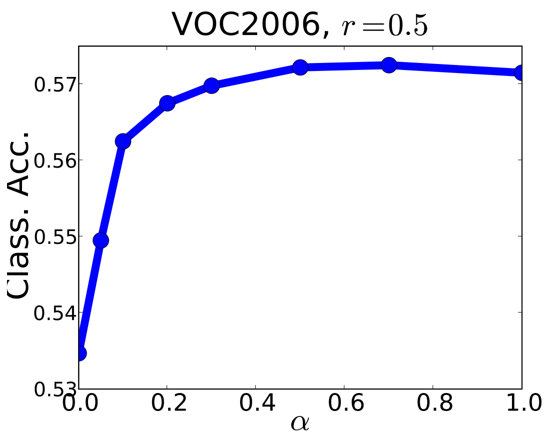
(b) Timings

Figure 4.6: (a) Classification accuracy with respect to the ratio of labeled samples. (b) Computation (train+test) time for two different ratios of labeled samples for semi-supervised methods.

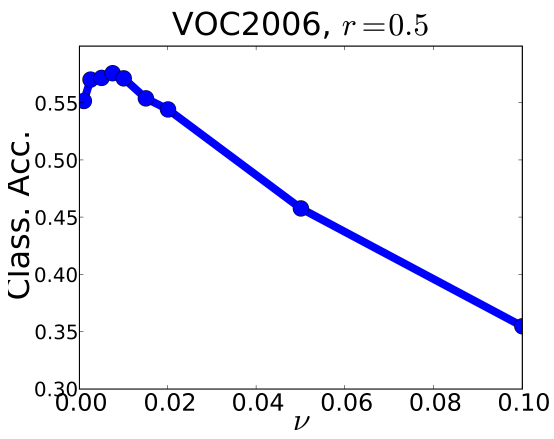
Our method not only obtains the best overall results, but is also the fastest compared to the other semi-supervised algorithms. This can be seen in Figure 4.6(b), where the computation (training and test) time is presented for two different ratios of labeled samples in the training set. Thus, using proper regularization and solving the multi-class problem directly, is essential in reducing the computation time.

Figure 4.7(a) shows how the accuracy changes with respect to γ. As it can be seen, the performance does not vary considerably for a large range of γ values. We found that setting γ to the ratio of labeled samples, *i.e.*, $\gamma = \frac{N_l}{N_l+N_u}$, often produces acceptable results. Figure 4.7(b) presents the effects of the shrinkage factor ν. Here, it becomes clear that selecting a proper value for ν is essential in order to obtain a good result. This is no surprise as, it is an established fact that selecting the proper step size for the gradient-based optimization methods is important for the robustness of the overall optimization procedure.

Since we perform gradient descent by using multi-class classifiers, it is interesting to see how successful the optimization process is. Figure 4.8(a) plots the classification accuracy (blue), the average gradients for the labeled (green) and unlabeled (red) samples as a function of the boosting iterations. After an initial rise, the accuracy slowly improves over time. Accordingly, the gradients for the labeled samples also continue to converge to very small values, which shows that the optimization over the labeled samples is also



(a) Gamma



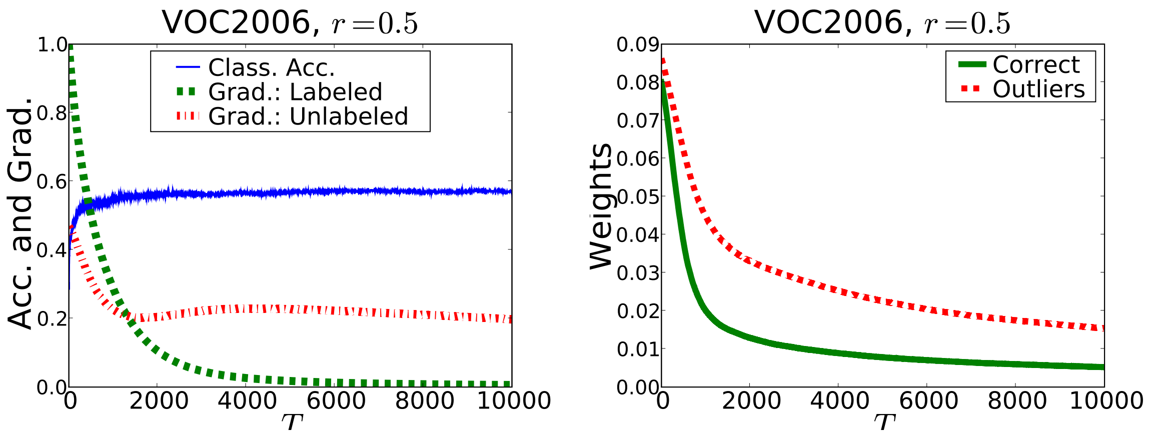
(b) Shrinkage

Figure 4.7: The effects of changing (a) γ, and (b) the shrinkage factor ν on the classification accuracy.

successful. However, the gradients for the unlabeled samples converge to a relatively small value. Further investigations of the weights and the cluster prior seem to clarify this behavior. Figure 4.8(b) shows the average weights of unlabeled samples in two groups: 1) those shown in green, where the prior is correct about their true (hidden) label, and 2) those shown in red, where the prior is wrong. In this example, the correct number is about 582, while the number of outliers is almost two times bigger: 1091. Therefore, we can see that our algorithm is able to resist learning these outliers, to some extent, and hence, there is always a residual error in the unlabeled loss with respect to these samples. This shows that the optimization procedure is successful in producing a balance between learning from the labeled data and from a very noisy (and mostly wrong) prior.

4.7.3 GPMBoost

In order to access the performance of the GPMBoost algorithm, we conduct several experimental evaluations. We first compare our algorithm with existing supervised/semi-supervised boosting and SVM based learning methods. Since the recently proposed semi-supervised boosting methods have different experimental settings, we split these experiments into three different parts following the same settings for each of these meth-



(a) Class. Acc.

(b) Weights

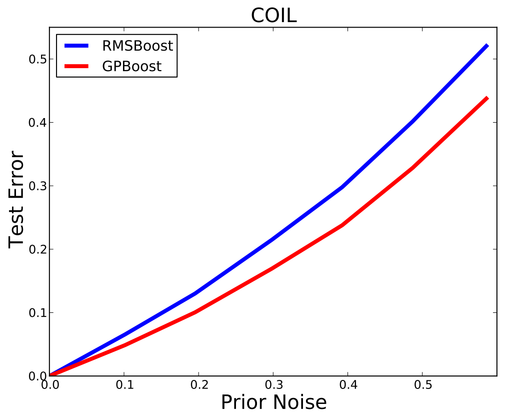
Figure 4.8: (a) Classification accuracy (blue), and average gradients for labeled (green) and unlabeled (red) samples versus the number of iterations. (b) The average weights for the unlabeled samples, where the prediction of the prior was correct (green) or wrong (red).

ods. As prior, we use the cluster prior, as it provides complementary information with respect to the manifold regularization term. Our software framework, which is a mix of Python and C++ implementations of GPMBoost algorithm is freely available from the following URL[3].

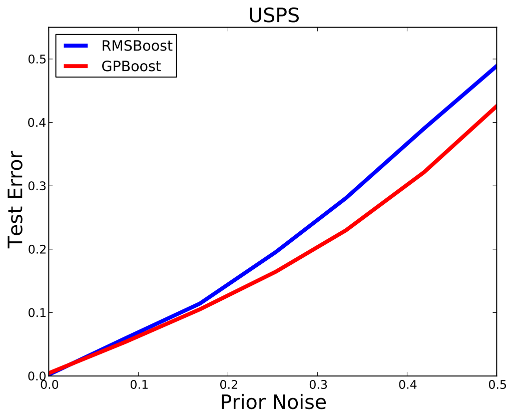
Robustness Experiments

In order to show the increased robustness of the GPMBoost by using the Jensen-Shannon loss function, we compare our semi-supervised boosting algorithm (GPMBoost) with the RMSBoost [87] which uses the Kullback-Leibler divergence, by setting $\lambda = 1$ (only using the prior regularization). In order to observe the robustness of each of these methods, in these experiments, we introduce random noise into the prior and train both semi-supervised boosting algorithms with the same settings. In order to make the comparison fair, we also change the supervised loss function of the RMSBoost to Savage loss. For these experiments, we choose two semi-supervised learning benchmark datasets [20]. The results averaged over 12 splits provided in the dataset for COIL and USPS datasets are

[3]http://www.ymer.org/amir/software/multi-class-semi-supervised-boosting/



(a) COIL



(b) USPS

Figure 4.9: Classification error for (a) COIL and (b) USPS dataset with noisy priors.

shown in Figure 4.9. As it can be seen, our algorithm retains lower test errors compared to the RMSBoost. It should be noted that specially for the COIL set which is multi-class dataset, the gap is larger from early on.

Machine Learning Benchmarks

For all these experiments, we use the following setup. As weak learners, we use small Random Forests (RF) with 5 random trees. RFs provide a very fast, efficient, and inherently multi-class weak learners. We fix the shrinkage factor of the boosting to $\nu = 0.02$ and set the boosting iterations to $T = 200$. We also use a fixed $\gamma = 0.1$. We conduct model selection procedures for choosing the prior-manifold trade-off parameter λ over the set $\{1.0, 0.75, 0.5, 0.25, 0.0\}$. For datasets which have a very small number of labeled samples (*e.g.* 10), we use a leave-one-out method, while for the rest we use a 5-fold cross validation. We perform the model selection for each dataset using a randomly chosen split and use the resulting hyperparameters for all the splits.

For the cluster prior, we train the *KMeans* clustering algorithm for 50 times and average their priors. The number of cluster centers is chosen randomly from the range $[K, 2K]$ where K is the number of classes.

For the similarity $s(\mathbf{x}, \mathbf{x}')$ function, we first form the adjacency matrix

$$a(\mathbf{x}, \mathbf{x}') = \mathbb{I}(\mathbf{x}' \in \mathcal{N}(\mathbf{x})), \tag{4.22}$$

where $\mathcal{N}(\mathbf{x})$ is the set of the 5 nearest-neighbors of the sample $\mathbf{x}$ based on the Euclidean distance. Then the similarity is computed as:

$$s(\mathbf{x}, \mathbf{x}') = \frac{1}{2}(a(\mathbf{x}, \mathbf{x}') + a(\mathbf{x}', \mathbf{x})). \tag{4.23}$$

This way we have a similarity measure which encodes the neighborhood of the data samples and is also symmetric.

SSL Benchmarks In these set of experiments, we compare our method with various supervised and semi-supervised learning methods using the standard semi-supervised benchmark datasets from the book of Chapelle *et al.* [20]. The recent method of Loeff *et al.* [63] named ManifoldBoost also follows the same experimental settings, and hence, these experiments allow us to compare to their algorithm as well. We additionally implemented the SemiBoost [66] and MCSSB [102] algorithms and report results for these algorithms using RFs as weak learners. These algorithms use the same weak learners as the GPBoost method, and all the other experimental settings are adjusted as suggested by their authors.

These datasets are organized in transductive setting, which allows tests to be conducted for both inductive and transductive algorithms. Therefore, the set of unlabeled samples in these experiments also serve as the test set. For each dataset, 10 and 100 samples are randomly selected as labeled samples, and the rest are used as unlabeled and test sets. There are 12 independent splits for each dataset. In these experiments, we compare with the following supervised algorithms: 1-NN, SVM with RBF kernel, RFs (the weak learners), supervised version of our algorithm, named *GBoost* (Gradient Boosting). For semi-supervised methods, we use the following inductive algorithms: TSVM with RBF kernel [52], SVM with RBF+Cluster Kernel [105], LapSVM with RBF kernel [7], SemiBoost [66], and MCSSB [102]. These widely used semi-supervised algorithms cover the spectrum of the inductive cluster and manifold based methods.

Table 4.6 and Table 4.7 show the classification error over the test (unlabeled) set, averaged over 12 splits. For methods based on RFs, we additionally run each algorithm 5 times per split. As it is expected, our algorithm has a relatively good performance over both cluster and manifold-like datasets. In fact, it outperforms other methods 6 out of 12 times, while is the second best in the 2 other cases. Compared to the ManifoldBoost method, our algorithm achieves overall better error rates, except for the BCI dataset, which our algorithm fails to keep up with other methods. Compared to the performance of the supervised version of our algorithm (GBoost), we can see that in 10 out of 12 cases

Methods-Dataset	g241c	g241d	Digit1	USPS	COIL	BCI
1-NN [20]	44.05	43.22	23.47	19.82	65.91	48.74
SVM [20]	47.32	46.66	30.60	20.03	68.36	49.85
RF (weak)	47.51	48.44	42.42	22.90	75.72	49.35
GBoost-RF	46.77	46.61	38.70	20.89	69.85	49.12
TSVM [20]	*24.71*	50.08	17.77	25.20	67.50	49.15
Cluster Kernel [20]	48.28	*42.05*	18.73	19.41	67.32	*48.31*
LapSVM [20]	46.21	45.15	**08.97**	19.05	N/A	49.25
ManifoldBoost [63]	42.17	42.80	19.42	19.97	N/A	**47.12**
SemiBoost-RF	48.41	47.19	*10.57*	*15.83*	*63.39*	49.77
MCSSB-RF	49.77	48.57	38.50	22.95	69.96	49.12
GPMBoost-RF	**15.69**	**39.45**	11.19	**14.92**	**62.60**	49.27

Table 4.6: Classification error on semi-supervised learning benchmark datasets for 10 labeled samples. The best performing method is marked in **bold-face** font, while the second best is shown in *italic*.

we reduce the test error significantly. This confirms the success of the semi-supervised learning and the optimization process.

Comparison with SemiBoost In this section, we use a set of benchmark datasets used in the recent SemiBoost [66] paper. We follow the same experimental settings in the paper by splitting the dataset into 50% training/test sets, and randomly labeling 10 samples from the training set. This procedure is repeated 20 times and the average classification error is reported. The settings for our method is the same as the previous section, and the model selection is performed by the leave-one-out strategy.

Table 4.8 shows the results[4]. Note that the results are different to the previous section, as the dataset is now split to two different train and test sets. We also report the results of SemiBoost and MCSSB algorithms with RFs as weak learners. Additionally, the results for an inductive version of the Low Density Separation (LDS) [22] is also presented. We report the results of SemiBoost with 3 different weak learners: Decision Stumps (DS), Decision Trees (Tree), and Linear SVM (SVM) from [66].

The first observation is that the performance of our implementation of SemiBoost with RFs is generally better than all other results reported in [66]. In fact, by looking at the average classification error over all datasets (the last column), SemiBoost-RF

[4]In SemiBoost experiments reported in [66], they converted the multi-class datasets into binary by only considering two classes with the majority of the samples.

