Online learning of multi-class Support Vector Machines

Xuan Tuan Trinh

October 10, 2012

Abstract

Support Vector Machines (SVMs) are state-of-the-art learning algorithms for classification problems due to their strong theoretical foundation and their good performance in practice. However, their extension from two-class to multi-class classification problems is not straightforward. While some approaches solve a series of binary problems, other, theoretically more appealing methods solve one single optimization problem. Training SVMs amounts to solving a convex quadratic optimization problem. But even with a carefully tailored quadratic program solver, training all-in-one multi-class SVMs takes a long time for large scale datasets. We first consider the problem of training the multi-class SVM proposed by Lee, Lin and Wahba (LLW), which is the first Fisher consistent multi-class SVM that has been proposed in the literature and has recently been shown to exhibit good generalization performance on benchmark problems. Inspired by previous work on online optimization of binary and multi-class SVMs, a fast approximative online solver for the LLW SVM is derived. It makes use of recent developments for efficiently solving all-in-one multi-class SVMs without bias. In particular, it uses working sets of size two instead of following the paradigm of sequential minimal optimization. After successful implementation of the online LLW SVM solver, it is modified to also support the popular multi-class SVM formulation by Crammer and Singer. This is done using a recently established unified framework for a larger class of all-in-one multi-class SVMs. This makes it very easy to in the future adapt the novel online solver to even more formulations of multi-class SVMs. The results suggest that online solvers may provide for a robust and well-performing way of obtaining an approximative solution to the full problem which constitutes a good trade-off between optimization time and reaching a sufficiently high dual value.

Acknowledgements

Contents

Chapter 1

Introduction

Machine learning is an active discipline in Computer Science which aims to enable computers to learn from data and make predictions based on this. A large number of accurate and efficient learning algorithms exist, which have been applied in various areas from search engines and natural language processing to bioinformatics. However, with recent advances in technology, a vast majority of data is collected, and stored. As a consequence, there is a high demand for learning algorithms that are fast and have low memory requirements when trained on large scale dataset.

Among learning algorithms, Support Vector Machines (SVMs) are state-of-the-art for classification problems due to their theoretical basis and their high accuracy in practice. Originally for binary classification, Support Vector Machines have been extended to handle multi-category classification by solving a series of binary problems like the one-versus-all (OVA) method or by solving a single problem in all-in-one methods. The OVA method combines separately trained binary SVM classifiers into a classifier for multiple classes. This method has the advantage of being simple to implement. However, it has a disadvantage that even if all binary classifiers are consistent, the resulting OVA classifier is inconsistent [13]. In all-in-one approaches, the multi-class problem is formulated and solved in one single optimization problem. There are three all-in-one approaches that will be presented in this thesis, namely Weston and Watkins (WW), Crammer and Singer (CS) and Lee, Lin and Wahba (LLW). These approaches are more theoretical involved and have been found to be able to lead to better generalizing classifiers compared to OVA in a comparison study [13]. However, in comparison, training multi-class SVMs in all-in-one approach normally tend to need longer time. Unfortunately, this is especially true for the LLW SVM, which is the first Fisher consistent multi-class SVM that has been proposed in the literature [24].

We will address the training time problem by employing online learning methods. Online learning methods iteratively adjust the parameters associated with incoming new and fresh examples when they come as a stream. Over the past years, online variants of binary and multi-class SVM solvers have been proposed and refined [5, 32, 3, 18, 6]. Most online SVMs provides an approximative solver for the full batch problem. Online SVMs can be preferable to standard SVMs even for batch learning. In particular, for large data sets we cannot afford to compute a close to optimal solution to the SVM optimization problem, but want to find a good approximation quickly. In the past, online SVMs were shown to be able to reach this.

The prominent online multi-class SVM LaRank [4] relies on the multi-class SVM formulation proposed by Crammer and Singer, which was in one study [13] found to performs worse than the theoretically more sound LLW algorithm. We will derive new online algorithm for the SVMs, in particular we will adapt LaRank paradigm for the LLW machine. This will lead to an online multi-class SVM that gets a fast approximation to a machine which is Fisher consistent. Furthermore, an unified framework is recently established, which unifies three all-in-one approaches [14]. As a result, we will develop an unified online learning solver based on the derived algorithm that can apply to the unification framework.

1.1 Structure of this thesis

This thesis aims at developing online learning algorithms for multi-class SVMs in all-in-one approaches, which transfer LaRank's concept to. Therefore, the thesis is organized as follows.

- **Chapter 2** will provide the general background that is used throughout the thesis. It discusses some important concepts in statistical learning theory. It also introduces the formulation of binary SVMs, all-in-one approaches for multi-class SVMs, and online algorithms for SVMs.

- **Chapter 3** will introduce the generalized quadratic optimization problem without equality constraints to which the formulation of WW and LLW SVMS can easily be converted to. In addition, this chapter introduces the sequential two-dimensional optimization (S2DO) method that is effective to solve the generalized optimization problem.

- **Chapter 4** will derive new online learning algorithm for the LLW SVM. It first introduces the dual form of the LLW SVM. After that, a new learning algorithm for the LLW SVM is established by combining online learning paradigm with S2DO method. We will assess the performance of the new online solver in comparison with the batch solver in this chapter.

- **Chapter 5** will propose an online solver for the unified framework for all-in-one approaches. The unification, that is established through the margin concept and loss function, is reviewed there. The online solver for the unified framework then is derived. The performance of the unified online solver is evaluated through a comparison between the online CS solver and the batch CS solver.

- **Chapter 6** concludes and summarizes this thesis and gives some suggestions for future work.

1.2 Contributions

The main contributions of this thesis are deriving online solvers for multi-class SVMs in all-in-one approaches. In detail, an online multi-class SVMs solver for LLW SVM is derived and evaluated. This is the first online solver that approximates a machine which is Fisher consistent. Additionally, it is the first time we have an online solver for the unified framework that unified the three all-in-one approaches. These online solver are implemented in Shark and several minor improvements of Shark are performed. The thesis also studies and compares the build-up behaviour of the online versus batch solvers, which gives insight into the open question if early-stopping of a batch solver would not be an equivalent alternative method.

Chapter 2

Background

In this chapter, we will review some important concepts that are used throughout in this thesis. We first discuss the learning problem, in particular supervised learning and its properties. After that, we will review Support Vector Machines in which the formulations of SVMs for binary and multi-class classification problems are presented. Finally, we discuss some exiting online learning algorithms for SVMs.

2.1 The Learning problem

2.1.1 Supervised learning

Supervised learning algorithms infer a relation between an input space $\mathcal{X} \subseteq \mathbb{R}^n$ and an output space $\mathcal{Y}$ based on sample data. In detail, let such sample data $S = \{(x_1, y_1), (x_2, y_2), .., (x_\ell, y_\ell)\}$ be drawn according to an unknown but fixed probability distribution p over $\mathcal{X} \times \mathcal{Y}$. This sample dataset is normally referred to as training set. Two important examples of supervised learning are the classification problem, where $\mathcal{Y}$ is a finite set of classes $\{C_1, C_2, .., C_d\}, 2 \leq d < \infty$, or a regression problem, where $\mathcal{Y} \subseteq \mathbb{R}^d, 1 \leq d < \infty$. However, this thesis will only focus on classification problems, thus $\mathcal{Y}$ is restricted to $\{C_1, C_2, .., C_d\}$.
The goal of supervised learning is to determine a hypothesis $h : \mathcal{X} \rightarrow \mathcal{Y}$ that takes an input $x \in \mathcal{X}$ and returns the corresponding output $y \in \mathcal{Y}$. This hypothesis is selected from a hypothesis class $\mathcal{H}$, for example a hypothesis in the set of all linear classifiers $\mathcal{H} = \{sign(w^T x + b), w \in \mathcal{X}, b \in \mathbb{R}\}$. A hypothesis $h \in \mathcal{H}$ is sometimes represented as $h(x) = sign(f(x))$, $f : \mathcal{X} \rightarrow \mathbb{R}$ in binary classification or $h(x) = \arg\max_{c \in \mathcal{Y}}(f_c(x))$, $f_c : \mathcal{X} \rightarrow \mathbb{R}$, $f = [f_{C_1}, .., f_{C_d}]$ in multi-class classification. The function f is called scoring function, and let $\mathcal{F}$ denote a set of scoring functions. Because of the close interconnection between f and g, in the following, h can be used as scoring function except for explicitly distinguishing cases.
The quality of a prediction of h is measured by a task-dependent loss function L, which fulfils $L : \mathcal{Y} \times \mathcal{Y} \rightarrow [0, \infty]$, $L(y, y) = 0$. Therefore, $L(\overline{y}, y)$ represents the cost of predicting $\overline{y}$ instead of y. Some common loss functions on a hypothesis are:

- 0-1 loss: $L(y, h(x)) = 0$ if $y = h(x)$, 1 otherwise. This is the ideal loss function.

- Square loss: $L(y, h(x)) = (y - h(x))^2$. This loss function is normally used in regression
- Hinge loss: $L(y, h(x)) = \max(1 - yh(x), 0)$. This is common loss in SVMs.
- Exponential loss: $L(y, h(x)) = \exp(-yh(x))$
- Margin based loss: $L(y, h(x)) = L(yh(x))$. This is a generalized version of the hinge loss.