Methods-Dataset	g241c	g241d	Digit1	USPS	COIL	BCI
1-NN [20]	40.28	37.49	06.12	07.64	23.27	44.83
SVM [20]	23.11	24.64	05.53	09.75	22.93	34.31
RF (weak)	44.23	45.02	17.80	16.73	34.26	43.78
GBoost-RF	31.84	32.38	06.24	13.81	21.88	40.08
TSVM [20]	18.46	22.42	06.15	09.77	25.80	33.25
Cluster Kernel [20]	*13.49*	**04.95**	03.79	09.68	21.99	35.17
LapSVM [20]	23.82	26.36	*03.13*	**04.70**	N/A	*32.39*
ManifoldBoost [63]	22.87	25.00	04.29	06.65	N/A	**32.17**
SemiBoost-RF	41.26	39.14	**02.56**	*05.92*	*15.31*	47.12
MCSSB-RF	45.11	40.26	13.19	12.31	31.09	47.64
GPMBoost-RF	**12.80**	*12.59*	**02.35**	06.33	**14.49**	45.41

Table 4.7: Classification error on semi-supervised learning benchmark datasets for 100 labeled samples. The best performing method is marked in **bold-face** font, while the second best is shown in *italic*.

is considerably better than SemiBoost-SVM. Comparing GPMBoost and SemiBoost,we can see that their performances are very close in these experiments. However, the classification error of GPMBoost is always better or comparable to the SemiBoost. This can be realized by considering the results of these two algorithms over all the datasets, together with the average classification error. In fact, on 3 datasets (g241n, BCI, Austra) GPMBoost has a good advantage over the SemiBoost-RF, while on all other datasets they achieve similar errors.

Comparison with MCSSB MCSSB [102] is a recently proposed multi-class semi-supervised boosting algorithm. We compare with this algorithm on 3 datasets following the same experimental settings as applied in the original paper. We choose datasets which have a separate train and test splits and have a relatively high number of data samples. We randomly choose 5% and 10% of the training samples to be labeled and use the rest as unlabeled samples. We repeat this procedure for 20 times and report the average mis-classification rates. Model selection is conducted as before, by making a 5-fold cross validation to select the hyperparameters of our algorithm.

Table 4.9 shows the results. This table contains the results for the ASSEMBLE method [8], which is one of the first semi-supervised boosting methods relying on the cluster assumption. We report the results from [102] for ASSEMBLE and MCSSB with 2 different weak learners: Decision Trees (Tree), and Multi-Layer Perceptron (MLP). We

Methods-Dataset	g241n	Digit1	COIL2	BCI	Austra	Optdigits	SatImage
SVM [66]	42.45	25.19	40.25	*47.55*	34.36	09.69	*00.87*
RF (weak)	46.99	35.40	46.35	49.25	24.01	12.50	10.17
GBoost-RF	45.69	28.78	43.21	49.55	*21.12*	05.46	09.91
TSVM [66]	48.86	*20.48*	49.77	49.50	26.62	07.66	N/A
ILDS [66]	49.75	20.47	45.38	49.27	34.00	03.60	05.80
LapSVM [66]	46.35	25.94	44.36	**45.63**	35.62	*01.66*	*00.88*
SemiBoost-DS [66]	45.45	21.91	44.16	50.62	36.54	06.88	14.01
SemiBoost-Tree [66]	45.29	25.03	44.73	49.33	36.64	06.77	13.45
SemiBoost-SVM [66]	*42.07*	22.11	44.56	47.98	28.64	03.65	12.29
SemiBoost-RF	45.71	**12.60**	*41.23*	49.15	22.47	**00.94**	**00.24**
MCSSB-RF	46.23	25.12	41.69	49.38	37.07	11.16	18.56
GPMBoost-RF	**41.11**	**12.21**	**38.71**	48.85	**18.84**	**00.59**	**00.49**

Table 4.8: Classification error on semi-supervised learning benchmark datasets compared to SemiBoost. The best performing method is marked in **bold-face** font, while the second best is shown in *italic*.

additionally present the SemiBoost and MCSSB algorithms with RFs as weak learners. For SemiBoost algorithm, we perform a 1-vs-all classification strategy.

From these results, again one can see that, our implementation of MCSSB algorithm with RFs as weak learners provides a large improvement compared to the results reported in [102]. For example, in the case of Optdigits dataset, the error of the MCSSB has an improvement from around 70% error to around 6%. Comparing the 3 semi-supervised boosting algorithms, we can see that over the multi-class dataset, the performance of the GPMBoost is considerably better than the other algorithms, while SemiBoost is the second closest algorithm.

Discussion This section included an extensive set of evaluations which involved 25 distinctive experiments over binary and multi-class benchmark datasets, with more than 2000 runs for boosting algorithms based on RFs. From these results, we can see that GPMBoost is a very successful and flexible algorithm, which is able to operate in a wide range of semi-supervised learning problems, due to its ability to use both cluster and manifold structures of the feature space. The results show that, compared to other boosting based algorithms considered here, GPMBoost generally has a better or comparable performance. Specially, for the multi-class datasets we can see that GPMBoost has a much better classification accuracy: it outperforms other methods in 6 out of 8

Methods-Dataset	Optdigits		Pendigits		Segmentation	
# Labels Ratio	5%	10%	5%	10%	5%	10%
Tree [102]	67.00	66.10	66.20	63.50	52.40	51.50
MLP [102]	76.70	74.60	70.00	68.30	56.80	53.70
RF (weak)	21.00	16.10	13.50	10.10	31.20	23.80
GBoost-RF	08.90	06.60	09.20	06.75	*29.40*	**19.52**
Assemble-Tree [102]	69.40	69.10	67.50	66.00	54.10	55.60
Assemble-MLP [102]	76.70	72.50	70.20	68.70	55.20	57.10
MCSSB-Tree [102]	67.00	66.10	40.70	42.30	52.40	51.50
MCSSB-MLP [102]	79.10	72.40	45.30	47.80	55.50	53.20
SemiBoost-RF	**03.96**	*03.62*	*06.19*	*05.11*	33.02	28.16
MCSSB-RF	06.30	05.42	09.11	06.80	34.71	25.04
GPMBoost-RF	**04.01**	**03.14**	**05.26**	**04.20**	**25.70**	*20.21*

Table 4.9: Classification error on UCI datasets for 5% and 10% labeled samples. The best performing method is marked in **bold-face** font, while the second best is shown in *italic.*

cases, and draws with SemiBoost on 1 dataset. For the binary problems, it performs considerably well by outperforming other methods in 7 out of 17, and drawing in 4 other cases with the best results obtained by SemiBoost. In fact, we can conclude that there is no single algorithm which performs as consistant as GPMBoost. Considering all the boosting algorithms, we can see that small RFs provide a very fast weak learning algorithm suitable for using inside the boosting algorithms. Another outcome from these results is that for binary problems, SemiBoost and LapSVM are the closest performing algorithms to GPMBoost, while for multi-class problems the 1-vs-all implementation of SemiBoost is competitive as well.

4.7.4 MV-GPBoost

Object Category Recognition

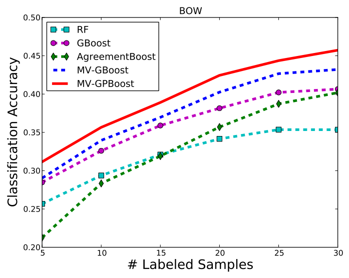
We evaluate various multi-view semi-supervised boosting algorithms on *Caltech101* object category recognition task. This dataset represents a challenging task for the semi-supervised learners, since the number of classes is large and the number of training samples per class is rather low. For these experiments, we randomly choose up to 80 images from each class and label $\{5, 10, 15, 20, 25, 30\}$ images for each of them. We use the rest of the samples as the test set. Since many of the classes do not have enough images to form a separate unlabeled set, we resort to the transductive settings where the test set is used as the unlabeled set. We repeat this procedure 5 times and report the average classification accuracy per class.

For feature extraction, we use the precomputed dense SIFT-based bag-of-words and PHOG features from Gehler and Nowozin [36] to form different views. In details, for BOW features, we use a vocabulary of size 300 extracted from gray level and individual color channels. We use a level-2 spatial histogram to represent these 2 views (BOW-grey and BOW-Color). Additionally, we use level-2 PHOG features formed from the oriented (PHOG-360) and unoriented (PHOG-180) gradients. Therefore, in total we have 4 different views for this dataset.

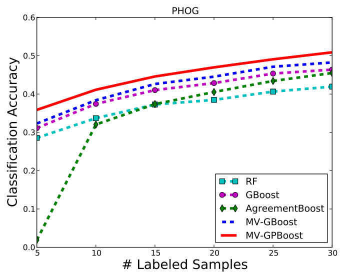
In these experiments, we use the Random Forests (RF), our supervised multi-class boosting algorithm (GBoost), multi-view boosting using GBoost as the basic learners (MV-GBoost), and our multi-view algorithm MV-GPBoost. Additionally, we extended the AgreementBoost algorithm [59] to cope with multi-class problems and report the results for this algorithm as well.

If we set $\gamma = 0$ in MV-GPBoost, we will end up exactly with the MV-GBoost algorithm. Therefore, the performance gains seen here are totally due to the incorporation of the prior regularization term. The settings for the RFs and our boosting algorithms are exactly the same settings used for machine learning benchmark experiments. For the multi-view algorithms, we iterate the learning process for 10 iterations and label 100 unlabeled samples (1 from each class) in each iteration. Since RFs and GBoost cannot use the views directly, we concatenate the features into a single feature vector.

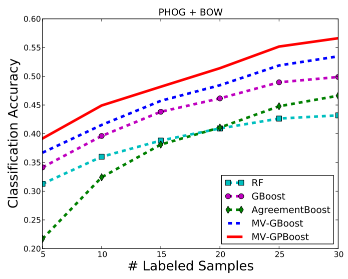
Figure 4.10 shows the results for three different settings: (a) only using the two views provided from BOW features, (b) using two views from PHOG features, and (c) using all 4 views. The first observation is that the GBoost algorithm successfully boosts the performance of the random forest and the accuracy gap can be as high as 5%. Comparing the performance of GBoost and MV-GBoost, we can see that in general, the multi-view learning strategy by labeling a subset of unlabeled samples iteratively, works and there is a clear performance gain between these two algorithms. However, the highest accuracy is obtained by MV-GPBoost, which has a considerable gap in classification accuracy compared to the MV-GBoost algorithm. Another observation here is that, as expected, the combination of all 4 views achieves the highest performance, compared to using either two views from BOW or PHOGs. Furthermore, the performance of the AgreementBoost, which uses the variance of the classifiers over different views to regularize the training process of boosting, is not comparable to the performance of other learning methods. Finally, it should be noted that the objective in these experiments was to show that, with ordinary classifiers such as RFs, we are able to gain considerable performance gains. The



(a) BOW views: grey and Color



(b) PHOG views: 360 and 180



(c) PHOG and BOW views

Figure 4.10: Caltech101 classification accuracy for: (a) BOW, (b) PHOG, and (c) BOW and PHOG.

state-of-the-art results at the moment for Caltech101 dataset is using a large combination of kernels for SVMs in a multi-class boosting framework [36]. Therefore, we expect that, by utilizing kernel SVMs as basic building blocks in our boosting framework, similar results can be achieved.

Object Tracking

Recently, boosting-based methods have achieved high accurate tracking performances running in real-time [3]. In these methods, usually an appearance-based classifier is trained with a marked object at the first frame versus its local back-ground. The object is then tracked by performing re-detection in the succeeding frames. In order to handle rapid appearance and illumination changes, the classifiers perform on-line self-updating [40]. However, during this self-updating process it is hard to decide where to select the positive and negative updates. If the samples are selected wrongly, slight errors

Approach	*sylv*	*david*	*faceocc2*	*tiger1*	*tiger2*	*coke*	*faceocc1*	*girl*
MV-GPBoost	17	**20**	**10**	**15**	**16**	*20*	**12**	**15**
CoBoost	*15*	33	*11*	22	19	**14**	*13*	*17*
SemiBoost	22	59	43	46	53	85	41	52
MILBoost	**11**	*23*	20	**15**	*17*	21	27	32

Table 4.10: Tracking results on the benchmark sequences measured as average center location errors (in pixels) over 5 runs per sequence. Best performing method is marked in **bold face**, while the second best is shown in *italic*.

can accumulate over time and cause drifting. Therefore, recent approaches applied online extensions of boosting that can handle the uncertainty in the update process, such as CoBoost [62], SemiBoost [41] or MILBoost [5]. The main idea of these approaches is to define a region around the current tracking position and leave it up to the learner which samples to incorporate as positives or negatives, in order to stabilize the tracking. In the following, we compare our method to the state-of-the-art.

We use eight publicly available sequences including variations in illumination, pose, scale, rotation and appearance, and partial occlusions. The sequences *Sylvester* and *David* are taken from [78] and *Face Occlusion 1* is taken from [1], respectively. *Face occlusion 2*, *Girl*, *Tiger1,Tiger2* and *Coke* are taken from [5]. All video frames are gray scale and of size 320×240. We report the tracking accuracy in terms of average center location error in pixel to the ground-truth.

Since our method is a multi-view approach, it is straight-forward to use different feature information. However, this would make the comparison to other methods that are based on single features unfair. So in the following we report tracking results only for Haar-features and it should be clear to the reader (also by looking at previous experiments) that further improvement can be achieved by adding additional feature queues. In particular, we use 30 selectors with each 30 weak learners. The different views are generated by random sub-sampling from a large amount of Haar-features. In Table 4.10 we depict the results for all tracking sequences, *i.e.*, CoBoost [62], SemiBoost [41] and MILBoost [5]. As can be seen, MV-GPBoost performs best on five tracking sequences. The resulting tracking videos can be found in the supplementary material.

4.8 Discussion

In this chapter, we introduced various semi-supervised learning algorithms. GPMBoost is our generic algorithm, which uses prior-based cluster and manifold assumptions, therefore, making it ideal to work in most semi-supervised application domains. Additionally, we showed how the same algorithm can be easily used for learning from multiple views. Compared to many other state-of-the-art methods, we showed that our algorithm is very competitive on a wide range of applications.

In all the experiments we conducted, small RFs were used as weak learners. Recall that the complexity of RFs is $\mathcal{O}(n \log_2 n)$. The prior regularization over the unlabeled samples does not have any influence over the complexity of the GPMBoost, as it is linear in the number of unlabeled samples. If we use the manifold part, we need to compute the pairwise similarities and use them for updating the weights and pseudo-labels. This extends the complexity to $\mathcal{O}(n^2)$. During the training, it is also usual to use a very sparse similarity measure, for example by only considering the $k \ll n$ closest neighbors. If such an approach is used, then the complexity for the training phase stays constant as $\mathcal{O}(n^2)$. Therefore, we can see that the whole algorithm scales approximately quadratically with respect to the number of data samples.

Chapter 5

Online Multi-Class LPBoost

In this chapter we focus on developing an online multi-class version of LPBoost algorithm [27]. Our approach is based on the online convex programming techniques [114] and uses a primal-dual approach. We first introduce how the original binary LPBoost algorithm works. In Section 5.2 we present our multi-class extension of the LPBoost algorithm and show how it can be solved in online fashion. In Section 5.3 we conduct an extensive set of experiments to evaluate our proposed algorithm.

5.1 LPBoost Formulation

Since the original LPBoost algorithm works for binary problems, in this section we assume that the classification problem is binary as well, *i.e.* $y \in \{-1, +1\}$. For such problems, it is sufficient to train a classifier with signed confidences, and because of the symmetry condition in Eq. (3.6) we have $f(\mathbf{x}) = f_+(\mathbf{x}) = -f_-(\mathbf{x})$.

Recalling from the previous chapter, we can see the boosting as a linear classifier working over the feature space built by the weak learners. Let

$$\phi(\mathbf{x}) = [g(\mathbf{x}; \boldsymbol{\theta}_1), \dots, g(\mathbf{x}; \boldsymbol{\theta}_{|\mathcal{G}|})]^T \in \mathbb{R}^{|\mathcal{G}|}, \tag{5.1}$$

to be the responses from all possible weak learners $g \in \mathcal{G}$ for a training sample $\mathbf{x}$. Note that it is possible that the space of weak learners is rich and $|\mathcal{G}| = \infty$. Therefore, in practice it might not be possible to directly compute this vector. For some special weak learners, like stumps, the size of the number of possible weak learners is limited (*i.e.* $|\mathcal{G}| = d \times (n-1)$ entries to be correct). However, for many other weak learners it can be infinite dimensional.

In analogy with SVMs, we can see the ϕ as a very high dimensional feature mapping. Therefore, it is intuitive to apply the same training principles used for SVMs to this feature mapping as well. A linear classifier can be constructed as

$$f(\mathbf{x}; \mathbf{w}) = \phi(\mathbf{x})^T \mathbf{w}, \tag{5.2}$$

where $\mathbf{w} \in \mathbb{R}^{|\mathcal{G}|}$ is the weight vector. Now given a set of training samples, we can use a modified L_1 regularized SVM primal form to solve for $\mathbf{w}$ as

$$\begin{aligned} &\min_{\mathbf{w},\boldsymbol{\xi}} C \sum_{(\mathbf{x},y)\in\mathcal{X}_l} \xi_{\mathbf{x}} + \|\mathbf{w}\|_1 \\ &\text{s.t. } \forall m : w_m \geq 0 \\ &\forall \mathbf{x} : \xi_{\mathbf{x}} \geq 0 \\ &\forall (\mathbf{x}, y) : y\phi(\mathbf{x})^T \mathbf{w} + \xi_{\mathbf{x}} \geq 1. \end{aligned} \tag{5.3}$$

Comparing to the traditional SVM, we can see that there are slight modifications. First of all we are using an L_1 regularization. This is due to the fact that the space of features could be infinite dimensional. Therefore, by using an L_1 regularization, we hope to obtain a sparse solution for $\mathbf{w}$. In terms of boosting, this corresponds to selecting a finite set of weak learners. Another modification is the constraint over the positivity of the weights of weak learners w_m. Again according to boosting models, the weight of a weak learner represents its importance in the final voting, and hence, it only makes sense to have non-negative weights.

Clearly this optimization problem is a linear program, and hence the name of the algorithm is chosen to be LPBoost as well. As explained earlier, it is usually impossible to compute the feature vector. Therefore, Demiriz *et al.* [27] proposed to use the *column generation* technique to solve this problem.

Let Φ be the matrix where each row corresponds to the feature mapping of a sample $\phi(\mathbf{x})^T$. Therefore, each column is the response of a specific weak learner to all the data samples in the training set. The column generation method is usually used in large scale linear and integer programming problems, where the space size of the data terms in relatively high. This method is already available in many commercial optimization packages, such as CPLEX. The column generation method does not require all the columns in the data matrix to be available, and at each iteration only a subset of the columns are used. The only requirement is that there should either exist a method for determining the optimality of an existing solution or a way to generate columns of the data matrix which violates the constraints.

Translating this technique to boosting terms, the optimizer does not need to have access to all the weak learners (columns), therefore, there is no need to explicitly compute the matrix Φ. At each iteration, the LPBoost algorithm uses the Φ matrix obtained so far to solve for $\mathbf{w}$ and then checks the optimality of the constraints. If the optimal (or approximate enough optimal) solution is obtained, the algorithm terminates. Otherwise, LPBoost turns into its dual program and solves for the dual variables (which correspond to the sample weights in boosting terms). The dual of Eq. (5.3) can be written as

$$\begin{aligned} &\max_{\mathbf{d}} \|\mathbf{d}\|_1 \\ &\text{s.t. } \forall(\mathbf{x}, y) : 0 \leq d_{\mathbf{x}} \leq C \\ &\forall m : \sum_{(\mathbf{x},y)\in\mathcal{X}_l} y d_{\mathbf{x}} \phi_m(\mathbf{x}) \leq 1, \end{aligned} \tag{5.4}$$

where $\mathbf{d}$ is the dual variable of $\mathbf{w}$. Then LPBoost asks the weak learner to produce a column which maximally violates the constraint shown in Eq. (5.4). This corresponds to the column generation step. If the weak learner is able to produce such a result, the LPBoost algorithm adds it to the set of weak learners obtained so far and continues for the next iteration. Otherwise, the algorithm stops as the weak learner is unable to provide a solution.

5.2 Multi-Class LPBoost

After a brief introduction to LPBoost, in this section, we formulate the online multi-class boosting as a linear programming optimization problem. We first state the problem and then present our learning method which is based on a primal-dual gradient descent-ascent strategy.

In online learning scenarios, the data samples are presented sequentially. Following the repeated game analogy of online learning, the goal of the learner is to achieve a low cumulative loss over time by updating its model incrementally. Let the loss at iteration t be ℓ_t, which measures how bad the prediction of the learner $\hat{y}_t$ was with respect to the true class label of the newest sample y_t. In our formulation, we assume that the number of classes is not known in advance, and the learner should be able to incorporate new classes on-the-fly. Since the classes are presented over time, we do not penalize the learner when a new class is introduced.

Let $\mathcal{C}_t \subseteq \mathcal{C}$ be the set of known classes up to time t and $\mathcal{C}$ be the total label set (unknown to the learner). Also let $K_t = |\mathcal{C}_t|$ be the number of known classes at time t. In our formulation, the learner maintains a model $f_t : \mathbb{R}^d \rightarrow \mathbb{R}^{K_t}$ which is a mapping from the input feature space to the multi-class hypothesis domain. We represent the confidence of the learner for the k-th class $f_{t,k}(\mathbf{x}_t)$ as the k-th element of the output vector $f_t(\mathbf{x}_t) = [f_{t,1}(\mathbf{x}_t), \dots, f_{t,K_t}(\mathbf{x}_t)]^T$. As represented in the previous chapter, we define the *hard margin* of a sample $\mathbf{x}_t$ as

$$m_{y_t}(\mathbf{x}_t) = f_{t,y_t}(\mathbf{x}_t) - \max_{\substack{k \in \mathcal{C}_t \\ k \neq y_t}} f_{t,k}(\mathbf{x}_t), \tag{5.5}$$

which measures the difference between the classification confidence of the true class and the closest non-target class

$$y'_t = \arg\max_{\substack{k \in \mathcal{C}_t \\ k \neq y_t}} f_{t,k}(\mathbf{x}_t).$$

Note that based on the decision rule of Eq (2.3), $m_{y_t}(\mathbf{x}_t) < 0$ means a wrong prediction, while a positive margin means a correct classification.

In this work, we use the *hinge* loss function

$$\ell_t(m_{y_t}) = \mathbb{I}(y_t \in \mathcal{C}_t) \max(0, 1 - m_{y_t}(\mathbf{x}_t)), \tag{5.6}$$

where $\mathbb{I}(\cdot)$ is an indicator function, which is introduced so that we do not penalize the model if there is a novel class introduced. Note that hinge loss is an upper bound on the mis-classification error

$$\sum_{t=1}^{T} \mathbb{I}(y_t \neq \hat{y}_t) \leq \sum_{t=1}^{T} \ell_t(m_{y_t}), \tag{5.7}$$

and hence, its minimization results in minimizing the mis-classification error rate.

In offline case, LPBoost sequentially adds base learners to the whole model. However, in the online learning formulation this scenario is not possible as not only the columns of the data matrix have to be added, but also its rows (which correspond to the data samples) are added over time. Therefore, following the previous online boosting models, our model utilizes a fixed set of online base learners and updates them sequentially by adjusting the weight of a sample.

Our learner is an adaptive boosting model, *i.e.* at time t can be written as

$$f_t(\mathbf{x}_t) = \sum_{m=1}^{M} w_{t,m} g_{t,m}(\mathbf{x}_t), \tag{5.8}$$

where $g_{t,m} : \mathbb{R}^d \to \mathbb{R}^{K_t}$ is the m-th weak learner, M represents the number of weak learners, and $w_{t,m}$ is the weight of m-th base. Following the compact formulation of Eq. (3.9), we represent the classifier as

$$f_t(\mathbf{x}_t) = G_t(\mathbf{x}_t)\mathbf{w}_t. \tag{5.9}$$

We denote $G_t(y, \cdot)$ to be the y-th row of this matrix, and $G_t(y, m)$ to be the element in the y-th row and the m-th column.

Let $\mathcal{B}_{\Delta T}$ be a cache with size ΔT. A cache of size $\Delta T = 1$ will correspond to the case that the learner discards the sample after updating with it. Considering our boosting model and the loss function presented in Eq (5.6), we propose the following regularized multi-class LPBoost problem to be optimized online

$$\begin{aligned}
&\min_{\mathbf{w}_T, \boldsymbol{\xi}} C \sum_{t \in \mathcal{B}_{\Delta T}} \sum_{k \neq y_t} \xi_{t,k} + \|\mathbf{w}_T\|_1 \\
&\text{s.t. } \forall m : w_{T,m} \geq 0 \\
&\forall t, \forall k \neq y_t : \xi_{t,k} \geq 0 \\
&\forall t, \forall k \neq y_t : \big(G_t(y_t, \cdot) - G_t(k, \cdot)\big)\mathbf{w}_T + \xi_{t,k} \geq 1
\end{aligned} \tag{5.10}$$

where C is the capacity parameter, and slack variables $\xi_{t,k}$ are added to create *soft margins* for boosting. Note that this formulation is a direct generalization of the original formulation of LPBoost [27] to the multi-class case. Furthermore, if we would share the slack between all the classes, then it would be closely related to the multi-class variant of ν-LPBoost proposed in [36]. In an offline scenario, such problems can be easily solved by standard optimization techniques. However, in the online setting, it is usually infeasible to solve this problem from scratch for every new sample added to the system. Therefore, an incremental solution is desired. Fortunately, due to the convexity of the problem, one can benefit from the previously proposed *online convex programming* approaches [114].

5.2.1 Online Learning

Our online learning method performs primal-dual gradient descent-ascent iteratively. In detail, we first convert the problem into its *augmented Lagrangian* form [71]. By each new sample, we first perform a dual ascent, which is equivalent to finding sample weights for each iteration of training weak learners. After finishing that step, we do a primal descent over the weights of weak learners.

The Lagrange dual function of the optimization problem Eq (5.10) is

$$\begin{aligned}
D(\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{d}) = & \sum_{t \in \mathcal{B}_{\Delta T}} \sum_{k \neq y_t} d_{t,k} + \\
& + \inf_{\mathbf{w}_T, \boldsymbol{\xi}} \Big(\sum_{m=1}^{M} (1 - \alpha_m) w_{T,m} + \\
& + \sum_{t \in \mathcal{B}_{\Delta T}} \sum_{k \neq y_t} (C - d_{t,k} - \beta_{t,k}) \xi_{t,k} - \\
& - \sum_{t \in \mathcal{B}_{\Delta T}} \sum_{k \neq y_t} d_{t,k} \big(G_t(y_t, \cdot) - G_t(k, \cdot) \big) \mathbf{w}_T \Big),
\end{aligned} \tag{5.11}$$

where $\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{d}$ are the Lagrange multipliers of the constraints. Due to the linearity of the inner problem of Eq (5.11), for a set of finite solutions the following conditions must hold

$$\begin{aligned}
& \forall t, \forall k \neq y_t : C - d_{t,k} - \beta_{t,k} = 0 \\
& \forall m : 1 - \alpha_m - \sum_{t \in \mathcal{B}_{\Delta T}} \sum_{k \neq y_t} d_{t,k} \Delta G_{t,k}(m) = 0,
\end{aligned}$$

where $\Delta G_{t,k}(m) = G_t(y_t, m) - G_t(k, m)$. Using the positivity conditions on Lagrange multipliers, we can derive the dual formulation of Eq (5.10) as

$$\begin{aligned}
& \max_{\mathbf{d}} \sum_{t \in \mathcal{B}_{\Delta T}} \sum_{k \neq y_t} d_{t,k} \\
& \text{s.t. } \forall m : \sum_{t \in \mathcal{B}_{\Delta T}} \sum_{k \neq y_t} d_{t,k} \Delta G_{t,k}(m) \leq 1 \\
& \forall t, \forall k \neq y_t : 0 \leq d_{t,k} \leq C.
\end{aligned} \tag{5.12}$$

The vector $\mathbf{d}$, which corresponds to sample weights, is the dual variable of weights on weak learners. The first set of constraints are equivalent to the *edge* constraints of binary LPBoost. As it can be seen, there are $K - 1$ weights per each sample. However, the weak learners usually accept only one weight per instance. Therefore, we only consider the most violating edge constraint for each example. This corresponds to finding the non-target class for which the margin is the smallest: y'_t [1]. Therefore, we will concentrate

[1]Note that this is a limitation imposed by weak learners. The following derivations can be easily generalized for all the weights.

on the following problem

$$
\begin{aligned}
&\max_{\mathbf{d}} \sum_{t \in \mathcal{B}_{\Delta T}} d_{t,y'_t} \\
&\text{s.t. } \forall m : \sum_{t \in \mathcal{B}_{\Delta T}} d_{t,y'_t} \Delta G_{t,y'_t}(m) \leq 1 \\
&\forall t : 0 \leq d_{t,y'_t} \leq C.
\end{aligned}
\tag{5.13}
$$