Given such a loss function, the expected risk of a hypothesis h over p can be defined as:

$$\mathcal{R}_p(h) = \int L(y, h(x))\mathrm{d}p(x, y)$$

The goal of learning is to find a hypothesis that minimizes the expected risk $\mathcal{R}_p(h)$.
In general, the underlying probability distribution p is unknown, thus the empirical risk $\mathcal{R}_S(h)$ of h over sample data S is used as a substitute:

$$\mathcal{R}_S(h) = \frac{1}{\ell}\sum_{i=1}^{\ell} L(y_i, h(x_i))$$

As a result, the empirical risk minimization is normally used.
Let us define some important concepts that deal with in the thesis. The best hypothesis over all possible hypotheses is called Bayes classifier and its associated risk is called Bayes risk:

$$\mathcal{R}_p^{Bayes} = \inf \mathcal{R}_p(h)$$
$$\mathcal{R}_p(h_{Bayes}) = \mathcal{R}_p^{Bayes}$$

For the best hypothesis within a hypothesis set $\mathcal{H}$, we write:

$$h_{\mathcal{H}} = \arg\min_{h\in\mathcal{H}} \mathcal{R}_p(h)$$

2.1.2 Generalization and regularization

One of major problems with supervised learning which relates to empirical risk minimization, is overfitting. Overfitting arises when learning algorithms typically choose non smooth or too complex function that minimizes empirical risk $\mathcal{R}_S(h)$. Overfitting leads to a problem in which a hypothesis is selected to fit the training dataset very well, but it does not generalize well to unseen test data from the same generating distribution. Generalization can be considered as the ability of a hypothesis that has low mistakes with future test data. There are several ways to handle the overfitting problem. This thesis mainly focuses on the regularization framework. This is the basis of several important learning algorithms, for instance Support Vector Machines, which will be at the center of this thesis. The regularization framework proposes to select a hypothesis from a normed function space $\mathcal{H}$ given a training set S as :

$$h^* = \min_{h\in\mathcal{H}}(\mathcal{R}_S(h) + \lambda g(||h||_{\mathcal{H}})) \tag{2.1}$$

Here, $g : \mathbb{R} \to \mathbb{R}$ is a monotonically increasing regularization function which penalizes the complexity of h as measured by $||h||_{\mathcal{H}}$. The parameter $\lambda > 0$ is fixed and controls a trade-off between the empirical risk and the smoothness of the hypothesis. In practice, $g(||h||_{\mathcal{H}}) = ||h||^2_{\mathcal{H}}$ is normally employed .

2.1.3 Consistency

Consistency concerns the behaviour of classifiers created by a learning algorithm as the number of training examples goes to infinity. While generalization is a property of one classifier, consistency is a property of family of classifiers.

Definition 1 ((Definition 1 in [33])). *Consider a sample dataset $S = \{(x_1, y_1), (x_2, y_2), .., (x_\ell, y_\ell)\}$ of size ℓ drawn according to an underlying probability distribution p over $\mathcal{X} \times \mathcal{Y}$. Let $\mathcal{H}$ be the hypothesis set from which the learning algorithm chooses its hypothesis. Then:*

- *a learning algorithm is called consistent with respect to $\mathcal{H}$ and p if the risk $\mathcal{R}_p(h)$ converges in probability to the risk $\mathcal{R}_p(h_\mathcal{H})$, where h is generated by this learning algorithm. This is, for all $\varepsilon > 0$:*

$$\Pr(\mathcal{R}_p(h) - \mathcal{R}_p(h_\mathcal{H}) > \varepsilon) \to 0, \ \textit{as } \ell \to \infty$$

- *a learning algorithm is called Bayes-consistent with respect to p if the risk $\mathcal{R}_p(h)$ converges in probability to the Bayes risk $\mathcal{R}_p^{Bayes}$. That is , for all $\varepsilon > 0$:*

$$\Pr(\mathcal{R}_p(h) - \mathcal{R}_p(h_{Bayes}) > \varepsilon) \to 0, \ \textit{as } \ell \to \infty$$

- *a learning algorithm is called universally consistent with respect to $\mathcal{H}$ if it is consistent with respect to $\mathcal{H}$ for all underlying probability distribution p.*

2.1.4 Fisher consistency

Another important concept in machine learning is Fisher consistency, in which the behaviour of loss functions in the limit of infinite data is analysed. Fisher consistency of a loss function is a necessary condition for a classification algorithm based on that loss to be Bayes consistent.
Given a particular loss function $L(y, h(x))$, let $h^*_L : \mathcal{X} \to \mathcal{Y}$ be the minimizer of the expected risk:

$$h^*_L = \arg\min_h \mathcal{R}_p(h) = \arg\min_h \int L(y, h(x)) \mathrm{d}p(x, y)$$

In the case of binary classification, the Bayes optimal rule is $sign(2p(y = 1|x) - 1)$. A loss function $L(y, h(x))$ is called Fisher consistent if $h^*_L(x)$ has the same sign with $sign(2p(y = 1|x) - 1)$.
This definition for binary classification can be extended to the multi-category case, where $h^*_L(x)$ is represented as $h^*_L(x) = \arg\max_{c \in \mathcal{Y}}(f^*_c(x))$. A loss function $L(y, h(x))$ is called Fisher consistent if

$$\arg\max_{c \in \mathcal{Y}}\{f^*_c\} \subset \arg\max_{y \in \mathcal{Y}}\{p(y|x)\}$$

In [23], Lin proved that margin-based loss functions listed above with some additional constraints are Fisher consistent for binary classification problems. Independently, Zhang [36, 35] proved that a larger class of convex loss functions are Fisher consistent in binary classification tasks. Bartlett et al. [2] and Tewari et al. [31] extended Zhang's results to establish conditions for a loss function to satisfy Fisher consistency for binary and multi-category classification, respectively.

2.1.5 Online learning and batch learning

There exists a set of two common paradigms of learning algorithms for classification problems, namely online learning and batch learning. Batch learning is a paradigm underlying many existing supervised learning algorithms, which assumes that all training examples are available at once. Normally, a cost function or objective function is defined to estimate how well a hypothesis behaves on these examples. The learning algorithm will then perform optimization steps towards reducing the cost function until some stopping conditions are reached. Because common batch learning algorithms scale between quadratically and cubically in the number of examples, which can result in quite long training times on large datasets. In contrast, online learning refers to a learning algorithm iteratively adjusting the parameters associated with incoming new and fresh examples when they come as a stream [7]. Inspired by this "real" online learning paradigm, there are optimization methods that replicate the strategy from online learning to approximate the solution of a fixed batch problem. When using the online learning paradigm as a heuristic for optimization, there is some freedom in how many computation steps on 'old' examples are done after the coming of each "new" example. If these steps are designed effectively, then a single pass over the dataset of online learning will take much shorter time and less computation than a full batch optimization run [9, 7, 5], but still provides a well-performing approximation to the solution that would be obtained by a full batch solver. During this thesis, such an online optimization algorithm using online learning is derived to achieve comparable accuracy with batch learning algorithm, but it consumes significant less time and computation.

2.2 Support Vector Machines

In this section, we will review some important concepts in Support Vector Machines. We will discuss some concepts in linear classification as well as kernel concepts before going to describing the formulation and the training of binary Support Vector Machines. We also review the three all-in-one approaches and the unified framework for multi-class SVMs. The final part of this section will present some online SVMs algorithms.

2.2.1 Linear classification

This part will review some basic concepts in linear classification. These concepts are important since they will appear later when deriving binary Support Vector Machines algorithm as well as in other sections. In linear classification, the hypothesis h for classification is defined on a affine linear function f on the input space as:

$$h(x) = sign(f(x))$$
$$f(x) = w^T x + b$$

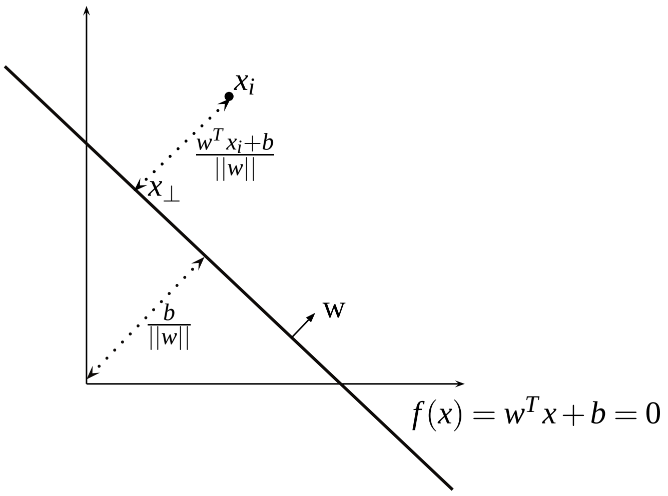
Here, $w \in \mathbb{R}^n$ is the weight vector and b is the bias. These parameters define a separating hyperplane that divides the input space into two half-spaces corresponding to the two classes.
Let us next introduce definitions around the concept of a margin between a separating hyperplane

and one or more data instances. The functional margin of a data instance (x_i, y_i) with respect to the hyperplane (w, b) is defined as:

$$\gamma_i = y_i(w^T x_i + b)$$

The geometric margin of an example (x_i, y_i) with respect to the hyperplane (w, b) is:

$$\rho_i = \frac{(y_i(w^T x_i + b))}{||w||} = \frac{\gamma_i}{||w||}$$



The absolute value of the geometric margin of an input pattern equals its distance from the separating hyperplane defined by (w, b) . Indeed, let d be the distance from the hyperplane and $x_\perp$ is the projection onto the hyperplane of an input pattern x_i. Therefore:

$$\begin{aligned} x_i &= x_\perp + \frac{w}{||w||} d \\ \Rightarrow \quad f(x_i) &= w^T (x_\perp + \frac{w}{||w||} d) + b \\ &= w^T x_\perp + b + ||w|| d \\ &= f(x_\perp) + ||w|| d \end{aligned}$$

However, $x_\perp$ lies on the hyperplane, thus $f(x_\perp) = 0$. As a result:

$$d = \frac{f(x_i)}{||w||} = \frac{w^T x_i + b}{||w||}$$

The information about the distance of an input pattern x_i from a hyperplane is thus encoded in the geometric margin. Furthermore, it can be seen that a hypothesis h classifies correctly an example if the functional and geometric margins corresponding to this hypothesis are positive. A dataset S is called a linearly separable dataset if there exist w, b that satisfy the following constrains:

$$\begin{gathered} w^T x_i + b > 0 \; if \; y_i = +1 \\ w^T x_i + b < 0 \; if \; y_i = -1 \\ Or \;\; y_i(w^T x_i + b) \geq \gamma, \;\; \gamma > 0 \; and \; 1 \leq i \leq \ell \end{gathered}$$

Here, γ is called a target functional margin.
The margin concepts of a hyperplane for a single example (x_i, y_i) can be extended to the training dataset S. The functional margin and geometric margin of a hyperplane with regards to a training set S can be defined as $\gamma = \min_{1 \leq i \leq \ell} \gamma_i$ and $\rho = \min_{1 \leq i \leq \ell} \rho_i$, respectively.