Optimizing the problem in Eq. (5.13) with an online convex programming technique [114], requires a projection step for finding solutions which are consistent with the constraints. However, in our case, such a step is expensive to compute; therefore, we formulate its augmented Lagrangian [71] as

$$
\begin{aligned}
&\max_{\mathbf{w}_T, \mathbf{d}} \sum_{t \in \mathcal{B}_{\Delta T}} d_{t,y'_t} + \\
&+ \sum_{m=1}^{M} w_{T,m} (1 - \sum_{t \in \mathcal{B}_{\Delta T}} d_{t,y'_t} \Delta G_{t,y'_t}(m) - \zeta_m) - \\
&- \frac{1}{2\theta} \sum_{m=1}^{M} (1 - \sum_{t \in \mathcal{B}_{\Delta T}} d_{t,y'_t} \Delta G_{t,y'_t}(m) - \zeta_m)^2 \\
&\text{s.t. } \forall m : \zeta_m \geq 0, w_{T,m} \geq 0 \\
&\forall t : 0 \leq d_{t,y'_t} \leq C,
\end{aligned}
\tag{5.14}
$$

by introducing a new set of slack variables ζ_m and using $\theta > 0$ as a constant. Note that the value of slacks can be easily found by computing the derivatives of the objective with respect to them and setting it to zero. This leads to

$$\zeta_m = \max(0, q_m),$$

where

$$q_m = 1 - \sum_{t \in \mathcal{B}_{\Delta T}} d_{t,y'_t} \Delta G_{t,y'_t}(m) - \theta w_{T,m}.$$

Now we follow this procedure over time: when a new sample arrives, we set its weight to C and update the cache by removing the oldest sample and inserting the newest. Then, for training the m-th weak learner, we compute the sample weights by dual gradient ascent update

$$\begin{aligned} \forall t: \ e_t =& d_{t,y'_t} + \nu_d \Big(1 + \frac{1}{\theta} \sum_{\substack{j=1 \\ q_j<0}}^{m-1} q_j \Delta G_{t,y'_t}(j)\Big) \\ d_{t,y'_t} \leftarrow& \max(0, \min(C, e_t)), \end{aligned} \tag{5.15}$$

where ν_d is the dual learning rate. After updating the sample weight and training the m-th weak learner according to them, we compute an update for the weight of this weak learner by a primal gradient descent update

$$\begin{aligned} \forall m: \ z_m =& w_{T,m} - \nu_p \Big(1 - \sum_{t \in \mathcal{B}_{\Delta T}} d_{t,y'_t} \Delta G_{t,y'_t}(m)\Big) \\ w_{T,m} \leftarrow& \max(0, z_m), \end{aligned} \tag{5.16}$$

where ν_p is the learning rate for the primal. This alternating primal-dual descent-ascent is continued for all the weak learners.

Discussion Although the update rules presented in Eq (5.15) and Eq (5.16) look complicated, in fact, they present intuitive learning strategies which are closely related to the boosting way of learning from data. In Eq (5.15), the inner sum shows the total confidence of the weak learners trained so far, with respect to the classification margin of the current sample. Note that since $q_j < 0$, for a sample, which many of the weak learners obtain a positive margin, this sum will be a large negative value. Hence, for such a sample with a large positive margin, the weight will decrease for the training of the next weak learner. Similarly, for the update in Eq (5.16), if a weak learner has a high weighted average margin over all the samples in the cache, the inner sum will be high, which will lead to an increase in its weight. Therefore, the weight of successful weak learners will increase.

5.3 Experiments

We evaluate the proposed Online Multi-Class LPBoost (OMCLP) algorithm by comparing its performance to other online learning algorithms. In the first two sets of experiments, we mainly compare with other multi-class online and offline algorithms, while in last section we will conduct tracking experiments.

5.3.1 Machine Learning Benchmarks

Since there is no other online multi-class boosting algorithm available in literature for comparison, we convert the recently proposed offline multi-class boosting algorithm of Zou *et al.* [115] to online formulation. Based on their formulation, we define a margin vector based on the current classifier as

$$\forall \mathbf{x}_t : \sum_{i=1}^{K} f_{t,i}(\mathbf{x}_t) = 0. \tag{5.17}$$

We then use a Fisher consistent convex loss function [115], which guarantees that by training over a large number of samples, the boosting model is able to recover the unknown Bayes decision rule. For this work, we experiment with two different loss functions: the exponential loss $e^{-f_{t,y_t}(\mathbf{x}_t)}$ and the Logit loss $\log(1 + e^{-f_{t,y_t}(\mathbf{x}_t)})$. For updating the m-th weak learner, we perform a functional gradient descent as

$$g_{t,m}(\mathbf{x}) = \arg\max_{g} \nabla \ell(f_{t,y_t}^{m-1}(\mathbf{x}_t)) g_{y_t}(\mathbf{x}_t), \tag{5.18}$$

where $\nabla \ell(f_{t,y_t}^{m-1}(\mathbf{x}_t))$ is the gradient of the loss function at the m-th stage of boosting. As it will be shown later, in principle, this is also a novel and successful online multi-class boosting algorithm. We call this algorithm Online Multi-Class Gradient Boost (OMCGB).

Additionally, we compare with Online Random Forests (ORF) [88] and the highly successful online multi-class support vector machine algorithm of Bordes *et al.* [12] named Linear LaRank. Note that both of these algorithms are inherently multi-class, so they provide a fair comparison. We also performed experiments with the online AdaBoost formulation of Oza *et al.* [72] by training 1-vs-all classifiers. However, its performance was not comparable to these baseline methods; therefore, due to lack of space we withhold reporting them. We also compare our method with the following offline trained multi-class classifiers: Random Forests [15], three multi-class formulations of AdaBoost namely SAMME [111], AdaBoost.ECC [43], and the recent algorithm of AdaBoost.SIP [110][2]. We also compare with the multi-class support vector machine algorithms of Keerthi *et al.* [53] with a linear kernel and the multi-class SVM from LibSVM with RBF kernel.

The OMCLP and OMCGB use small ORFs with 10 trees as their weak learners and we set $M = 10$. ORF, when used as a single model, uses 100 trees trained online. For our algorithm, we fix the cache size to 1 and set $\nu_d = \theta = 2$, $\nu_p = 1e^{-6}$, and $C = 5$.

[2]For these algorithms we report the results presented in [110].

Methods - Dataset	DNA		Letter		Pendigit		USPS	
# Epochs	1	10	1	10	1	10	1	10
OMCLP	0.0983	**0.0565**	**0.1202**	**0.0362**	**0.0747**	**0.0241**	0.1185	**0.0809**
OMCGB-Log	0.2648	0.0777	0.3033	0.1202	0.1666	0.0599	0.2418	0.1241
OMCGB-Exp	0.1395	0.0616	0.2484	0.0853	0.1282	0.0501	0.1926	0.1103
ORF	0.2243	0.0786	0.2696	0.0871	0.1343	0.0464	0.2066	0.1085
LaRank	**0.0944**	0.0818	0.5656	0.5128	0.1712	0.2109	**0.0964**	0.1004
RF	0.0683		0.0468		0.0387		0.0610	
MCSVM-Lin	0.0727		0.2575		0.1266		0.0863	
MCSVM-RBF	0.0559		*0.0298*		*0.0360*		*0.0424*	
SAMME [110]	0.1071		0.4938		0.3391		N/A	
AdaBoost.ECC [110]	*0.0506*		0.2367		0.1029		N/A	
AdaBoost.SIP [110]	0.0548		0.1945		0.0602		N/A	

Table 5.1: Classification error on machine learning benchmark datasets for 1 and 10 epochs. The **bold-face** shows the best performing *online* method, while the *italic* shows the best *offline* method.

Note that these set of parameters will be used for all the datasets in this section and the next section. For offline methods, we use a 5-fold cross-validation to obtain their hyperparameters.

Table 5.1 shows the classification error on 4 benchmark datasets chosen from the UCI repository. All the experiments are run for 5 independent runs and the results are the average classification error on the held out test set. In order to simulate large scale datasets, we conduct the experiments in different number of epochs: each epoch corresponds to seeing all the data points once in random order. As it can be seen, our algorithm outperforms other online learning methods in 6 out of 8 cases and comes very close to the performance of the best offline methods. Another interesting observation is the fact that the performance of the OMCLP at the first epoch is very close to the performance of the ORF after 10 epochs. This shows that given a slow converging algorithm like ORF, we are able to speed up its convergence rate as well. Our C++ implementation of OMCLP and OMCGB is freely available from the following link[3].

[3]http://www.ymer.org/amir/software/online-multiclass-lpboost/

# Train	15		30	
# Epochs	1	10	1	10
OMCLP	**0.7437**	**0.6093**	**0.6672**	**0.5406**
OMCGB	0.7520	0.6226	0.6860	0.5693
ORF	0.8969	0.8265	0.8880	0.8142
LaRank	0.7856	0.6353	0.7205	0.5803

Table 5.2: Classification error on Caltech101 datasets for 1 and 10 epochs. The **boldface** shows the best performing *online* method.

5.3.2 Object Category Recognition

Online multi-class learning is essential when dealing with large-scale image databases. For example, typical image or video search engines often need to update their internal model when a set of new data is available. However, rebuilding the entire model is infeasible in practice. Considering the fact that the problem of object category recognition is inherently multi-class, therefore such systems can benefit from an online multi-class learner.

We evaluate on *Caltech101* object category recognition task, which is a challenging task for an online learner, since the number of classes is large and the number of training samples per class is small. For these experiments, we use the Linear LaRank as the weak learners of the online boosting methods, due to the fact that ORFs were performing poorly on this task. We convert the SVM scores of LaRank to probabilities via a multinomial logistic regression. All other settings are the same as the experiments presented in Section 5.3.1.

We present the results using the standard Caltech101 settings of training on 15 and 30, and testing on 50 images per class. For feature extraction, we use the precomputed 360 degree Level2-PHOG features from Gehler and Nowozin [36]. Table 5.2 shows the results obtained by various online methods, averaged over 5 independent runs on 5 different train and test splits (total 25 runs per algorithm)[4]. As it can be seen, our algorithm performs the best compared to other methods on this difficult task.

Figure 5.1(a) shows how the performance varies by the number of training images per category. As expected, more training samples help algorithms to improve over time. However, it is notable that our method consistently obtains lower errors compared to other algorithms over different amounts of labeled data. Figure 5.1(b) shows the

[4]We only report the performance of OMCGB-Exp as with Logit loss the results were similar.

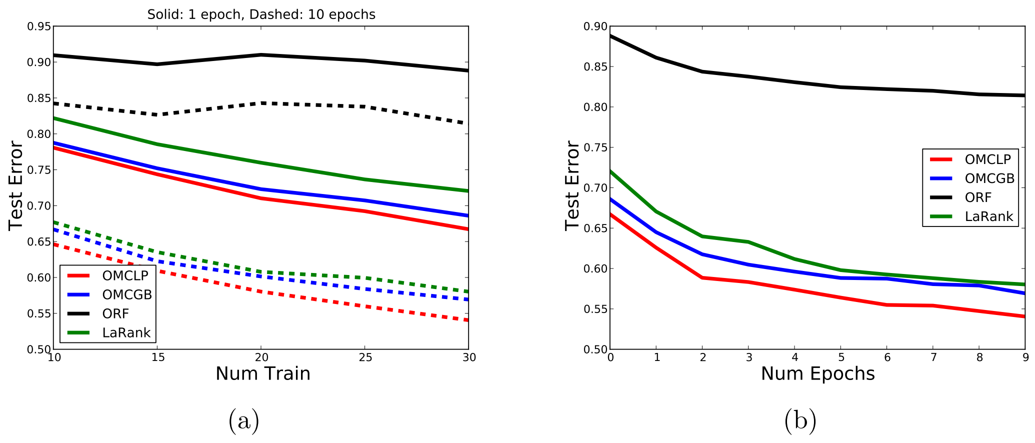


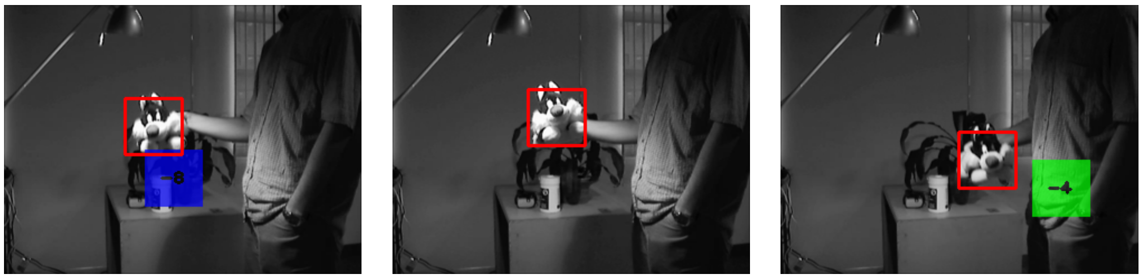
Figure 5.1: (a) Classification error on Caltech101 for 1 epoch (solid) and 10 epochs runs when the number of training images per class is varying. (b) Classification error over different number of epochs when using 30 images per category for training.

dynamics of the learners when the same training data is reshuffled and presented as new samples. We can see that although all methods benefit from revisiting the samples, our algorithms makes the most out of the epochs, and as can be seen towards the end of the 10-th epoch, it has the highest gap compared to the second best algorithm, OMCGB.

5.3.3 Tracking with Virtual Classes

Object Tracking is a common application for online learning algorithms in computer vision. Within this section, we will show the performance of our algorithm in a *tracking by detection* scenario. When training a discriminative object detector, the problem is usually formulated as binary classification. In tracking, we usually have fast changing, cluttered, complex background, which has to be described with a single background model. However, our approach is to break this binary task into a multi-class problem and utilize our robust online learner to discriminate between these classes.

From the classification margin point of view, the background might have samples which are very close to the decision boundaries of the target object. These samples are usually potential false positive detections during the tracking, specially when there are fast appearance changes or occlusions. Since, we know that our classifier can maximize



(a) Addition of a virtual class (b) No negative update (c) Updating a virtual class

Figure 5.2: Tracking with virtual background classes.

the soft-margin of data instances, we sample densely from the decision boundaries of the target class in the feature space for potential false positive background regions. Then, each of these background regions is assigned to a separate *virtual* class. Hence, our online multi-class classifier will maximize its margin with respect to all these classes, while also in image domain, it will keep tracking them as they were indeed target objects. Since our learner is able to accommodate new classes on-the-fly, we can keep track of any new object entering the scene which might cause possible confusions. Figure 5.2 shows this procedure in action: Figure 5.2(a) depicts the addition of a virtual background class, and Figure 5.2(c) indicates the update of an existing virtual class[5].