2.2.2 Kernels

Kernels are a flexible and powerful tool which allows to easily modify linear methods such that they yield non-linear decision functions in the input space. The underlying purpose is to transform a non-linearly separable dataset into a Hilbert space with higher dimension or even infinite dimension, where the dataset can be separated by a hyperplane. The interesting point of view is that this transformation is performed directly through kernel evaluations, without explicit computation of any feature map.

Definition 2. *Let $\mathcal{X}$ be a non-empty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if there exists a Hilbert space $\mathcal{H}$ and a map $\phi : \mathcal{X} \to \mathcal{H}$ satisfying:*

$$k(x, y) = \langle \phi(x), and \phi(y) \rangle_{\mathcal{H}} \; \forall x, y \in \mathcal{X}$$

Here, $\phi : \mathcal{X} \to \mathcal{H}$ is called feature map and $\mathcal{H}$ is called feature space, ϕ and $\mathcal{H}$ might be not unique. As a trivial example considers $k(x, y) = x * y$, $\phi(x) = x$ with $\mathcal{H} = \mathbb{R}$ or $\phi(x) = [\frac{x}{\sqrt{2}}, \frac{x}{\sqrt{2}}]^T$ with $\mathcal{H} = \mathbb{R}^2$.

A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called positive definite if for all $m \in \mathbb{N}$ and $x_1, x_2, .., x_m \in \mathcal{X}$, the Gram matrix K with entries $k_{ij} = k(x_i, x_j)$ is positive definite. That is, for all $a \in \mathbb{R}^m$, $a^T K a \geq 0$. It can be seen that a kernel is also a positive definite function because:

$$\sum_{i,j=1}^{m} a_i a_j k(x_i, x_j) = \Big\langle \sum_{i=1}^{m} a_i \phi(x_i), \sum_{j=1}^{m} a_j \phi(x_j) \Big\rangle_{\mathcal{H}} = \Big\| \sum_{i=1}^{m} a_i \phi(x_i) \Big\|_{\mathcal{H}}^2$$

Vice versa, Mercer's theorem [25] guarantees that for every positive definite function k, there exists a Hilbert space $\mathcal{H}$ and a feature map $\phi : \mathcal{X} \to \mathcal{H}$ such that the kernel is the inner product of features in the feature space: $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$.

Other important concepts are that of reproducing kernels and reproducing Hilbert space that are at the theoretical core of the kernelization technique. They can be defined as follows:

Definition 3. *Let $\mathcal{X}$ be a non-empty set and $\mathcal{H}$ be a Hilbert space of functions on $\mathcal{X}$. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a reproducing kernel and $\mathcal{H}$ is a reproducing kernel Hilbert space (RKHS) if they satisfy:*

- $\forall x \in \mathcal{X} : k(\cdot,x) \in \mathcal{H}$
- $\forall x \in \mathcal{X}, \forall f \in \mathcal{H} : \langle f, k(\cdot, x)\rangle_{\mathcal{H}} = f(x)$

In particular $\forall x, y \in \mathcal{X} : k(x, y) = \langle k(\cdot, x), k(\cdot, y)\rangle_{\mathcal{H}} = \langle k(\cdot, y), k(\cdot, x)\rangle_{\mathcal{H}}$. It is clear that a reproducing kernel is a kernel since a feature map can be defined: $\phi(x) = k(\cdot, x)$. The Moore-Aronszajn theorem [1] states that for every positive definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ over a non-empty set $\mathcal{X}$, there is a unique RKHS $\mathcal{H}$ with reproducing kernel k. As as result, there are strong connections between the concepts of positive definite function, kernel and reproducing kernel.

The natural question that has been raised is whether kernel-based learning algorithm can be consistent given a certain kernel. In other words, given a kernel, can the classifier resulting from a kernel-based learning algorithm lead to Bayes optimal classifier as the sampling size goes to infinity. The answer is that this depends on a property of the kernel function, the so-called universal property . The RKHS $\mathcal{H}$ induced by a universal kernel is rich enough to approximate any optimal classifier or function arbitrarily well [30]. For example, the Gaussian kernel is universal kernel whereas linear kernel is not.

2.2.3 Regularized risk minimization in RKHS

We now combine the framework of kernels and RKHS function spaces of the last section with the regularization framework presented in Section 2.1.2. An interesting result within this combination is established by (Kimeldorf and Wahba, 1971) in [21] and (Schölkopf, Herbrich and Smola, 2001) in [28], which is the representer theorem. The representer theorem says that if $\mathcal{H}$ is a RKHS induced by a reproducing kernel k and g is a monotonically increasing regularization function, then the optimal solution f of regularization framework (2.1) admits a representation form:

$$f(\cdot) = \sum_{i=1}^{\ell} a_i k(\cdot, x_i), \quad x_i \in S$$

As a result, the optimal solution is expressed as a linear combination of a finite number of kernel functions on the dataset, regardless of the dimensionality of $\mathcal{H}$. Therefore, the representer theorem enables us to perform empirical risk minimization in infinite dimensional feature spaces. Since, by adding the regularization term, the problem reduces to finite number of variables. An example of algorithms performing regularized risk minimization in RKHS are Support Vector Machines to which the representer theorem thus naturally applies as well [15], and which we introduce SVMs in the next section. However, for ease of understanding, we derive the non-kernelized SVM first.

2.2.4 Binary Support Vector Machines

Support Vector Machines (SVM) are state-of-the-art algorithms in machine learning for pattern recognition due to their theoretical properties and high accuracies in practice. In this part, Support Vector Machines are first formulated in linear classification through the large margin concept, then the formulation will be extended to non-linear classification through the kernel trick.
With a linearly separable dataset S, there might be numerous hyperplanes that can separate the data set into two classes. The fundamental idea of large-margin classification, which is also inherently related to SVM classification, is to choose that optimal hyperplane which maximizes the geometric margin with respect to this dataset. In detail, a hyperplane is selected such that the geometric margin of the closest data points of both classes is maximized. These data points which lie on the solid line in Figure 2.1, are called support vectors. The problem of large-margin

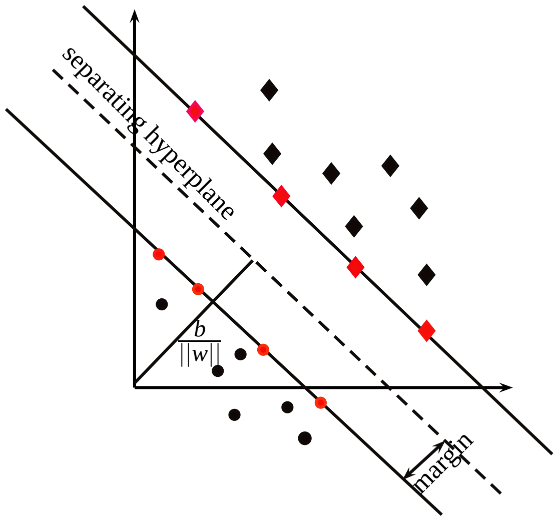


Figure 2.1: The geometry interpretation of Binary Support Vector Machines

classification can be formulated as the following optimization problem over w and b :

$$\text{maximize}_{w,b}\ \rho = \frac{\gamma}{||w||}$$
$$\text{Subject to } y_i(w^T x_i + b) \geq \gamma\ ,\ \ i = 1, .., \ell$$

Because (w, b) can be rescaled to (cw, cb), which will result in the same hyperplane and optimization problem, γ can be fixed to 1 (otherwise, we have to fix $||w|| = 1$). Therefore, $||w||$ should be minimized, which is equivalent to the convex optimization problem:

$$\text{minimize}_{w,b}\ \ \frac{1}{2}||w||^2$$
$$\text{Subject to }\ y_i(w^T x_i + b) \geq 1\ \forall (x_i, y_i) \in S$$

When the Bayes risk of this dataset itself is not zero, which means that the dataset is no longer linearly separable. Therefore, it becomes necessary allow some examples to violate the margin condition. This idea can be integrated as $y_i(w^T x_i + b)) \geq 1 - \xi_i$, with slack variables $\xi_i \geq 0$. As a result, the convex optimization problem becomes:

$$\begin{aligned} \text{minimize}_{w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{l} \xi_i \qquad (2.2) \\ \text{Subject to} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i \ \forall (x_i, y_i) \in S \\ & \xi_i \geq 0, 1 \leq i \leq \ell \end{aligned}$$

Here, $C > 0$ is the regularization parameter. It is obvious that the formulation (2.2) fits well into the regularization framework (2.1) where :

$$\begin{aligned} g(||f||) &= ||w||^2 \\ \lambda &= \frac{1}{2C\ell} \\ \mathcal{R}_S(f) &= \frac{1}{\ell}\sum_{i=1}^{\ell} \max(1 - y_i(w^T x_i + b), 0) \end{aligned}$$

That is the reason why C is called regularization parameter.
In practice, the convex optimization problem (2.2) can be converted into its so-called dual form by the technique of Lagrange multipliers for constrained convex optimization problems [11], this results in the equation:

$$\begin{aligned} \text{maximize}_{\alpha} \quad & D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \qquad (2.3) \\ \text{Subject to} \quad & \sum_{i=1}^{l} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \forall i \in \{1, .., \ell\} \end{aligned}$$

Here, α_i is a Lagrange multiplier.
Another extension of SVM for handling datasets which are not linearly separable is to apply kernel trick which is introduced in section 2.2.2. This will map the original input space into a higher dimensional feature space that can be linear separable. Kernelization is performed by replacing every inner product $\langle x_i, x_j \rangle$ in (2.3) by a kernel evaluation $k(x_i, x_j)$. As a result, the final dual formulation can be represented as follows:

$$\text{maximize}_{\alpha} \quad D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \qquad (2.4)$$

$$\text{Subject to} \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \qquad (2.5)$$

$$0 \leq \alpha_i \leq C, \forall i \in \{1, .., \ell\} \qquad (2.6)$$

2.2.5 Support Vector Machine training

Sequential minimal optimization

We will here consider iterative optimization algorithms which can be viewed as in each step, finding a new update direction u as well as update step size λ such that $\max_{\lambda,u} D(\alpha+\lambda u)$, where $\alpha+\lambda u$ is a feasible vector. One of the most effective algorithms for the above problem is Sequential Minimal Optimization (SMO) [26] in which the search direction vector has as few non-zero entries as possible and the set of the corresponding indices is called working set. Because of the equality constraint (2.5), the smallest number of variables that can be changed at a time is 2. As a result, the search direction can be represented as:

$$u = \pm(0, .., y_i, 0, .., -y_j, 0.., 0)$$

The sign $\pm$ is chosen to satisfy the equality constraint (2.5) due to the value of labels y_i, y_j. Without loss of generalization, assume that:

$$u_{ij} = (0, .., y_i, 0, .., -y_j, 0.., 0)$$

To make the following mathematical equations easier to read, we define the following notation::

$$\begin{aligned}
k_{ij} &= k(x_i, x_j)\\
g_i = \frac{\partial D}{\partial \alpha_i} &= 1 - y_i \sum_{j=1}^{l} y_j \alpha_j\\
[a_i, b_i] &= \begin{cases} [0, C] & if\ y_i = +1 \\ [-C, 0] & if\ y_i = -1 \end{cases}\\
I_{up} &= \{i \mid y_i\alpha_i < b_i\}\\
I_{down} &= \{j \mid y_j\alpha_j > a_j\}
\end{aligned}$$

Taking Taylor expansion for D around α in the search direction u_{ij}, we have:

$$D(\alpha + \lambda u_{ij}) = D(\alpha) + \lambda \frac{\partial D(\alpha)}{\partial \alpha} u_{ij}^T + \frac{1}{2}\lambda^2 u_{ij}^T \frac{\partial^2 D}{\partial^2 \alpha} u_{ij}$$

In addition, the second order derivatives of D with respect to α_i and α_j can be derived as:

$$\frac{\partial D}{\partial \alpha_i \partial \alpha_j} = -y_i y_j k_{ij}$$

And, the first order derivative of D with respect to α is:

$$\frac{\partial D(\alpha)}{\partial \alpha} u_{ij}^T = y_i g_i - y_j g_j$$

As a result, the function D can be written:

$$D(\alpha + \lambda u_{ij}) = D(\alpha) + \lambda(y_i g_i - y_j g_j) - \frac{1}{2}(k_{ii} + k_{jj} - 2k_{ij})\lambda^2 \tag{2.7}$$

Solving the one dimensional optimization problem (2.7) without the box constraint (2.6), the optimal value is:

$$\lambda^* = \frac{y_i g_i - y_j g_j}{k_{ii} + k_{jj} - 2k_{ij}} \tag{2.8}$$

If only $\lambda \geq 0$ is considered, from first order Taylor expansion:

$$D(\alpha + \lambda u_{ij}) = D(\alpha) + \lambda(y_i g_i - y_j g_j) + O(\lambda^2)$$

Therefore, $D(\alpha)$ reaches the optimality if there does not exist (i, j) such that $y_i g_i - y_j g_j \geq 0$. As a result, the optimality condition is:

$$\max_{i \in I_{up}} y_i g_i - \min_{j \in I_{down}} y_j g_j \leq 0$$

In reality, the condition is replaced by:

$$\max_{i \in I_{up}} y_i g_i - \min_{j \in I_{down}} y_j g_j \leq \varepsilon \text{ , for some } \varepsilon > 0 \tag{2.9}$$

From (2.8) and (2.9), the sequential minimal optimization is derived as in Algorithm 1

Algorithm 1 Sequential Minimal Optimization

```
1: $\alpha \leftarrow 0, g \leftarrow 1$
2: repeat
3:     select indices $i \in I_{up},\ j \in I_{down}$
4:     $\lambda = \min\{b_i - y_i\alpha_i, y_j\alpha_j - a_i, \frac{y_i g_i - y_j g_j}{k_{ii} + k_{jj} - 2k_{ij}}\}$
5:     $\forall p \in \{1, .., \ell\} : g_p = g_p - \lambda y_p k_{ip} + \lambda y_p k_{jp}$
6:     $\alpha_i = \alpha_i + \lambda y_i$
7:     $\alpha_j = \alpha_j - \lambda y_j$
8: until $\max_{i \in I_{up}} y_i g_i - \min_{j \in I_{down}} y_j g_j \leq \varepsilon$
```

The operations from step (4) to step (7) in Algorithm 1 form a SMO optimization step which is frequently used later.

Working set selection

The performance of SMO heavily depends on how the working set is selected. There are two common strategies for working set selection.

- Most violating pair strategy which is based on the gradient information.

$$i = \arg\max_{p \in I_{up}} y_p g_p$$

$$j = \arg\min_{q \in I_{down}} y_q g_q$$

- Another strategy [17, 16] employs second order information to select a working set that maximizes the gain. The gain is determined as follows. Rewriting equation (2.7), we have:

$$D(\alpha + \lambda u_{ij}) - D(\alpha) = \lambda(y_i g_i - y_j g_j) - \frac{1}{2}(k_{ii} + k_{jj} - 2k_{ij})\lambda^2$$

  Substituting the optimal value λ^* into the equation yields:

$$D(\alpha + \lambda u_{ij}) - D(\alpha) = \frac{(y_i g_i - y_j g_j)^2}{2(k_{ii} + k_{jj} - 2k_{ij})}$$

  The idea is selecting i and j such that the gain $\frac{(y_i g_i - y_j g_j)^2}{2(k_{ii} + k_{jj} - 2k_{ij})}$ is maximized. However, $\frac{\ell(\ell-1)}{2}$ pairs need to be checked, which results in long computation times. Therefore, an idea which only needs $O(\ell)$ computation steps:

  - The index i is picked up, which is based on most violating pair strategy $i = \arg\max_{p \in I_{up}} y_p g_p$.
  - the index j is selected to maximize the gain , $j = \arg\max_{q \in I_{down}} \frac{(y_i g_i - y_q g_q)^2}{2(k_{ii} + k_{qq} - 2k_{iq})}$

2.2.6 Multi-class SVMs

In practice, pattern recognition problems often involve multiple classes. Support Vector Machines can be extended to handle such cases by training a series of binary SVMs or by solving a single optimization problem. There are two common ways in training a series of binary SVMS, namely the one-versus-one (OVO) method and the one-versus-all (OVA) method. However, they are quite similar, thus we restrict ourself to only discuss about the OVA method. The OVA method is to combine separately trained binary SVM classifiers into a classifier for multiple classes. This method has the advantage of being simple to implement on top of an existing binary SVM solver. However, it has a disadvantage that even if all binary classifiers are consistent, the resulting OVA classifier is inconsistent [13]. Formulating a single optimization problem to train multi-class SVMS is called all-in-one approaches. These approaches are more theoretical involved and and have been shown to be able to give better accuracies than OVA on a benchmark set of datasets [13]. However, their time complexities which depend on the dataset size, can be significant higher than those of OVA.
The remainder of this section will present some all-in-one SVMs in detail. They all share the

common property (together with the OVA machine) that the final classification decision is made according to a rule of the form:

$$x \longrightarrow \arg\max_{c\in\{1,..,d\}} (w_c^T \phi(x) + b_c)$$

Just like in section 2.2.2, here $\phi(x) = k(x, \cdot)$ is a feature map into a reproducing kernel Hilbert space $\mathcal{H}$ with kernel k, d is the number of class, w_c, b_c are parameters that define the decision function of class c. There are several ways to approach multi-class SVM classification via one single optimization problem (all-in-one approach):

- Weston and Watkins (WW) SVM [34]:

$$\begin{aligned} &\text{minimize}_{w_c} \ \frac{1}{2}\sum_{c=1}^{d}\langle w_c, w_c\rangle + C\sum_{i=1}^{l}\sum_{c=1}^{d}\xi_{i,c} \\ &\forall n \in \{1,..,l\}, \forall c \in \{1,..,d\}\backslash\{y_n\} : \langle w_{y_n} - w_c, \phi(x_n)\rangle + b_{y_n} - b_c \geq 2 - \xi_{n,c} \\ &\forall n \in \{1,..,l\}, \forall c \in \{1,..,d\} : \xi_{n,c} \geq 0 \end{aligned}$$

- Crammer and Singer (CS) SVM[10]:

$$\begin{aligned} &\text{minimize}_{w_c} \ \frac{1}{2}\sum_{c=1}^{d}\langle w_c, w_c\rangle + C\sum_{i=1}^{\ell}\xi_i \\ &\forall n \in \{1,..,l\}, \forall c \in \{1,..,d\}\backslash\{y_n\} : \langle w_{y_n} - w_c, \phi(x_n)\rangle \geq 1 - \xi_n \\ &\forall n \in \{1,..,l\} : \xi_n \geq 0 \end{aligned}$$