We conduct an extensive evaluation of our tracking method. Since we want to show that the performance gain comes from our robust multi-class classifier and from the addition of novel virtual background classes, we only use simple Haar-like Features without any post-processing. For the evaluation of our tracker, we use the detection-criterion of the VOC Challenge [30], which is defined as $|R_T \cap R_{GT}|/|R_T \cup R_{GT}|$, where R_T is the tracking rectangle and R_{GT} the ground-truth. The advantage of this score is that it truly shows how accurate the detection of the model is, rather than computing the raw distance measure between the target and background. We measure the accuracy of a tracker by computing the average detection score for the entire video. We run each tracker 5 times and report the median average score. Table 5.3 lists the results for several publicly available benchmark sequences in comparison to other state-of-the-art tracking methods: MILTracker [5], FragTracker [1], and AdaBoostTracker [39]. In 5 out of 8 videos we outperform other methods, while for the remaining 3 videos we are the second best method. Our unoptimized C++ implementation of OMCLP algorithm

[5]Please refer to supplementary material for videos describing our tracking method in details and its results.

Sequence	OMCLP	MIL [5]	Frag [1]	OAB [39]
Sylvester	**0.67**	0.60	*0.62*	0.520
Face 1	*0.80*	0.60	**0.88**	0.48
Face 2	**0.78**	*0.68*	0.44	*0.68*
Girl	**0.64**	0.53	*0.60*	0.40
Tiger 1	**0.53**	*0.52*	0.19	0.23
Tiger 2	*0.44*	**0.53**	0.15	0.28
David	**0.61**	*0.57*	0.43	0.26
Coke	*0.24*	**0.33**	0.08	0.17

Table 5.3: Average detection score: **bold-face** shows the best method, while *italic* indicates the second best.

reaches near real-time performance (around 10 to 15 frames/second on average).

5.4 Discussion

In this chapter, we presented the Online Multi-Class LPBoost algorithm, which is able to build in an online setting, a robust multi-class boosting model with maximizing the soft-margin of the data samples. We solved the optimization problem by performing a variant of the online convex programming technique, based on an efficient primal-dual gradient descent-ascent strategy. Based on an extensive set of experiments, we showed that our method outperforms the state-of-the-art on a wide range of applications, such as pattern recognition tasks, object category recognition tasks, and object tracking. The complexity of our optimization steps are linear in number of samples the algorithm works, and therefore, its complexity will be dependent mainly on the complexity of the weak learners.

Chapter 6

Conclusions

6.1 Summary

In this thesis, we developed various multi-class boosting algorithms for supervised, semi-supervised, and online learning scenarios. Boosting is one of the widely used learning algorithms in machine learning and computer vision, and has been shown to produce state-of-the-art results in many applications. Many of these applications are inherently multi-class problems. Although there are ways to use a binary algorithms for multi-class tasks, these methods have their own disadvantages. Therefore, we focused specifically on multi-class problems.

In Chapter 3, we developed a generic multi-class boosting algorithm which can use many different loss functions. This makes this algorithm ideal for many applications and provides the flexibility of choosing a proper behavior, depending on the nature of the task. Chapter 4 saw the extension of this algorithm to multi-class semi-supervised tasks. We provided a generic method which can be used for the semi-supervised tasks where the cluster, manifold, or both assumptions hold for the data. Our method is based on the concept of learning from priors and is able to use prior knowledge in terms of uncertainties encoded in probabilities. We investigated the effect of different loss functions suitable for these kinds of problems and provided a special robust loss function applicable to many applications, where the prior could contain a considerable amount of noise. We also showed how the same principle can be used for learning from multiple views. We applied these algorithms to a wide range of computer vision problems, such as object category recognition or visual object tracking. Our experimental results confirm the effectiveness of our algorithms and its applicability to many computer vision tasks.

We focused our attention to the online learning problem in Chapter 5, where we developed an online multi-class variant of LPBoost. Our methodology was to convert the LPBoost formulation into a primal-dual form where we applied a variant of the online convex programming technique to perform gradient descent-ascent over the primal-dual formulation. We conducted an extensive set of experiments by applying our algorithm to different pattern recognition and computer vision problems.

We released the source code implementation of all of the algorithms discussed in this thesis. This makes it easy for the machine learning and computer vision community to experiment with the algorithms and develop them further.

6.2 Extensions

The algorithms we proposed in this dissertation can be improved in several ways. The amount of digital data that we produce everyday in form of documents, images, and videos is growing exponentially. Most of these data will remain unlabeled and it is infeasible to annotate even a small portion of them. Therefore, having semi-supervised algorithms which can handle Internet-scale data is a very interesting open research direction.

It has been shown that for many problems where the number of data samples is huge, online learning methods are the only ones that can work properly. An offline learning algorithm needs to analyze all the data samples before making any optimization or adjustment of its parameters. For example, when making a gradient descent step, the optimizer should compute the gradients with respect to all the data samples. This is feasible if the data is small and in fact can fit into memory. Such offline learning strategies can not simply handle the Internet-scale problems. However, online learning algorithms, by nature, work with individual data samples and do not need to have access to the entire problem domain. This makes them an attractive, or simply the only option, for learning from very large-scale problems.

Following the discussion regarding the size of the data and the fact that most of them will be unlabeled, leads to an intuitive step to develop online semi-supervised learning algorithms. We already have started to look into such algorithms and provided extensions of some of the semi-supervised algorithms we discussed in this thesis in [109, 56]. However, their main applications were the visual object tracking task. Therefore, it is an interesting research direction to extend and examine these algorithms in the context of very large-scale problems.

Internet-based learning requires algorithms which can not only scale well with respect to the amount of the data and can handle unlabeled samples, but are also able to handle a considerable amount of noise. For example, many photo and video sharing websites provide the possibility to tag an entree or write descriptions and comments about it.

These kind of information can be used to somehow guess the true label of an image for example. However, such high level information might be quite noisy. For example, people have different opinions about what has to be a tag for an image, and it could be that all of the proposed tags make sense or just a subset of them. Obviously the comments regarding an entire, is a place of debate and therefore, it is expected that they contain many conflicting facts. As a result, any high level information mined from an online website could have a considerable amount of noise in it. Hence, another research direction related to the work we discussed in this thesis is to develop robust algorithms which are able to cope with a considerable amount of noise.

Another interesting research direction related to the topics covered in this thesis, is the possibility of learning from various information domains. Domain adaptation, transfer learning, and learning from attributes are special topics in machine learning and computer vision, with the goal of using the information available in one domain to improve the performance of a classifier which has access to only a limited amount of data for the task which it specializes on. Combining ideas from online and semi-supervised learning algorithms with these methods can lead to very useful algorithms, applicable to many problems in computer vision and other fields.

Appendices

A: Gradients for Supervised Loss Functions

In this section, we present various multi-class boosting algorithms based on the choice of different loss functions presented in Section 3.3. First, we need to compute the gradients of the loss function ℓ, with respect to the current model f_{t-1}. Then we need to design the algorithms in a way that it is possible for large classes of base functions to be able to operate inside the boosting mechanism.

Gradients for Margin-Based Loss Functions

Using the chain rule, we can write the k-th element of the gradient vector $\nabla\ell(\mathbf{x}, y; f_{t-1})$ as [1]

$$\frac{\partial\ell(\mathbf{x}, y; f)}{\partial f_k(\mathbf{x})} = \frac{\partial\ell(\mathbf{x}, y; f)}{\partial m(\mathbf{x}, y; f)} \frac{\partial m(\mathbf{x}, y; f)}{\partial f_k(\mathbf{x})}. \tag{.1}$$

Note that the last term is independent of the loss function, therefore, we develop it first as it will be shared between all margin-based loss functions.

The margin term includes a max operator, which is not differentiable on its own. Therefore, we propose to replace the max with the equivalent L_∞-norm as

$$\max_{j \neq y} f_j(\mathbf{x}) = \lim_{p \to \infty} (\sum_{j \neq y} f_j(\mathbf{x})^p)^{\frac{1}{p}}. \tag{.2}$$

Using the L_∞-norm, we can develop the gradients as

$$\begin{aligned}\frac{\partial \max\limits_{j \neq y} f_j(\mathbf{x})}{\partial f_k(\mathbf{x})} =& \mathbb{I}(k \neq y) \lim_{p \to \infty} \frac{1}{p} [(\sum_{j \neq y} f_j(\mathbf{x})^p)^{\frac{1}{p}-1} (p f_k(\mathbf{x})^{p-1})] = \\ =& \mathbb{I}(k \neq y) \frac{1}{f_k(\mathbf{x})} \lim_{p \to \infty} (\sum_{j \neq y} f_j(\mathbf{x})^p)^{\frac{1}{p}} \frac{f_k(\mathbf{x})^p}{\sum_{j \neq y} f_j(\mathbf{x})^p}. \end{aligned} \tag{.3}$$

[1] For notational brevity, we simply write $\frac{\partial\ell(\mathbf{x},y;f)}{\partial f_k(\mathbf{x})}$ instead of the more correct form $\frac{\partial\ell(\mathbf{x},y;f)}{\partial f_k(\mathbf{x})}|_{f(\mathbf{x})=f_{t-1}(\mathbf{x})}$.

The last term in this equation is the *soft-max* function, which in limit, is equivalent to the indicator function for selecting the maximum of f. The first term inside the limit is also the maximum of f. Therefore, this expression can be simplified as

$$\frac{\partial \max\limits_{j \neq y} f_j(\mathbf{x})}{\partial f_k(\mathbf{x})} = \mathbb{I}(k \neq y)\mathbb{I}(k = \arg\max_{j \neq y} f_j(\mathbf{x})). \tag{.4}$$

In other words, if j is among those classes which are the closest to the target class y, then the gradient of the margin is 1.

By using these results, we can now compute the gradients of the margin as

$$\frac{\partial m(\mathbf{x}, y; f)}{\partial f_k(\mathbf{x})} = \mathbb{I}(k = y) - \mathbb{I}(k \neq y)\mathbb{I}(k = \arg\max_{j \neq y} f_j(\mathbf{x})). \tag{.5}$$

Now that we have the gradients of the margin, we only need to find the derivatives of the loss function with respect to the margin term. The following list provides these derivatives:

- Hinge loss: [2]
$$\frac{\partial \ell_h(\mathbf{x}, y; f)}{\partial m(\mathbf{x}, y; f)} = -\mathbb{I}(m(\mathbf{x}, y; f_{t-1}) < 1), \tag{.6}$$

- Exponential loss:
$$\frac{\partial \ell_e(\mathbf{x}, y; f)}{\partial m(\mathbf{x}, y; f)} = -e^{-m(\mathbf{x}, y; f_{t-1})}, \tag{.7}$$

- Logit loss:
$$\frac{\partial \ell_l(\mathbf{x}, y; f)}{\partial m(\mathbf{x}, y; f)} = -\frac{e^{-m(\mathbf{x}, y; f_{t-1})}}{1 + e^{-m(\mathbf{x}, y; f_{t-1})}}, \tag{.8}$$

- Savage loss:
$$\frac{\partial \ell_s(\mathbf{x}, y; f)}{\partial m(\mathbf{x}, y; f)} = -\frac{4e^{2m(\mathbf{x}, y; f_{t-1})}}{(1 + e^{2m(\mathbf{x}, y; f_{t-1})})^3}. \tag{.9}$$

[2]Since the hinge loss also has a max operator, we use the results from the previous part to compute its derivatives.

Gradients for Log-Likelihood Loss Function

It is relatively easy to compute the gradients of the negative log-likelihood loss function. Using the chain rule, we have

$$\frac{\partial \ell_{ll}(\mathbf{x}, y; f)}{\partial f_k(\mathbf{x})} = \frac{\partial \ell_{ll}(\mathbf{x}, y; f)}{\partial p(y|\mathbf{x}; \boldsymbol{\beta})} = \frac{\partial p(y|\mathbf{x}; \boldsymbol{\beta})}{\partial f_k(\mathbf{x})}. \tag{.10}$$

Throughout this work, we will use the multi-nomial logistic regression model of Eq. (3.19). Therefore, the last term can be computed as

$$\begin{aligned}\frac{\partial p(y|\mathbf{x}; \boldsymbol{\beta})}{\partial f_k(\mathbf{x})} =& \frac{\mathbb{I}(k = y) e^{f_y(\mathbf{x})} \sum_{i=1}^{K} e^{f_i(\mathbf{x})} - e^{f_y(\mathbf{x})} e^{f_k(\mathbf{x})}}{(\sum_{i=1}^{K} e^{f_i(\mathbf{x})})^2} = \\ =& \frac{e^{f_y(\mathbf{x})}}{\sum_{i=1}^{K} e^{f_i(\mathbf{x})}} (\mathbb{I}(k = y) - \frac{e^{f_k(\mathbf{x})}}{\sum_{i=1}^{K} e^{f_i(\mathbf{x})}}) = \\ =& p(y|\mathbf{x}; \boldsymbol{\beta})(\mathbb{I}(k = y) - p(k|\mathbf{x}; \boldsymbol{\beta})).\end{aligned} \tag{.11}$$

This results in

$$\begin{aligned}\frac{\partial \ell_{ll}(\mathbf{x}, y; f)}{\partial f_k(\mathbf{x})} =& - \frac{1}{p(y|\mathbf{x}; \boldsymbol{\beta}_{t-1})} p(y|\mathbf{x}; \boldsymbol{\beta}_{t-1})(\mathbb{I}(k = y) - p(k|\mathbf{x}; \boldsymbol{\beta}_{t-1})) = \\ =& p(k|\mathbf{x}; \boldsymbol{\beta}_{t-1}) - \mathbb{I}(k = y).\end{aligned} \tag{.12}$$

B: Gradients for Unsupervised Loss Functions

In the following sections, we will use

$$q_{\mathbf{x}} = [q(1|\mathbf{x}), \dots, q(K|\mathbf{x})]^T,$$

and

$$p_{\mathbf{x}} = [p(1|\mathbf{x}; \boldsymbol{\beta}), \dots, p(K|\mathbf{x}; \boldsymbol{\beta})]^T,$$

as the vectorized representation of the prior and the posterior for a given sample $\mathbf{x}$.

We also show that the sum of the gradients for the prior term and the manifold term is zero for all these loss functions. This will be used later on to show that if one wants to use a multi-class classifier as the base learner of the boosting method shown in Algorithm 4, then the weights computed by each of these loss functions is in fact non-negative.

Priors Regularization

Kullback-Leibler Divergence

As noted before, for priors the $H(q)$ is fixed and therefore, can be eliminated from the optimization. Hence, by using the multi-nomial logistic regression model the loss function becomes

$$\begin{aligned} \jmath_{kl}^{p}(\mathbf{x}, q; f) =& H(q, p) = -\sum_{j \in \mathcal{Y}} q(j|\mathbf{x}) \log p(j|\mathbf{x}; \boldsymbol{\beta}) = \\ =& -\sum_{j \in \mathcal{Y}} q(j|\mathbf{x})(f_j(\mathbf{x}; \boldsymbol{\beta}) - \log \sum_{i \in \mathcal{Y}} e^{f_i(\mathbf{x}; \boldsymbol{\beta})}) = \\ =& -q_{\mathbf{x}}^T f(\mathbf{x}; \boldsymbol{\beta}) + \log \sum_{i \in \mathcal{Y}} e^{f_i(\mathbf{x}; \boldsymbol{\beta})}. \end{aligned} \tag{.13}$$

Using this simplification, the gradients can be computed as

$$\frac{\partial \jmath_{kl}^{p}(\mathbf{x}, q; f)}{\partial f_k(\mathbf{x})} = -q(k|\mathbf{x}) + \frac{e^{f_k(\mathbf{x}; \boldsymbol{\beta})}}{\sum_{i \in \mathcal{Y}} e^{f_i(\mathbf{x}; \boldsymbol{\beta})}} = -q(k|\mathbf{x}) + p(k|\mathbf{x}; \boldsymbol{\beta}). \tag{.14}$$

This clearly shows that the gradients are equal to the negative of the differences between the prior and the model for each class. The following theorem shows that the sum of the gradient terms over the classes is zero.

Theorem .0.1. *Let*

$$d_{\mathbf{x},k} = -\frac{\partial \jmath^{p}_{kl}(\mathbf{x}, q; f)}{\partial f_k(\mathbf{x})}. \tag{.15}$$

The sum of the weights over the classes is zero

$$\sum_{k\in\mathcal{Y}} d_{\mathbf{x},k} = 0. \tag{.16}$$

Proof. The proof follows the fact that

$$\sum_{k\in\mathcal{Y}} d_{\mathbf{x},k} = \sum_{k\in\mathcal{Y}} q(k|\mathbf{x}) - p(k|\mathbf{x};\boldsymbol{\beta}) = 0. \tag{.17}$$

□

Symmetric Kullback-Leibler Divergence

The symmetric version of KL divergence for external priors can be written as

$$\begin{aligned}
\jmath^{p}_{skl}(\mathbf{x}, q; f) =& H(q,p) + H(p,q) - H(p) = \\
=& -\sum_{j\in\mathcal{Y}} [p(j|\mathbf{x};\boldsymbol{\beta}) \log q(j|\mathbf{x}) + (q(j|\mathbf{x}) - p(j|\mathbf{x};\boldsymbol{\beta})) \log p(j|\mathbf{x};\boldsymbol{\beta})] = \\
=& -\sum_{j\in\mathcal{Y}} [p(j|\mathbf{x};\boldsymbol{\beta}) \log q(j|\mathbf{x}) + (q(j|\mathbf{x}) - p(j|\mathbf{x};\boldsymbol{\beta})) f_j(\mathbf{x};\boldsymbol{\beta})] = \\
=& -(q_{\mathbf{x}} - p_{\mathbf{x}})^T f(\mathbf{x};\boldsymbol{\beta}) - \sum_{j\in\mathcal{Y}} p(j|\mathbf{x};\boldsymbol{\beta}) \log q(j|\mathbf{x}),
\end{aligned} \tag{.18}$$

which follows from the fact that

$$\sum_{j\in\mathcal{Y}} (q(j|\mathbf{x}) - p(j|\mathbf{x};\boldsymbol{\beta})) \log \sum_{i\in\mathcal{Y}} e^{f_i(\mathbf{x};\boldsymbol{\beta})} = 0. \tag{.19}$$

Using the result from Eq.(.11)

$$\frac{\partial p(j|\mathbf{x};\boldsymbol{\beta})}{\partial f_k(\mathbf{x})} = p(j|\mathbf{x};\boldsymbol{\beta})(\mathbb{I}(k=j) - p(k|\mathbf{x};\boldsymbol{\beta})), \tag{.20}$$

we can compute the gradients as

$$\begin{aligned}\frac{\partial J^p_{skl}(\mathbf{x}, q; f)}{\partial f_k(\mathbf{x})} =& - q(k|\mathbf{x}) + p(k|\mathbf{x};\boldsymbol{\beta}) + \\ &+ \sum_{j\in\mathcal{Y}} p(j|\mathbf{x};\boldsymbol{\beta})(\mathbb{I}(k=j) - p(k|\mathbf{x};\boldsymbol{\beta}))(f_j(\mathbf{x};\boldsymbol{\beta}) - \log q(j|\mathbf{x})) = \\ =& - q(k|\mathbf{x}) + p(k|\mathbf{x};\boldsymbol{\beta}) + p(k|\mathbf{x};\boldsymbol{\beta})(\log \frac{p(k|\mathbf{x};\boldsymbol{\beta})}{q(k|\mathbf{x})} - D_{KL}(p\|q)). \end{aligned} \tag{.21}$$

As with the KL divergence based loss function, here we show that the sum of the sample weights is zero.

Theorem .0.2. *Let*

$$d_{\mathbf{x},k} = -\frac{\partial J^p_{skl}(\mathbf{x}, q; f)}{\partial f_k(\mathbf{x})}. \tag{.22}$$

The sum of the weights over the classes is zero

$$\sum_{k\in\mathcal{Y}} d_{\mathbf{x},k} = 0. \tag{.23}$$

Proof. The proof is similar to the previous theorem:

$$\begin{aligned}\sum_{k\in\mathcal{Y}} d_{\mathbf{x},k} =& \sum_{k\in\mathcal{Y}} q(k|\mathbf{x}) - p(k|\mathbf{x};\boldsymbol{\beta}) - \\ &- \sum_{k\in\mathcal{Y}} p(k|\mathbf{x};\boldsymbol{\beta})(\log \frac{p(k|\mathbf{x};\boldsymbol{\beta})}{q(k|\mathbf{x})} - D_{KL}(p\|q)) = \\ =& - D_{KL}(p\|q) + D_{KL}(p\|q) \sum_{k\in\mathcal{Y}} p(k|\mathbf{x};\boldsymbol{\beta}) = 0. \end{aligned} \tag{.24}$$

□

Jensen-Shannon Divergence

The Jensen-Shannon divergence uses the average of the two distributions $m = \frac{1}{2}(p+q)$ as the common point for measuring the divergence of these two distributions. Let

$$m(j|\mathbf{x};\boldsymbol{\beta}) = \frac{p(j|\mathbf{x};\boldsymbol{\beta}) + q(j|\mathbf{x})}{2}. \tag{.25}$$

Then, the loss function can be written as

$$\begin{aligned}\jmath_{js}^{p}(\mathbf{x},q;f) =&H(q,m)+H(p,m)-H(p) = \\ =&-2\sum_{j\in\mathcal{Y}} m(j|\mathbf{x};\boldsymbol{\beta})\log m(j|\mathbf{x};\boldsymbol{\beta}) + \sum_{j\in\mathcal{Y}} p(j|\mathbf{x};\boldsymbol{\beta})\log p(j|\mathbf{x};\boldsymbol{\beta}). \end{aligned} \tag{.26}$$

The gradients can be easily computed as

$$\begin{aligned}\frac{\partial \jmath_{js}^{p}(\mathbf{x},q;f)}{\partial f_k(\mathbf{x})} =&\sum_{j\in\mathcal{Y}} p(j|\mathbf{x};\boldsymbol{\beta})(\mathbb{I}(k=j)-p(k|\mathbf{x};\boldsymbol{\beta}))\Big(\log p(j|\mathbf{x};\boldsymbol{\beta}) - \log m(j|\mathbf{x};\boldsymbol{\beta})\Big) = \\ =&p(k|\mathbf{x};\boldsymbol{\beta})(\log\frac{p(k|\mathbf{x};\boldsymbol{\beta})}{m(k|\mathbf{x};\boldsymbol{\beta})} - D_{KL}(p\|m)). \end{aligned} \tag{.27}$$

Again, we are able to show that the sum of the gradients is zero.

Theorem .0.3. *Let*

$$d_{\mathbf{x},k} = -\frac{\partial \jmath_{js}^{p}(\mathbf{x},q;f)}{\partial f_k(\mathbf{x})}. \tag{.28}$$

The sum of the weights over the classes is zero

$$\sum_{k\in\mathcal{Y}} d_{\mathbf{x},k} = 0. \tag{.29}$$

Proof. The proof follows the previous two theorems:

$$\begin{aligned}\sum_{k\in\mathcal{Y}} d_{\mathbf{x},k} =& -\sum_{k\in\mathcal{Y}} p(k|\mathbf{x};\boldsymbol{\beta})(\log\frac{p(k|\mathbf{x};\boldsymbol{\beta})}{m(k|\mathbf{x};\boldsymbol{\beta})} - D_{KL}(p\|m)) = \\ =& -D_{KL}(p\|m) + D_{KL}(p\|m)\sum_{k\in\mathcal{Y}} p(k|\mathbf{x};\boldsymbol{\beta}) = 0. \end{aligned} \tag{.30}$$

□

Manifold Regularization

For the manifold regularization term

$$\jmath(\mathcal{X}_u;\boldsymbol{\beta}) = \sum_{\mathbf{x}\in\mathcal{X}_u}\sum_{\substack{\mathbf{x}'\in\mathcal{X}_u\\ \mathbf{x}'\neq\mathbf{x}}} s(\mathbf{x},\mathbf{x}')\jmath_m(\mathbf{x},\mathbf{x}';\boldsymbol{\beta}), \tag{.31}$$

the sample $\mathbf{x}$ appears once in the outer sum and additionally, when it is a neighboring point of another sample. Therefore, we need to develop the gradient terms with respect to both of these appearances.

In details, since the KL divergence is not symmetric with respect to the order of the probability distributions, this means that we need to consider the following gradients separately

$$\frac{\partial \jmath^m(\mathbf{x}, \mathbf{x}'; f)}{\partial f_k(\mathbf{x})}, \quad \frac{\partial \jmath^m(\mathbf{x}', \mathbf{x}; f)}{\partial f_k(\mathbf{x})}. \tag{.32}$$

However, for the SKL and JS divergences, due to symmetry we only need to develop one of these terms. Given these two gradients, we can write the overall gradients as

$$\frac{\partial \jmath(\mathcal{X}_u; \boldsymbol{\beta})}{\partial f_k(\mathbf{x})} = \sum_{\substack{\mathbf{x}' \in \mathcal{X}_u \\ \mathbf{x}' \neq \mathbf{x}}} s(\mathbf{x}, \mathbf{x}') \frac{\partial \jmath^m(\mathbf{x}, \mathbf{x}'; f)}{\partial f_k(\mathbf{x})} + s(\mathbf{x}', \mathbf{x}) \frac{\partial \jmath^m(\mathbf{x}', \mathbf{x}; f)}{\partial f_k(\mathbf{x})}. \tag{.33}$$

Note that in practice, we usually use symmetric similarity measures $s(\mathbf{x}, \mathbf{x}') = s(\mathbf{x}', \mathbf{x})$. Therefore, without loss of generality, we assume that this is the case.

Kullback-Leibler Divergence

For KL divergence, the gradients can be written as

$$\frac{\partial \jmath_{KL}(\mathcal{X}_u; \boldsymbol{\beta})}{\partial f_k(\mathbf{x})} = \sum_{\substack{\mathbf{x}' \in \mathcal{X}_u \\ \mathbf{x}' \neq \mathbf{x}}} s(\mathbf{x}, \mathbf{x}') \Big(\frac{\partial \jmath^m_{kl}(\mathbf{x}, \mathbf{x}'; f)}{\partial f_k(\mathbf{x})} + \frac{\partial \jmath^m_{kl}(\mathbf{x}', \mathbf{x}; f)}{\partial f_k(\mathbf{x})} \Big). \tag{.34}$$

Note that based on the definition of the symmetric KL divergence, this is equivalent to

$$\frac{\partial \jmath_{KL}(\mathcal{X}_u; \boldsymbol{\beta})}{\partial f_k(\mathbf{x})} = 2 \sum_{\substack{\mathbf{x}' \in \mathcal{X}_u \\ \mathbf{x}' \neq \mathbf{x}}} s(\mathbf{x}, \mathbf{x}') \frac{\partial \jmath^m_{skl}(\mathbf{x}, \mathbf{x}'; f)}{\partial f_k(\mathbf{x})}. \tag{.35}$$

Therefore, the derivation for the KL divergence are the same as the SKL presented in next section.

Symmetric Kullback-Leibler Divergence

Note that we can write the SKL as

$$\jmath^m_{skl}(\mathbf{x}, \mathbf{x}'; f) = \sum_{j \in \mathcal{Y}} (p(j|\mathbf{x}; \boldsymbol{\beta}) - p(j|\mathbf{x}'; \boldsymbol{\beta}))(\log p(j|\mathbf{x}; \boldsymbol{\beta}) - \log p(j|\mathbf{x}'; \boldsymbol{\beta})). \tag{.36}$$

Therefore, we can write the gradients as

$$\frac{\partial \jmath_{SKL}(\mathcal{X}_u; \boldsymbol{\beta})}{\partial f_k(\mathbf{x})} = 2 \sum_{\substack{\mathbf{x}' \in \mathcal{X}_u \\ \mathbf{x}' \neq \mathbf{x}}} s(\mathbf{x}, \mathbf{x}') \frac{\partial \jmath^m_{skl}(\mathbf{x}, \mathbf{x}'; f)}{\partial f_k(\mathbf{x})}, \tag{.37}$$

where

$$\begin{aligned} \frac{\partial \jmath^m_{skl}(\mathbf{x}, \mathbf{x}'; f)}{\partial f_k(\mathbf{x})} =& \sum_{j \in \mathcal{Y}} (\log p(j|\mathbf{x}; \boldsymbol{\beta}) - \log p(j|\mathbf{x}'; \boldsymbol{\beta})) \frac{\partial p(j|\mathbf{x}; \boldsymbol{\beta})}{\partial f_k(\mathbf{x})} + \\ &+ \sum_{j \in \mathcal{Y}} (p(j|\mathbf{x}; \boldsymbol{\beta}) - p(j|\mathbf{x}'; \boldsymbol{\beta})) \frac{1}{p(j|\mathbf{x}; \boldsymbol{\beta})} \frac{\partial p(j|\mathbf{x}; \boldsymbol{\beta})}{\partial f_k(\mathbf{x})} = \\ =& p(k|\mathbf{x}; \boldsymbol{\beta}) - p(k|\mathbf{x}'; \boldsymbol{\beta}) + p(k|\mathbf{x}; \boldsymbol{\beta})(\log \frac{p(k|\mathbf{x}; \boldsymbol{\beta})}{p(k|\mathbf{x}'; \boldsymbol{\beta})} - D_{KL}(p_{\mathbf{x}} \| p_{\mathbf{x}'})). \end{aligned} \tag{.38}$$

We can prove that the following theorem holds.