- Lee, Lin and Wahba(LLW) SVM [22]:

$$\begin{aligned} &\min_{w_c} \ \frac{1}{2}\sum_{c=1}^{d}\langle w_c, w_c\rangle + C\sum_{n=1}^{\ell}\sum_{c=1}^{d}\xi_{n,c} \\ \text{Subject to } &\forall n \in \{1,..,\ell\}, c \in \{1,..,d\} \backslash\{y_n\} : \langle w_c, \phi(x_n)\rangle + b_c \leq -\frac{1}{d-1} + \xi_{n,c} \\ &\forall n \in \{1,..,l\}, c \in \{1,..,d\} : \xi_{n,c} \geq 0 \\ &\forall h \in \mathcal{H} : \sum_{c=1}^{d}(\langle w_c, h\rangle + b_c) = 0 \end{aligned} \tag{2.10}$$

All these extensions fit into a general regularization framework:

$$\frac{1}{2}\sum_{c=1}^{d}\langle w_c, w_c\rangle + C\sum_{i=1}^{\ell} L(y_i, f(x_i))$$

Where:

- LLW-SVM loss function: $L(y, f(x)) = \sum_{j \neq y}[1 + f_j(x)]_+$
- WW-SVM loss function: $L(y, f(x)) = \sum_{j \neq y}[1 - (f_y(x) - f_j(x)]_+$
- CS-SVM loss function: $L(y, f(x)) = \sum_{j \neq y}[1 - \min_j(f_y(x) - f_j(x))]_+$

Here, $f = [f_1, f_2, .., f_d]$ is a vector of hypotheses $f_c = w_c^T \phi(x) + b_c$, and $[u]_+ = \max(u, 0)$. Among these loss functions, Liu in [24] proved that only the LLW-SVM loss function is Fisher consistent. Therefore, it is highly desirable to derive online learning algorithms for LLW-SVMs.

2.2.7 The unified framework for multi-class SVMs

The margin concept is an important concept in binary classification, which is inherited by binary SVM in order to select the maximum margin classifier. However, there are different ways to interpret this concept for multi-class classification. In addition, there are several strategies to construct a loss function based on margin concept. We will review these different concepts before re-stating the unified formulation.

Margin in multi-class SVM classification

As discussed in the Section 2.2.1, the (functional or geometric) margins encodes the information of whether a classification is correct or not by a linear classifier $f(x) = w^T x + b$. In binary classification, $\mathcal{Y} = \{+1, -1\}$, $y(w^T x + b) \geq \gamma$ for a specific target functional margin $\gamma > 0$, which corresponds to the correct classification of f for an example (x, y). When the Bayes risk is not zero, it is necessary to allow margin violations and we measure this margin violation by $\max(0, \gamma - y(w^T x + b))$ like in section 2.2.4.
In multi-class case, the hypothesis or the decision function is normally obtained as $\arg\max_{c \in \mathcal{Y}}(f_c(x))$, thus, there are several ways to interpret the margin concepts. With two scoring functions f_c and f_e corresponding to two classes c, e, the difference $f_c(x) - f_e(x)$ encodes the information that the class c is preferred over class e in the decision function. This observation is developed in CS and WW formulations. Therefore, a margin can be measured as follows:

Definition 4 (relative margin). *For a labelled point $(x, y) \in \mathcal{X} \times \mathcal{Y}$ the values of the margin function*

$$\mu_c^{rel}(f(x), y) = \frac{1}{2}(f_y(x) - f_c(x))$$

for all $c \in \mathcal{Y}$ are called relative margins.

The following, different margin definition is the basis of the LLW SVM:

Definition 5 (absolute margin). *For a labelled point $(x, y) \in \mathcal{X} \times \mathcal{Y}$ the values of the margin function*

$$\mu_c^{abs}(f(x), y) = \begin{cases} +f_c(x) & \text{if } c = y \\ -f_c(x) & \text{if } c \in \mathcal{Y} \setminus \{y\} \end{cases}$$

for all $c \in \mathcal{Y}$ are called absolute margins.

With this definition, the decision function just selects the largest score value $f_c(x)$ and it does not directly take in account the differences $f_y(x) - f_c(x)$.

Margin-based surrogate loss functions

In binary classification, the hinge loss is applied to the margin violation, which itself is obtained from the functional margin. Similarly, margin-based loss functions in multi-class SVMs are also created through the margin violation concept. Given a margin function $\mu_c(f(x), y)$ and a target margin $\gamma_{y,c}$ that possibly depends on both class y and c, the target margin violation is defined as:

$$v_c(f(x), y) = \max(0, \gamma_{y,c} - \mu_c(f(x), y))$$

However, in multi-category classification with d classes, there are $(d - 1)$ possibilities to make a mistake when classifying an input pattern. Therefore, there are several ways to combine the d different function scores into one single loss value.

Definition 6 (sum-loss)**.** *For a labelled point $(x, y) \in \mathcal{X} \times \mathcal{Y}$ the discriminative sum-loss is given by*

$$L^{sum}(f(x), y) = \sum_{c \in \mathcal{Y}'} v_c(f(x), y)$$

for $\mathcal{Y}' = \mathcal{Y} \backslash \{y\}$. Setting $\mathcal{Y}' = \mathcal{Y}$ results in the total sum-loss.

The LLW and WW machines use the discriminative sum-loss in their formulations.

Definition 7 (max-loss)**.** *For a labelled point $(x, y) \in \mathcal{X} \times \mathcal{Y}$ the discriminative max-loss is given by*

$$L^{sum}(f(x), y) = \max_{c \in \mathcal{Y}'} v_c(f(x), y)$$

for $\mathcal{Y}' = \mathcal{Y} \backslash \{y\}$. Setting $\mathcal{Y}' = \mathcal{Y}$ results in the total max-loss.

The CS machine is based on the discriminative max-loss.

Unifying the margins and loss functions

Firstly, it is straightforward that the different concepts in the earlier section can be unified as:

$$\mu_c(f(x), y) = \sum_{m=1}^{d} \nu_{y,c,m} f_m(x)$$

Here, $\nu_{y,c,m}$ is a coefficient to the score function f_m, which also depends on the combination y, c. As can be seen, the relative margin and absolute margin can be brought into the unified form by defining the coefficients $\nu_{y,c,m}$ as in the Table 2.1
Secondly, the different margin-based loss functions also need to be unified. It can be seen that all

margin type	$\nu_{y,c,m}$
relative	$\delta_{y,m} - \delta_{c,m}$
absolute	$-(-1)^{\delta_{c,y}}\delta_{m,c}$

Table 2.1: Coefficients $\nu_{y,c,m}$ for the different margins in the unified form

the loss functions in section 2.2.7 are linear combinations of target margin violations. Each target margin violation can be represented through a slack variable like in section 2.2.4. Therefore, for an example $(x, y) \in \mathcal{X} \times \mathcal{Y}$, the unified loss function can be defined as follows.

$$L(f(x), y) = \min_{\xi} \sum_{r \in R_y} \xi_r \tag{2.11}$$
$$\text{Subject to} \quad \xi_{s_y(p)} \geq \nu_p(f(x), y) \quad \forall p \in P_y$$
$$\xi_r \geq 0 \quad \forall r \in R_y$$

Here, $P_y \in \mathcal{Y}$ lists all the target margin violations that enter the loss, R_y is the index set that lists all slack variables, $s_y : P_y \to R_y$ is a surjective function that assigns slack variables to margin components. Table 2.2 shows some examples of loss functions and their corresponding representation in the unified notation of equation 2.11.

loss type	$L(f(x), y)$	P_y	R_y	s_y
discriminative max-loss	$\max_{c \in \mathcal{Y}\setminus\{y\}} \nu_c(f(x), y)$	$\mathcal{Y}\setminus y$	$*$	$p \mapsto *$
total max-loss	$\max_{c \in \mathcal{Y}} \nu_c(f(x), y)$	$\mathcal{Y}$	$*$	$p \mapsto *$
discriminative sum-loss	$\sum_{c \in \mathcal{Y}\setminus\{y\}} \nu_c(f(x), y)$	$\mathcal{Y}\setminus\{y\}$	$\mathcal{Y}\setminus\{y\}$	$p \mapsto p$
total sum-loss	$\sum_{c \in \mathcal{Y}} \nu_c(f(x), y)$	$\mathcal{Y}$	$\mathcal{Y}$	$p \mapsto p$

Table 2.2: Margin-based loss functions and their corresponding parameters in the unified form. Here, * means a constant index.

Unifying the primal problems

All the multi-class SVM machines in the section 2.2.6 fit into a regularization framework. Therefore, integrating the unified loss function above into the regularization framework, we have a unified primal problem:

$$\min_{w,b,\xi} \quad \frac{1}{2}\sum_{c=1}^{d} \|w_c\| + C\sum_{i=1}^{\ell}\sum_{r\in R_{y_i}} \xi_{i,r} \tag{2.12}$$

$$\begin{aligned}
\text{Subject to} \quad & \sum_{c=1}^{d} \nu_{y_i,p,c}\left(\langle w_c, \phi(x_i)\rangle + b_c\right) \geq \gamma_{y_i,p} - \xi_{i,s_{y_i}(p)} \quad 0 \leq i \leq \ell,\ p \in P_{y_i} \\
& \xi_{i,r} \geq 0 \quad 0 \leq i \leq \ell,\ r \in R_{y_i} \\
& \sum_{c=1}^{d} \left(\langle w_c, \phi(x)\rangle + b_c\right) = 0 \quad \Leftrightarrow \quad \sum_{c=1}^{d} w_c = 0 \ and \ \sum_{c=1}^{d} b_c = 0 \qquad (2.13)
\end{aligned}$$

The equality constraint (2.13) is the sum-to-zero constraint that does not appear in all machines. Therefore, this is optional in the unified primal problem.

2.2.8 Online SVMs

Inspired by the potential advantage of online learning and the success of Support Vector Machines, there have been many scientific studies to design effective online learning algorithms for Support Vector Machines. In linear SVMs, significant achievements have been seen over last decade. For example, Pegasos [29] is known as effective and scalable online algorithm working on the primal representation. The reason for its efficiency is that for a linear kernel, the weight vectors w can be computed explicitly. Therefore, the kernel matrix is no longer needed and gradients are cheap to compute in the primal formulation. This situation will change in the non-linear case where the weight vector cannot be expressed explicitly. In that case, optimizing the dual is the most frequently followed proceeding.
LaSVM [5] is an algorithm in the dual representation that tries to employ an optimization scheme inspired by online algorithms in order to more quickly get accuracies close to those of batch solvers. The main idea of LaSVM is to operate in an online learning scheme, each time when new example is presented, the algorithm will perform a so-called PROCESS operation, and then a so-called REPROCESS operation. In order to understand these two operations, it is important to know that LaSVM maintains an index set of support vectors I which correspond to non-zero alpha coefficients. The PROCESS operation will do a SMO optimization step in which one variable corresponds to a new example and another variable is selected from the set of current support vectors I to form a violating pair. The REPROCESS step performs a SMO optimization step on two old support vectors that are the most violating pair. The purpose of PROCESS is to add a new support vector to the set of current support vector S while REPROCESS tries to keep I as small as possible by removing examples which no longer are required as support vectors. The pseudo-code of LaSVM [5] is presented in Algorithm 2.
For multi-class SVMs, LaRank [4] is an extension of LaSVM to perform online learning with

Algorithm 2 LaSVM

```
1: Initialization
   Seed $S$ with some examples from each class
   Set $\alpha \leftarrow 0$ and initialize gradient $g$
2: Online Iteration
   Repeat a predifined number of iterations:
   - An example $k_t$ is selected.
   - Run PROCESS($k_t$)
   - Run REPROCESS once
3: Finishing
   Repeat REPROCESS until convergence criterion reached.
```

the CS SVM formulation. The CS dual objective optimized by LaRank takes the form:

$$\begin{aligned} \text{maximize}_\beta \quad & \sum_{i=1}^{\ell} \beta_i^{y_i} - \frac{1}{2} \sum_{i,j}^{\ell} \sum_{y=1}^{d} \beta_i^y \beta_j^y k(x_i, x_j) \\ \text{Subject to} \quad & \beta_i^{y_i} \leq C\delta(y, y_i), \ 1 \leq i \leq \ell, \ 1 \leq y \leq d \\ & \sum_{y=1}^{d} \beta_i^y = 0, \ 1 \leq i \leq \ell \end{aligned}$$

Here, β_i^y is the coefficient that is associated with the pair (x_i, y). It can be seen that this CS SVM formulation introduces equality constraints corresponding to examples. This means that a SMO update can only be done on variables of the same example. LaRank defines new concepts: support vectors and support patterns. Support vectors are all pairs (x_i, y) whose coefficients β_i^y, $1 \leq i \leq \ell$ are non-zero. Similarly, support patterns are all patterns x_i for which there exist some y, $1 \leq y \leq d$, such that (x_i, y) are support vector. As a result, the PROCESS and REPROCESS steps from LaSVM are extended to PROCESS_NEW, PROCESS_OLD and OPTIMIZE operations in which SMO steps are implemented with different strategies.

- PROCESS_NEW will perform a SMO step on two variables from the new example, which is not yet support pattern at this point. The two variables must define a feasible direction.
- PROCESS_OLD will randomly select a support pattern and then choose two variables that is the most violating pair associated with this support pattern.
- OPTIMIZE: This function will randomly select a support pattern which is similar to PROCESS_OLD. However, it will select most violating pair which are support vectors associated with this support pattern.

Furthermore, the derivative of the dual objective function with respect to variable β_i^y is:

$$g_i(y) = \delta(y_i, y) - \sum_j \beta_j^y k(x_i, x_j).$$

This equation shows that each gradient $g_i(y)$ only relies on coefficients of its own class y. Furthermore, because of the added example-wise equality constraints, PROCESS_OLD only requires d gradient computations. These computations can also be speeded up by exploiting the sparseness structure of support vectors. This leads to the implementation in which gradients needed in PROCESS_NEW and PROCESS_OLD steps are effectively computed. Motivated by this, LaRank chooses to only store and update the gradients of current support vectors. This lowers the memory requirements and makes the gradient updates much faster. As a downside, some gradients need to be computed from scratch in the PROCESS_OLD steps, namely those which do not belong to current support vectors.

Chapter 3

Training Multi-class Support Vector Machines

Training multi-class Support Vector Machines often takes long time, since its complexities scale between quadratically and cubically in the number of examples. Due to these time demand, previous work [13] has focused much on making a solver for these problems as fast as possible, which results in a novel decomposition method. The key elements of this solver strategy are considering hypothesis without bias, exploiting second order information and using working set of size two instead of SMO principle. We will examine a general quadratic problem without equality constraints that a larger number of multi-class SVMs formulations can be easily converted to. After that, the parts of the efficient decomposition method for solving this generalized quadratic problem will also be presented.

3.1 Sequential two-dimensional optimization

The optimization problems in section 2.2.6 are normally solved in dual form. Dogan et al. [13] suggested that the bias terms b_c should be removed. The reason is that bias terms make these problems difficult to solve, because bias terms will introduce equality constraints. These equality constraints require that at each optimization step, at least d variables are simultaneously updated in the LLW and WW SVM formulation. In addition, the bias terms are of minor importance when working with universal kernels.
The dual problems of all primal multi-class SVMs above without bias terms can be generalized to:

$$\max_{\alpha} \; D(\alpha) = v^T\alpha - \frac{1}{2}\alpha^T Q\alpha \tag{3.1}$$
$$\text{Subject to } \; \forall n \in \{1, .., m\} : L_n \leq \alpha_n \leq U_n$$

Here, $\alpha \in \mathbb{R}^m$, $v \in \mathbb{R}$ are some vectors, and $Q \in \mathbb{R}^{mxm}$ is a symmetric positive definite matrix. The gradient $g = \nabla D(\alpha)$ has components $g_n = v_n - \sum_{i=1}^{m} \alpha_i Q_{in}$. As discussed in section 2.2.5, the state of the art algorithms for solving quadratic problem (3.1) are decomposition methods.

The decomposition methods try to iteratively solve sub-quadratic problems in which free variable are restricted to working set B. However, instead of using the smallest working set size as in SMO, [13] suggests using working sets of size two whenever possible, to balance the complexity of solving the subproblems, the availability of well-motivated heuristics for working set selection, the computational cost per decomposition iteration. This method is called sequential two-dimensional optimization (S2DO). The strategy of S2DO is to select the first index according to the gradient value, $i = \arg\max_n |g_n|$ and second component by maximization of the gain. The sequential two-dimensional optimization method is presented in Algorithm 3. In the next

Algorithm 3 Sequential two-dimensional optimization

```
1: Initilize a feasilbe point $\alpha$, $g \leftarrow v - Q\alpha$
2: **repeat**
3:     $i \leftarrow \arg\max_p |g_p|$
4:     $j \leftarrow \arg\max_q computeGain(i, q)$
5:     Solve sub-problem with respect to $B = \{i, j\}$: $\mu \leftarrow solve2D(i, j)$
6:     Update alpha: $\alpha \leftarrow \alpha + \mu$
7:     Update gradient: $g \leftarrow g - Q\mu$
8: **until** Stop conditions are reached
```

parts of this chapter, we will derive how the gain is computed, and how the quadratic problem (3.1) is solved when using working sets of size 2.

3.2 Gain computation for the unconstrained sub-problem

Let the optimization problem (3.1) be restricted to $\alpha_B = [\alpha_i, \alpha_j]^T$ with $B = \{i, j\}$. We want to find the optimal $\hat{\alpha}_B = [\hat{\alpha}_i, \hat{\alpha}_i]^T$ of D without constraints with respect to variables α_i, α_j, and the gain $D(\hat{\alpha}_B) - D(\alpha_B)$.
Let $\mu_B = [\mu_i, \mu_j]^T$ denote the update step size, $\mu_B = \hat{\alpha}_B - \alpha_B$ and $g_B = [g_i, g_j]^T$ denote the gradient where $g_i = \frac{\partial D}{\partial \alpha_i}(\alpha_B)$, $g_j = \frac{\partial D}{\partial \alpha_j}(\alpha_B)$. Taking the Taylor expansion around α_B, we have:

$$D(\alpha_B) = D(\hat{\alpha}_B) - \mu_B^T \nabla D(\hat{\alpha}_B) - \frac{1}{2}\mu_B^T Q_B \mu_B$$

Here, Q_B is the restriction of Q to those variables in B. It can be seen that $\nabla D(\hat{\alpha}_B) = 0$ since $\hat{\alpha}_B$ is the optimal of the unconstrained optimization problem, which results in $\hat{g}_B = g_B - Q_B\mu_B = 0$. Therefore, the gain is $D(\hat{\alpha}_B) - D(\alpha_B) = \frac{1}{2}\mu_B^T Q_B \mu_B$. We have three cases to be considered:

- if $Q_B = 0$ then we have 2 cases. If $g_B = 0$ then the gain is 0. If $g_B \neq 0$ then $D(\alpha_B)$ is a linear function with respect to α_B, thus the gain is infinite.
- if $\det Q_B = 0$ and $Q_B \neq 0$, then at least one of elements $Q_{ii}, Q_{jj} \neq 0$. Without loss of generality, we can assume that $Q_{ii} \neq 0$. In addition, the objective function $D(\alpha_B +$