Theorem .0.4. *Let*

$$d_{\mathbf{x},k} = -\frac{\partial \jmath^m_{skl}(\mathbf{x}, q; f)}{\partial f_k(\mathbf{x})}. \tag{.39}$$

The sum of the weights over the classes is zero

$$\sum_{k \in \mathcal{Y}} d_{\mathbf{x},k} = 0. \tag{.40}$$

Proof. The proof is similar to the previous theorem:

$$\begin{aligned} \sum_{k \in \mathcal{Y}} d_{\mathbf{x},k} =& \sum_{k \in \mathcal{Y}} p(k|\mathbf{x}'; \boldsymbol{\beta}) - p(k|\mathbf{x}; \boldsymbol{\beta}) - \\ &- \sum_{k \in \mathcal{Y}} p(k|\mathbf{x}; \boldsymbol{\beta})(\log \frac{p(k|\mathbf{x}; \boldsymbol{\beta})}{p(k|\mathbf{x}'; \boldsymbol{\beta})} - D_{KL}(p_{\mathbf{x}} \| p_{\mathbf{x}'})) = \\ =& - D_{KL}(p_{\mathbf{x}} \| p_{\mathbf{x}'}) + D_{KL}(p_{\mathbf{x}} \| p_{\mathbf{x}'}) \sum_{k \in \mathcal{Y}} p(k|\mathbf{x}; \boldsymbol{\beta}) = 0. \end{aligned} \tag{.41}$$

□

Jensen-Shannon Divergence

For the Jensen-Shannon divergence let us define $m = \frac{1}{2}(p+q)$ as the common point for measuring the divergence of these two distributions. Denote

$$m(j|\mathbf{x};\boldsymbol{\beta}) = \frac{p(j|\mathbf{x};\boldsymbol{\beta}) + p(j|\mathbf{x}';\boldsymbol{\beta})}{2}. \tag{.42}$$

The the loss function can be defined as

$$j^m_{js}(\mathbf{x},\mathbf{x}';f) = 2H(m) - H(p_{\mathbf{x}}) - H(p_{\mathbf{x}'}), \tag{.43}$$

where H is the entropy of a distribution. Therefore, we have

$$\begin{aligned}\frac{\partial j^m_{js}(\mathbf{x},\mathbf{x}';f)}{\partial f_k(\mathbf{x})} =& -2\sum_{j\in\mathcal{Y}}(\log m(j|\mathbf{x};\boldsymbol{\beta})+1)\frac{\partial m(j|\mathbf{x};\boldsymbol{\beta})}{\partial f_k(\mathbf{x})}+\\ &+\sum_{j\in\mathcal{Y}}(\log p(j|\mathbf{x};\boldsymbol{\beta})+1)\frac{\partial p(j|\mathbf{x};\boldsymbol{\beta})}{\partial f_k(\mathbf{x})}=\\ &=\sum_{j\in\mathcal{Y}}\log\frac{p(j|\mathbf{x};\boldsymbol{\beta})}{m(j|\mathbf{x};\boldsymbol{\beta})}p(j|\mathbf{x};\boldsymbol{\beta})(\mathbb{I}(k=j)-p(k|\mathbf{x};\boldsymbol{\beta}))=\\ &=p(k|\mathbf{x};\boldsymbol{\beta})(\log\frac{p(k|\mathbf{x};\boldsymbol{\beta})}{m(k|\mathbf{x}';\boldsymbol{\beta})} - D_{KL}(p\|m)).\end{aligned} \tag{.44}$$

It is easy to prove that the sum of the gradients is zero.

Theorem .0.5. *Let*

$$d_{\mathbf{x},k} = -\frac{\partial j^m_{js}(\mathbf{x},q;f)}{\partial f_k(\mathbf{x})}. \tag{.45}$$

The sum of the weights over the classes is zero

$$\sum_{k\in\mathcal{Y}} d_{\mathbf{x},k} = 0. \tag{.46}$$

Proof. The proof is similar to the previous theorem:

$$\begin{aligned}\sum_{k\in\mathcal{Y}} d_{\mathbf{x},k} =& -\sum_{k\in\mathcal{Y}} p(k|\mathbf{x};\boldsymbol{\beta})(\log\frac{p(k|\mathbf{x};\boldsymbol{\beta})}{m(k|\mathbf{x};\boldsymbol{\beta})} - D_{KL}(p\|m)) =\\ =& -D_{KL}(p\|m) + D_{KL}(p\|m)\sum_{k\in\mathcal{Y}} p(k|\mathbf{x};\boldsymbol{\beta}) = 0.\end{aligned} \tag{.47}$$

□

C: List of Publications

The following is the list of research publications I, together with my colleagues, have made or, at the time of writing this thesis was in under review process for publication, and is organized in reversed chronological order.

JMLR [47]	Isabelle Guyon, Amir Saffari, Gideon Dror, Gavin Cawley, **Model Selection: Beyond the Bayesian/Frequentist Divide**, Journal of Machine Learning Research (JMLR), Vol. 11, Pages 61-87, 2010.
CVPR [84]	Amir Saffari, Martin Godec, Thomas Pock, Christian Leistner, Horst Bischof, **Online Multi-Class LPBoost**, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010.
CVPR [109]	Bernhard Zeisl, Christian Leistner , Amir Saffari , Horst Bischof, **On-line Semi-Supervised Multiple-Instance Boosting**, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010.
CVPR [89]	Jakob Santner, Christian Leistner, Amir Saffari, Thomas Pock, Horst Bischof, **PROST: Parallel Robust Online Simple Tracking**, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010.
ICPR [38]	Martin Godec, Christian Leistner, Amir Saffari, Horst Bischof, **On-line Random Naive Bayes for Tracking**, International Conference on Pattern Recognition (ICPR), 2010.
ECCV	Amir Saffari, Christian Leistner, Martin Godec, Horst Bischof, **Robust Multi-View Boosting with Priors**, *submitted to* European Conference on Computer Vision (ECCV), 2010.
ECCV	Christian Leistner, Amir Saffari, Horst Bischof, **MILForests: Multiple Instance Learning with Randomized Trees**, *submitted to* European Conference on Computer Vision (ECCV), 2010.
PR	Christian Leistner, Amir Saffari, Martin Godec, Bernhard Zeisl, Horst Bischof, **On-line Semi-Supervised Boosting**, *submitted to* Pattern Recognition, Special Issue on Semi-Supervised Learning for Visual Content Analysis and Understanding, 2010.

CVPR [87]	Amir Saffari, Christian Leistner, Horst Bischof, **Regularized Multi-Class Semi-Supervised Boosting**, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Oral Presentation, 2009.
ICCV [58]	Christian Leistner, Amir Saffari, Jakob Santner, Horst Bischof, **Semi-Supervised Random Forests**, Proceedings of IEEE International Conference on Computer Vision (ICCV), 2009.
BMVC [90]	Jakob Santner, Markus Unger, Thomas Pock, Christian Leistner, Amir Saffari, Horst Bischof, **Interactive Texture Segmentation using Random Forests and Total Variation**, British Machine Vision Conference (BMVC), 2009.
ICCV-OLCV [88]	Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, Horst Bischof, **On-line Random Forests**, 3rd IEEE ICCV Workshop on On-line Learning for Computer Vision, 2009.
ICCV-OLCV [57]	Christian Leistner, Amir Saffari, Peter Roth, Horst Bischof, **On Robustness of On-line Boosting A Competitive Study**, 3rd IEEE ICCV Workshop on On-line Learning for Computer Vision, 2009.
OAGM [54]	Inayatullah Khan, Amir Saffari, Horst Bischof, **TVGraz: Multi-Modal Learning of Object Categories by Combining Textual and Visual Features**, Proc. 33rd Workshop of the Austrian Association for Pattern Recognition, AAPR / OAGM, 2009.
ECCV [85]	Amir Saffari, Helmut Grabner, Horst Bischof, **SERBoost: Semi-supervised Boosting with Expectation Regularization**, Proceedings of European Conference on Computer Vision (ECCV), 2008.
NN [46]	Isabelle Guyon, Amir Saffari, Gideon Dror, Gavin Cawley, **Analysis of the IJCNN 2007 Agnostic Learning vs. Prior Knowledge Challenge**, Neural Networks, Vol. 21, Pages 544-550, 2008.
DAGM [83]	Amir Saffari, Horst Bischof, **Boosting for Model-Based Data Clustering**, Proc. of 30th Symposium of the German Association for Pattern Recognition (DAGM 2008), Oral Presentation, Pages 51-60, 2008.

CVWW [82]	Amir Saffari, Horst Bischof, **Clustering in a Boosting Framework**, Proc. of Computer Vision Winter Workshop (CVWW), St. Lambrecht, Austria, Pages 75-82, 2007.
IJCNN [45]	Isabelle Guyon, Amir Saffari, Gideon Dror, Gavin Cawley, **Agnostic Learning vs. Prior Knowledge Challenge**, Proc. of IEEE International Joint Conference on Neural Networks (IJCNN), Orlando, Florida, USA, Pages 829-834, 2007.
NIPS-Demo [49]	Isabelle Guyon, Amir Saffari, Hugo Escalante, Gokhan Bakir, Gavin Cawley, **CLOP: a Matlab Learning Object Package**, NIPS 2007 Demonstrations, Vancouver, British Columbia, Canada, 2007.
ICG-Tech [86]	Amir Saffari, Isabelle Guyon, **Quick Start Guide for CLOP**, Technical Report, Institute for Computer Graphics and Vision, Graz University of Technology and Clopinet, 2006.
ICG-Tech [81]	Amir Saffari, Horst Bischof, **Video Tracking Red Light Enforcement**, Technical Report, Institute for Computer Graphics and Vision, Graz University of Technology, 2006.
NIPS-MLI [48]	Isabelle Guyon, Amir Saffari, Gideon Dror, Gavin Cawley, Olivier Guyon, **NIPS 2006 Model Selection Game**, NIPS Workshop on Multi-level Inference, Vancouver, BC, Canada, 2006.
WCCI [44]	Isabelle Guyon, Amir Saffari, Gideon Dror, Joachim Buhmann, **Performance Prediction Challenge**, Proc. of International Joint Conference on Neural Networks (IJCNN), IEEE World Congress on Computational Intelligence (WCCI), Vancouver, British Columbia, Canada, Pages 2958-2965, 2006.
IJCIA [80]	Amir Saffari, **Book Review: Complex Worlds from Simpler Nervous Systems**, International Journal of Computational Intelligence and Applications (IJCIA), Vol. 6, Pages 569-572, 2006.
NIPS-IFUM [73]	Michael Pfeiffer, Amir Saffari, Andreas Juffinger, **Predicting Text Relevance from Sequential Reading Behavior**, Proc. of the NIPS Workshop on Machine Learning for Implicit Feedback and User Modeling, Whistler, British Columbia, Canada, 2005. Challenge Winner.

Bibliography

[1] A. Adam, E. Rivlin, and I. Shimshoni.
Robust fragments-based tracking using the integral histogram.
In *CVPR*, 2006.

[2] Erin L. Allwein, Robert E. Schapire, and Yoram Singer.
Reducing multiclass to binary: A unifying approach for margin classifiers.
Journal of Machine Learning Research, 1:113–141, December 2000.

[3] S. Avidan.
Ensemble tracking.
In *Proc. CVPR*, volume 2, pages 494–501, 2005.

[4] Arik Azran.
The rendezvous algorithm: multiclass semi-supervised learning with markov random walks.
In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 49–56, New York, NY, USA, 2007. ACM Press.

[5] B. Babenko, M. H. Yang, and S. Belongie.
Visual tracking with online multiple instance learning.
In *CVPR*, 2009.

[6] H. Bekel, I. Bax, G. Heidemann, and H. Ritter.
Adaptive computer vision: Online learning for object recognition.
In *DAGM*, pages 447–454, 2004.

[7] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani.
Manifold regularization: A geometric framework for learning from labeled and unlabeled examples.
Journal of Machine Learning Research, 7:2399–2434, November 2006.

[8] Kristin P. Bennett, Ayhan Demiriz, and Richard Maclin.
Exploiting unlabeled data in ensemble methods.
In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 289–296, 2002.

[9] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra.
A maximum entropy approach to natural language processing.
Comput. Linguist., 22(1):39–71, March 1996.

[10] Avrim Blum.
On-line algorithms in machine learning.
In *Online Algorithms*, pages 306–325, 1996.

[11] Avrim Blum and Tom Mitchell.
Combining labeled and unlabeled data with co-training.
In *COLT: Proceedings of the Workshop on Computational Learning Theory*, pages 92–100, 1998.

[12] A. Bordes, L. Bottou, P. Gallinari, and J. Weston.
Solving multiclass support vector machines with larank.
In *ICML*, 2007.

[13] Ulf Brefeld, Christoph Büscher, and Tobias Scheffer.
Multi-view discriminative sequential learning.
In *ECML*, pages 60–71. 2005.

[14] Leo Breiman.
Bagging predictors.
In *Machine Learning*, volume 24, pages 123–140, 1996.

[15] Leo Breiman.
Random forests.
Machine Learning, 45(1):5–32, October 2001.

[16] D'alche F. Buc, Y. Grandvalet, and C. Ambroise.
Semi-supervised marginboost.
In *Advances in Neural Information Processing Systems (NIPS)*, pages 553–560, 2002.

[17] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina.
An empirical evaluation of supervised learning in high dimensions.
In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 96–103, New York, NY, USA, 2008. ACM.

[18] B. Cestnik.
Estimating probabilities: A crucial task in machine learning.
In *Proc. of the European Conference on Artificial Intelligenc*, pages 147–149, 1990.

[19] Chih C. Chang and Chih J. Lin.
Libsvm: a library for support vector machines, 2001.

[20] O. Chapelle, B. Schölkopf, and A. Zien.
Semi-Supervised Learning.
Cambridge, MA, 2006.

[21] O. Chapelle, J. Weston, and B. Schölkopf.
Cluster kernels for semi-supervised learning.
In *NIPS*, volume 15 of *NIPS*, pages 585–592, 2003.

[22] Olivier Chapelle and Alexander Zien.
Semi-supervised classification by low density separation, 2005.

[23] C. Mario Christoudias, Raquel Urtasun, and Trevor Darrell.
Unsupervised distributed feature selection for multi-view object recognition.
In *CVPR*, 2008.