$\mu_B) = D(\alpha_B) + g_B^T\mu_B - \frac{1}{2}\mu_B^T Q_B\mu_B$ and $g_B = Q_B\mu_B$, then the gain is $\frac{1}{2}g_B^T\mu_B$. Furthermore, $Q_{ii}\mu_i + Q_{ij}\mu_j = g_i \Rightarrow \mu_i = \frac{g_i - Q_{ij}\mu_i}{Q_{ii}}$. As a result, the gain is:

$$\frac{1}{2}(\mu_i g_i + \mu_j g_j) = \frac{g_i(g_i - Q_{ij}\mu_j)}{Q_{ii}} + \mu_j g_j = \frac{g_i^2}{Q_{ii}} + \frac{\mu_j(g_i Q_{ij} - g_j Q_{ii})}{Q_{ii}}$$

If $(g_i Q_{ij} - g_j Q_{ii}) \neq 0$, the gain is infinite. If $(g_i Q_{ij} - g_j Q_{ii}) = 0$, there are numerous ways to assign values of μ_B. We can compute the gain by assuming $\mu_B = \lambda g_B, \lambda \in \mathbb{R}, \lambda \neq 0$, then $\frac{1}{\lambda}g_B = Q_B g_B$. From Rayleigh quotients equation, we have $\lambda = \frac{g_B^T g_B}{g_B^T Q_B g_B}$, then the gain is:

$$\frac{(g_B^T g_B)^2}{2(g_B^T Q_B g_B)} = \frac{(g_i^2 + g_i^2)^2}{2(g_i^2 Q_{ii} + 2g_i g_j Q_{ij} + g_j^2 Q_{jj})}$$

- If $\det(Q_B) \neq 0$ then $\mu_B = Q_B^{-1} g_B$, thus the gain is:

$$\frac{g_i^2 Q_{jj} - 2g_i g_j Q_{ij} + g_j^2 Q_{ii}}{2(Q_{ii}Q_{jj} - Q_{ij}^2)}$$

In summary, the gain computation can be represented as in Algorithm 4

3.3 Parameter update for the constrained sub-problem

After the working pair $B = \{i, j\}$ has been selected, the following sub-problem needs to solve:

$$\max_{\mu_B} \; D(\alpha_B + \mu_B) = D(\alpha_B) + g_B^T\mu_B - \frac{1}{2}\mu_B^T Q_B \mu_B$$
$$\text{Subject to } L_i \leq \alpha_i + \mu_i \leq U_i$$
$$L_j \leq \alpha_j + \mu_j \leq U_j$$

Taking derivative of $D(\alpha_B + \mu_B)$ with respect to μ_B , we obtain:

$$\frac{\partial D}{\partial \mu_B}(\alpha_B + \mu_B) = g_B - Q_B\mu_B$$

Without inequality constraints, $\frac{\partial D}{\partial \mu_B}(\alpha_B+\mu_B)$ vanishes at the optimum, in other words, $g_B = Q_B\mu_B$. However, with constraints, there are three cases that need to be considered:

1. If $Q_B = 0$ then the objective function becomes $\max_{\mu_B} \; D(\alpha_B + \mu_B) = D(\alpha_B) + g_i\mu_i + g_j\mu_j$ such that $\mu_i + \alpha_i \in [L_i, U_i], \mu_j + \alpha_j \in [L_i, U_i]$. Therefore:

Algorithm 4 Gain computation for S2DO

```
Input: $B = \{i, j\}, g_i, g_j, Q_{ii}, Q_{ij}, Q_{jj}$
Output: the gain with respect to variables $\alpha_i,\ \alpha_j$
 1: if $Q_B = 0$ then
 2:     if $g_B = 0$ then
 3:         return 0
 4:     else  return $\infty$
 5:     end if
 6: else if $Q_B \neq 0$ and $det(Q_B) = 0$ then
 7:     if ($Q_{ii} \neq 0$ and $g_i Q_{ij} - g_j Q_{ii} \neq 0$)
 8:         or ($Q_{jj} \neq 0$ and $g_j Q_{ij} - g_i Q_{jj} \neq 0$) then
 9:         return $\infty$
10:     else
11:         return $\frac{(g_i^2 + g_j^2)^2}{2(g_i^2 Q_{ii} + 2g_i g_j Q_{ij} + g_j^2 Q_{jj})}$
12:     end if
13: else
14:     return $\frac{g_i^2 Q_{jj} - 2g_i g_j Q_{ij} + g_j^2 Q_{ii}}{2(Q_{ii}Q_{jj} - Q_{ij}^2)}$
15: end if
```

- To find μ_i, we have several cases that are considered:
  - If $g_i > 0 \Rightarrow \mu_i + \alpha_i = U_i \Rightarrow \mu_i = U_i - \alpha_i$.
  - If $g_i < 0 \Rightarrow \mu_i + \alpha_i = L_i \Rightarrow \mu_i = L_i - \alpha_i$
  - If $g_i = 0 \Rightarrow \mu_i = 0$
- with μ_j the procedure is the same with finding μ_i

2. If $det(Q_B) = 0$ and $Q \neq 0$, then $D(\alpha_B + \mu_B)$ is a convex function. The equality $g_B = Q_B \mu_B$ has no solution or infinite solutions in line $g_i = Q_{ii}\mu_i + Q_{ij}\mu_j$. However, in both cases, the optimum must be on one of the four edges of the constraints $L_i \leq \alpha_i + \mu_i \leq U_i, L_j \leq \alpha_j + \mu_j \leq U_j$ due to the convexity. The optimum is obtained by solving four one-variable sub-problems and selecting the one that gives the best value of $D(\alpha_B + \mu_B)$:

   - If $\alpha_i + \mu_i = U_i$, then $\mu_j = \max(L_j - \alpha_j, \min(\frac{g_j - Q_{ij}(U_i - \alpha_i)}{Q_{jj}}, U_j - \alpha_j))$
   - If $\alpha_i + \mu_i = L_i$, then $\mu_j = \max(L_j - \alpha_j, \min(\frac{g_j - Q_{ij}(L_i - \alpha_i)}{Q_{jj}}, U_j - \alpha_j))$
   - If $\alpha_j + \mu_j = U_j$, then $\mu_i = \max(L_i - \alpha_i, \min(\frac{g_i - Q_{ij}(U_j - \alpha_j)}{Q_{ii}}, U_i - \alpha_i))$
   - If $\alpha_j + \mu_j = L_j$, then $\mu_i = \max(L_i - \alpha_i, \min(\frac{g_i - Q_{ij}(L_j - \alpha_j)}{Q_{ii}}, U_i - \alpha_i))$

3. If Q has full rank, then the unconstrained optimum is:

$$\hat{\mu}_B = Q_B^{-1} g_B = \frac{1}{\det(Q_B)} \begin{pmatrix} Q_{jj} g_i - Q_{ij} g_j \\ Q_{ii} g_j - Q_{ij} g_i \end{pmatrix}$$

   If $\hat{\mu}_B$ satisfies the inequality constraints, it is the optimal value. If not, by convexity, the problem can be transformed to solving one-dimensional problems with one of the inequality constraints active. Therefore, the proceeding is the same as in the case before.

As a result, the procedure that solves the two-dimensional sub-problem is shown in Algorithm 5

Algorithm 5 Solving the two-dimensional sub-problem

```
Input: $B = \{i, j\}, g_i, g_j, Q_{ii}, Q_{ij}, Q_{jj}, \alpha_i, \alpha_j, U_i, L_i, U_j, L_j$
Output: $\mu_i, \mu_j$
 1: Let $Q_B$ be matrix $[Q_{ii}\ Q_{ij};\ Q_{ij}\ Q_{jj}]$
 2: if $Q_{ii} = Q_{ij} = Q_{jj} = 0$ then
 3:     if $g_i > 0$ then $\mu_i \leftarrow U_i - \alpha_i$
 4:     else if $g_i < 0$ then $\mu_i \leftarrow L_i - \alpha_i$
 5:     else $\mu_i \leftarrow 0$
 6:     end if
 7:     if $g_j > 0$ then $\mu_j \leftarrow U_j - \alpha_j$
 8:     else if $g_j < 0$ then $\mu_j \leftarrow L_j - \alpha_j$
 9:     else $\mu_j \leftarrow 0$
10:     end if
11: else if $det(Q_B) = 0$ and $Q_B \neq 0$ then
12:     Let:
13:     $(\mu_{i1}, \mu_{j1}) \leftarrow \big(U_i - \alpha_i, \max(L_j - \alpha_j, \min(\frac{g_j - Q_{ij}(U_i - \alpha_i)}{Q_{jj}}, U_j - \alpha_j))\big)$
14:     $(\mu_{i2}, \mu_{j2}) \leftarrow \big(L_i - \alpha_i, \max(L_j - \alpha_j, \min(\frac{g_j - Q_{ij}(L_i - \alpha_i)}{Q_{jj}}, U_j - \alpha_j))\big)$
15:     $(\mu_{i3}, \mu_{j3}) \leftarrow \big(U_j - \alpha_j, \max(L_i - \alpha_i, \min(\frac{g_i - Q_{ij}(U_j - \alpha_j)}{Q_{ii}}, U_i - \alpha_i))\big)$
16:     $(\mu_{i3}, \mu_{j3}) \leftarrow \big(L_j - \alpha_j, \max(L_i - \alpha_i, \min(\frac{g_i - Q_{ij}(L_j - \alpha_j)}{Q_{ii}}, U_i - \alpha_i))\big)$
17:     $(\mu_i, \mu_j) \leftarrow \arg\max_{k=\overline{1,4}}\big(\mu_{ik} g_i + \mu_{jk} g_j - \frac{1}{2}(Q_{ii}\mu_{ik}^2 + 2Q_{ij}\mu_{ik}\mu_{jk} + Q_{jj}\mu_{jk}^2)\big)$
18: else
19:     $\hat{\mu}_B \leftarrow \frac{1}{\det(Q_B)}\begin{pmatrix} Q_{jj}g_i - Q_{ij}g_j \\ Q_{ii}g_j - Q_{ij}g_i \end{pmatrix}$
20:
21:     if $\hat{\mu}_B \in [L_i, U_i] \times [L_j, U_j]$ then
22:         $\mu_i \leftarrow \hat{\mu}_i$
23:         $\mu_j \leftarrow \hat{\mu}_j$
24:     else
25:         Repeat steps 13-17
26:     end if
27: end if
```

Chapter 4

Online training of the LLW multi-class SVM

Although the LLW SVM has nice the theoretical property of Fisher consistency, training LLW SVM normally takes long time and high computation efforts for large scale datasets. The reason is that training LLW SVMs require solving quadratic optimization problem of the size that is proportional to the number of training examples and the number of classes. As a consequence, this limits the applicability of the LLW SVM. The natural idea is reducing the training time and the computation by approximating the solution of LLW SVMs. The online learning paradigm can be used as an optimization heuristic to to do that, which is introduced in sections 2.1.5, 2.2.8. In this chapter, the LLW SVM will be formulated in the dual form, then online learning paradigm and S2DO method which have already been established both in earlier chapters, are employed to derive an new effective learning algorithm.