[24] M. Collins and Y. Singer.
Unsupervised models for named entity classification.
In *EMNLP*, 1999.

[25] R. T. Collins, Y. Liu, and M. Leordeanu.
Online selection of discriminative tracking features.
PAMI, 27:1631–1643, 2005.

[26] B. V. Dasarathy and B. V. Sheela.
Composite classifier system design: concepts and methodology.
In *Proceedings of the IEEE*, volume 67, pages 708–713, 1979.

[27] Ayhan Demiriz, Kristin P. Bennett, and John S. Taylor.
Linear programming boosting via column generation.
Machine Learning, 46(1-3):225–254, 2002.

[28] Thomas Dietterich.
An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization.
In *Bagging, boosting, and randomization. Machine Learning*, volume 40, pages 139–157, 1998.

[29] Carlos Domingo and Osamu Watanabe.
Conference on computational learning theory (colt): A modification of adaboost, 2000.

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge 2007 (voc2007) results.

[31] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The pascal visual object classes challenge 2006 (voc2006) results. Technical report, 2006.

[32] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML)*, pages 148–156, 1996.

[33] Yoav Freund. An adaptive version of the boost by majority algorithm. In *In Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, volume 43, pages 102–113, 2000.

[34] Jerome Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[35] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 38(2):337–374, 2000.

[36] Peter Gehler and Sebastian Nowozin. On feature combination for multiclass object classification. In *ICCV*, 2009.

[37] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. In *Machine Learning*, volume 63, pages 3–42, 2006.

[38] Martin Godec, Christian Leistner, Amir Saffari, and Horst Bischof. On-line random naive bayes for tracking. In *IEEE International Conference on Pattern Recognition (ICPR)*, 2010.

[39] H. Grabner and H. Bischof. On-line boosting and vision. In *CVPR*, pages 260–267, 2006.

[40] H. Grabner and H. Bischof. On-line boosting and vision. In *Proc. CVPR*, volume 1, pages 260–267, 2006.

[41] H. Grabner, C. Leistner, and H. Bischof.
On-line semi-supervised boosting for robust tracking.
In *ECCV*, 2008.

[42] E. Granger, Y. Savaria, and P. Lavoie.
A pattern reordering approach based on ambiguity detection for online category learning.
PAMI, 25:525–529, 2003.

[43] V. Guruswami and Sahai.
Multiclass learning, boosting, and error-correcting codes.
In *COLT*, 1999.

[44] Isabelle Guyon, Amir Saffari, Gideon Dror, and Joachim Buhmann.
Performance prediction challenge.
In *Proc. of International Joint Conference on Neural Networks (IJCNN), IEEE World Congress on Computational Intelligence (WCCI), Vancouver, British Columbia, Canada*, pages 2958–2965, July 2006.

[45] Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley.
Agnostic learning vs. prior knowledge challenge.
In *Proc. of International Joint Conference on Neural Networks (IJCNN), Orlando, Florida, USA*, pages 829–834, August 2007.

[46] Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley.
Analysis of the ijcnn 2007 agnostic learning vs. prior knowledge challenge.
Neural Networks, 21:544–550, 2008.

[47] Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley.
Model selection: Beyond the bayesian/frequentist divide.
Journal of Machine Learning Research, 11:61–87, 2010.

[48] Isabelle Guyon, Amir Saffari, Gideon Dror, Gavin Cawley, and Olivier Guyon.
Results of the nips 2006 model selection game, December 2006.

[49] Isabelle Guyon, Amir Saffari, Hugo Escalante, Gökhan Bakir, and Gavin Cawley.
Clop: a matlab learning object package, December 2007.

[50] O. Javed, S. Ali, and M. Shah.
Online detection and classification of moving objects using progressively improving detectors.
In *CVPR*, pages 695–700, 2005.

[51] E. T. Jaynes.
Information theory and statistical mechanics.
Physical Review, 106(4):620–630, 1957.

[52] Thorsten Joachims.
Transductive inference for text classification using support vector machines.
In *Proceedings of International Conference on Machine Learning (ICML)*, pages 200–209, 1999.

[53] S. S. Keerthi, S. Sundararajan, K. W. Chang, C. J. Hsieh, and C. J. Lin.
A sequential dual coordinate method for large-scale multi-class linear svms.
In *KDD*, 2008.

[54] Inayatullah Khan, Amir Saffari, and Horst Bischof.
Tvgraz: Multi-modal learning of object categories by combining textual and visual features.
In *Proc. 33rd Workshop of the Austrian Association for Pattern Recognition*, 2009.

[55] S. Lazebnik, C. Schmid, and J. Ponce.
Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories.
In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2169–2178, 2006.

[56] Christian Leistner, Amir Saffari, Martin Godec, Bernhard Zeisl, and Horst Bischof.
On-line semi-supervised boosting.
submitted to Pattern Recognition, Special Issue on On-line Learning, 2010.

[57] Christian Leistner, Amir Saffari, Peter M. Roth, and Horst Bischof.
On robustness of on-line boosting - a competitive study.
In *3rd IEEE ICCV Workshop on On-line Computer Vision*, 2009.

[58] Christian Leistner, Amir Saffari, Jakob Santner, and Horst Bischof.
Semi-supervised random forests.
In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[59] Boaz Leskes and Leen Torenvliet.
The value of agreement a new boosting algorithm.
J. Comput. Syst. Sci., 74(4):557–586, 2008.

[60] Anat Levin, Paul Viola, and Yoav Freund.
Unsupervised improvement of visual detectors using co-training.
In *ICCV*, volume I, pages 626–633, 2003.

[61] Yi Lin.
A note on margin-based loss functions in classification.
Statistics & Probability Letters, 68(1):73–82, June 2004.

[62] R. Liu, J. Cheng, and H. Lu.
A robust boosting tracker with minimum error bound in a co-training framework.
In *ICCV*, 2009.

[63] N. Loeff, D. Forsyth, and D. Ramachandran.
Manifoldboost: stagewise function approximation for fully-, semi- and unsupervised learning.
In *Proceedings of International Conference on Machine Learning (ICML)*, pages 600–607, 2008.

[64] Philip M. Long and Rocco A. Servedio.
Random classification noise defeats all convex potential boosters.
In *ICML*, volume 307, pages 608–615, 2008.

[65] Richard Maclin and David Opitz.
An empirical evaluation of bagging and boosting.
In *In Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 546–551, 1997.

[66] Pavan K. Mallapragada, Rong Jin, Anil K. Jain, and Yi Liu.
Semiboost: Boosting for semi-supervised learning.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(11):2000–2014, September 2009.

[67] Gideon S. Mann and Andrew Mccallum.
Simple, robust, scalable semi-supervised learning via expectation regularization.
In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 593–600, 2007.

[68] L. Mason, J. Baxter, P. Bartlett, and M. Frean.
Advances in Large Margin Classifiers, pages 221–247.
MIT Press, Cambridge, MA., 1999.

[69] Andrew Mccallum, Gideon Mann, and Gregory Druck.
Generalized expectation criteria.
Technical report, August 2007.

[70] Kamal Nigam and Rayid Ghani.
Analyzing the effectiveness and applicability of co-training.

In *CIKM*, pages 86–93, 2000.

[71] Jorge Nocedal and Stephen Wright.
Numerical Optimization.
Springer, April 2000.

[72] N. Oza and S. Russell.
Online bagging and boosting.
In *AISTAT*, pages 105–112, 2001.

[73] Michael Pfeiffer, Amir Saffari, and Andreas Juffinger.
Predicting text relevance from sequential reading behavior.
In Kai Puolamaki and Samuel Kaski, editors, *Proc. of the NIPS 2005 Workshop on Machine Learning for Implicit Feedback and User Modeling, Whistler, British Columbia, Canada*, pages 25–30, December 2005.

[74] M. T. Pham and T. J. Cham.
Online asymetric boosted clasifiers for object detection.
In *CVPR*, 2007.

[75] G. Rätsch, T. Onoda, and K. R. Müller.
Soft margins for adaboost.
Machine Learning, 42(3):287–320, March 2001.

[76] Simon Rogers and Mark Girolami.
Multi-class semi-supervised learning with the e-truncated multinomial probit gaussian process.
JMLR, 1:17–32, 2007.

[77] Chuck Rosenberg, Martial Hebert, and Henry Schneiderman.
Semi-supervised self-training of object detection models.
In *WACV-MOTION '05: Proceedings of the Seventh IEEE Workshops on Application of Computer Vision (WACV/MOTION'05) - Volume 1*, pages 29–36, Washington, DC, USA, 2005. IEEE Computer Society.

[78] D. Ross, J. Lim, R.-S. Lin, and M.-H. Yang.
Incremental learning for robust visual tracking.
IJCV, 2008.

[79] Saharon Rosset, Ji Zhu, Trevor Hastie, and Robert Schapire.
Boosting as a regularized path to a maximum margin classifier.
Journal of Machine Learning Research, 5:941–973, 2004.

[80] Amir Saffari.
Book review: Complex worlds from simpler nervous systems, by frederick r. prete, editor, mit press 2004.
International Journal of Computational Intelligence and Applications (IJCIA), 6(4):569–572, December 2006.

[81] Amir Saffari and Horst Bischof.
Video tracking - red light enforcement.
Technical report, 2006.

[82] Amir Saffari and Horst Bischof.
Clustering in a boosting framework.
In *Proc. of Computer Vision Winter Workshop (CVWW), St. Lambrecht, Austria*, pages 75–82, February 2007.

[83] Amir Saffari and Horst Bischof.
Boosting for model-based data clustering.
In *Proc. of 30th Symposium of the German Association for Pattern Recognition (DAGM 2008)*, June 2008.

[84] Amir Saffari, Martin Godec, Thomas Pock, Christian Leistner, and Horst Bischof.
Online multi-class lpboost.
In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[85] Amir Saffari, Helmut Grabner, and Horst Bischof.
Serboost: Semi-supervised boosting with expectation regularization.
In *Proceedings of European Conference on Computer Vision (ECCV)*, October 2008.

[86] Amir Saffari and Isabelle Guyon.
Quick start guide for clop.
Technical report, 2006.

[87] Amir Saffari, Christian Leistner, and Horst Bischof.
Regularized multi-class semi-supervised boosting.
In *IEEE Conference on Computer Vision and Pattern Recogntion*, 2009.

[88] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof.
On-line random forests.
In *3rd IEEE ICCV Workshop on On-line Computer Vision*, 2009.

[89] Jakob Santner, Christian Leistner, Amir Saffari, Thomas Pock, and Horst Bischof.
Prost: Parallel robust online simple tracking.

In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[90] Jakob Santner, Markus Unger, Thomas Pock, Christian Leistner, Amir Saffari, and Horst Bischof.
Interactive texture segmentation using random forests and total variation.
In *British Machine Vision Conference (BMVC)*, 2009.

[91] Robert E. Schapire.
The strength of weak learnability.
Machine Learning, 5(2):197–227, June 1990.

[92] Shai Shalev-Shwartz.
Online Learning: Theory, Algorithms, and Applications.
PhD thesis, The Hebrew University of Jerusalem, July 2007.

[93] Chunhua Shen and Hanxi Li.
On the dual formulation of boosting algorithms.
IEEE Transactions on Pattern Analysis and Machine Intelligence, Dec 2010.

[94] Hamed M. Shirazi and Nuno Vasconcelos.
On the design of loss functions for classification: theory, robustness to outliers, and savageboost.
In *NIPS*, pages 1049–1056, 2008.

[95] Vikas Sindhwani and Sathiya S. Keerthi.
Large scale semi-supervised linear svms.
In *In Proceedings of the International ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 477–484, 2006.

[96] Vikas Sindhwani and David S. Rosenberg.
An rkhs for multi-view learning and manifold co-regularization.
In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 976–983, 2008.

[97] Yangqiu Song, Changshui Zhang, and Jianguo Lee.
Graph based multi-class semi-supervised learning using gaussian process.
In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 450–458. 2006.

[98] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf.
Large scale multiple kernel learning.
Journal of Machine Learning Research, 7:1531–1565, July 2006.

[99] S. Sun and Q. Zhang.

Multiple-view multiple-learner semi-supervised learning.
Technical report, 2007.

[100] F. Tang, S. Brennan, Q. Zhao, and H. Tao.
Co-tracking using semi-supervised support vector machines.
In *ICCV*, 2007.

[101] A. Torralba, K. P. Murphy, and W. T. Freeman.
Sharing visual features for multiclass and multiview object detection.
Pattern Analysis and Machine Intelligence, IEEE Transactions on, 29(5):854–869, 2007.

[102] Hamed Valizadegan, Rong Jin, and Anil K. Jain.
Semi-supervised boosting for multi-class classification.
In *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases*, pages 522–537, 2008.

[103] Manfred K. Warmuth, Karen Glocer, and Gunnar Rätsch.
Boosting algorithms for maximizing the soft margin.
In *NIPS*, 2007.

[104] Manfred K. Warmuth, Karen A. Glocer, and S. V. Vishwanathan.
Entropy regularized lpboost.
In *ALT*, pages 256–271, Berlin, Heidelberg, 2008. Springer-Verlag.

[105] Jason Weston, Christina Leslie, Eugene Ie, Dengyong Zhou, Andre Elisseeff, and William S. Noble.
Semi-supervised protein classification using cluster kernels.
Bioinformatics, 21:3241–3247, 2005.

[106] B. Wu and R. Nevatia.
Improving part based object detection by unsupervised, online boosting.
In *CVPR*, 2007.

[107] Linli Xu and Dale Schuurmans.
Unsupervised and semi-supervised multi-class support vector machines.
In *AAAI*, 2005.

[108] David Yarowsky.
Unsupervised word sense disambiguation rivaling supervised methods.
In *In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, 1995.

[109] Bernhard Zeisl, Christian Leistner, Amir Saffari, and Horst Bischof.
Online semi-supervised multiple-instance boosting.
In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[110] B. Zhang, G. Ye, Y. Wang, J. Xu, and G. Herman.
Finding shareable informative patterns and optimal coding matrix for multiclass boosting.
In *ICCV*, 2009.

[111] Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie.
Multi-class adaboost.
Technical report, 2006.

[112] X. Zhu and Z. Ghahramani.
Learning from labeled and unlabeled data with label propagation, 2002.

[113] Xiaojin Zhu.
Semi-supervised learning literature survey.
Technical report, 2008.

[114] Martin Zinkevich.
Online convex programming and generalized infinitesimal gradient ascent.
In *ICML*, 2003.

[115] Hui Zou, Ji Zhu, and Trevor Hastie.
New multi-category boosting algorithms based on multi-category fisher-consistent losses.
Annals of Applied Statistics, 2008.

Index