4.1 The LLW multi-class SVMs formulation

We introduced the LLW SVM [22] in its primal form in section 2.2.6. Here, we will for convenience give the primal again, then first derive the dual form, then link it to regular SVM solver strategies, and finally, as one contribution of this thesis, derive all necessary elements for an online solver for the LLW SVM.
The LLW SVM is modelled as follow:

$$\min_{w_c} \ \frac{1}{2}\sum_{c=1}^{d}\langle w_c, w_c\rangle + C\sum_{n=1}^{\ell}\sum_{c=1}^{d}\xi_{n,c} \tag{4.1}$$

$$\begin{aligned}
\text{Subject to} \ & \forall n \in \{1,..,\ell\}, c \in \{1,..,d\} \setminus \{y_n\} : \langle w_c, \phi(x_n)\rangle + b_c \le -\frac{1}{d-1} + \xi_{n,c} \\
& \forall n \in \{1,..,l\}, c \in \{1,..,d\} : \xi_{n,c} \ge 0 \\
& \forall h \in \mathcal{H} : \sum_{c=1}^{d}(\langle w_c, h\rangle + b_c) = 0 \Leftrightarrow \quad \sum_{c=1}^{d} w_c = 0 \ and \ \sum_{c=1}^{d} b_c = 0
\end{aligned} \tag{4.2}$$

Equation (4.2) is the sum-to-zero constraint that is equivalent to $\sum_{c=1}^{d} w_c = 0$ and $\sum_{c=1}^{d} b_c = 0$. We next transform the primal optimization problem to the dual representation using the technique of Lagrangian multiplier [8]:

$$\begin{aligned}\mathcal{L} = \;& \frac{1}{2}\sum_{c=1}^{d}\langle w_c, w_c\rangle + C\sum_{n=1}^{\ell}\sum_{c=1}^{d}\xi_{n,c} \\ &+ \sum_{n=1}^{\ell}\sum_{c=1}^{d}\alpha_{n,c}(\langle w_c, \phi(x_n)\rangle + b_c + \frac{1}{d-1} - \xi_{n,c}) \\ &- \sum_{n=1}^{\ell}\sum_{c=1}^{d}\beta_{n,c}\xi_{n,c} - \eta(\sum_{c=1}^{d} w_c) - \tau(\sum_{c=1}^{d} b_c)\end{aligned}$$

Here, $\alpha_{n,c} \geq 0, \beta_{n,c} \geq 0$, and $\eta \in \mathcal{H}$, $\tau \in \mathbb{R}$ are Lagrange multipliers.
Taking derivatives with respect to variables w_c, b_c and $\xi_{n,c}$, respectively, we obtain:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial w_c} &= w_c + \sum_{n=1}^{\ell}\alpha_{n,c}\phi(x_n) - \eta \\ &\Rightarrow w_c = \eta - \sum_{n=1}^{\ell}\alpha_{n,c}\phi(x_n) \qquad (4.3) \\ \frac{\partial\mathcal{L}}{\partial b_c} &= \sum_{n=1}^{\ell}\alpha_{n,c} - \tau \\ &\Rightarrow \tau = \sum_{n=1}^{\ell}\alpha_{n,c} \\ \frac{\partial\mathcal{L}}{\partial \xi_{n,c}} &= C - \alpha_{n,c} - \beta_{n,c} = 0 \\ &\Rightarrow 0 \leq \alpha_{n,c} \leq C, \quad \forall n \in \{1,..,\ell\}, \forall c \in \{1,..,d\} \\ \alpha_{n,y_n} &= 0\end{aligned}$$

Substituting (4.3) into the constraint $\sum_{c=1}^{d} w_c = 0$, we obtain:

$$\begin{aligned}d\eta &= \sum_{c=1}^{d}\sum_{n=1}^{\ell}\alpha_{n,c}\phi(x_n) \\ \Rightarrow \eta &= \frac{1}{d}\left(\sum_{c=1}^{d}\sum_{n=1}^{\ell}\alpha_{n,c}\phi(x_n)\right)\end{aligned}$$

Therefore, the weight w_c can be written as:

$$\begin{aligned} w_c &= \frac{1}{d}(\sum_{p=1}^{d}\sum_{n=1}^{l}\alpha_{n,p}\phi(x_n)) - \sum_{n=1}^{\ell}\alpha_{n,c}\phi(x_n) \\ &= -\sum_{p=1}^{d}\sum_{n=1}^{l}(\delta_{p,c} - \frac{1}{d})\alpha_{n,p}\phi(x_n) \end{aligned} \tag{4.4}$$

Here, $\delta_{p,c}$ denotes the Kronecker symbol.
As a result, we have:

$$\begin{aligned} \mathcal{L} =& \frac{1}{2}\sum_{c=1}^{d}\langle w_c, w_c\rangle + \sum_{n=1}^{\ell}\sum_{c=1}^{d}\xi_{n,c}(C - \alpha_{n,c} - \beta_{n,c}) \\ &+ \sum_{n=1}^{\ell}\sum_{c=1}^{d}\alpha_{n,c}(\langle w_c, \phi(x_n)\rangle) + \sum_{c=1}^{d}b_c(\sum_{n=1}^{\ell}\alpha_{n,c} - \tau) + \frac{1}{d-1}(\sum_{n=1}^{\ell}\sum_{c=1}^{d}\alpha_{n,c}) \\ =& \frac{1}{d-1}(\sum_{n=1}^{\ell}\sum_{c=1}^{d}\alpha_{n,c}) + \frac{1}{2}\sum_{c=1}^{d}\langle w_c, w_c\rangle + \sum_{n=1}^{\ell}\sum_{c=1}^{d}\alpha_{n,c}(\langle w_c, \phi(x_n)\rangle) \end{aligned}$$

Incorporating the result from (4.4) yields:

$$\begin{aligned} \mathcal{L} =& \frac{1}{d-1}(\sum_{n=1}^{\ell}\sum_{c=1}^{d}\alpha_{n,c}) \\ &+ \frac{1}{2}\sum_{c=1}^{d}\Big(\sum_{m=1,n=1}^{\ell}\sum_{p=1,q=1}^{d}(\delta_{p,c} - \frac{1}{d})(\delta_{q,c} - \frac{1}{d})\alpha_{m,p}\alpha_{n,q}\phi(x_m)\phi(x_n)\Big) \\ &- \sum_{m=1,n=1}^{\ell}\sum_{p=1,q=1}^{d}(\delta_{p,q} - \frac{1}{d})\alpha_{m,p}\alpha_{n,q}\phi(x_m)\phi(x_n) \end{aligned}$$

In addition, we can re-arrange:

$$\begin{aligned} &\sum_{c=1}^{d}\sum_{p=1,q=1}^{d}(\delta_{p,c} - \frac{1}{d})(\delta_{q,c} - \frac{1}{d}) \\ &= \sum_{c=1}^{d}\sum_{p=1,q=1}^{d}(\delta_{p,c}\delta_{q,c} - \frac{1}{d}(\delta_{p,c} + \delta_{q,c}) + \frac{1}{d^2}) \\ &= \sum_{p=1,q=1}^{d}(\sum_{c=1}^{d}\delta_{p,c}\delta_{q,c} - \frac{1}{d}\sum_{c=1}^{d}(\delta_{p,c} + \delta_{q,c}) + \frac{1}{d}) \\ &= \sum_{p=1,q=1}^{d}(\delta_{p,q} - \frac{2}{d} + \frac{1}{d}) \\ &= \sum_{p=1,q=1}^{d}(\delta_{p,q} - \frac{1}{d}) \end{aligned}$$

In summary, we have the dual problem:

$$\max_{\alpha} \quad \frac{1}{d-1}\sum_{n=1}^{\ell}\sum_{c=1}^{d}\alpha_{n,c} - \frac{1}{2}\Big(\sum_{m=1,n=1}^{\ell}\sum_{p=1,q=1}^{d}(\delta_{p,q} - \frac{1}{d})\alpha_{m,p}\alpha_{n,q}k(x_m, x_n)\Big) \tag{4.5}$$

$$\text{Subject to} \quad 0 \le \alpha_{n,c} \le C \quad \forall n \in \{1,..,\ell\}, \forall c \in \{1,..,d\}$$

$$\sum_{n=1}^{\ell}\alpha_{n,c} = \tau, \forall c \in \{1,..,d\} \tag{4.6}$$

$$\alpha_{n,y_n} = 0 \quad \forall n \in \{1,..,\ell\}$$

Here, the value of the Lagrange multiplier $\tau \in \mathbb{R}$ can be chosen to some constant value.
It can be seen that equality constraints (4.6) are introduced by bias terms. The previous discussion in section (3.1) is about why bias terms can be removed. This will obviate the equality constraints (4.6). As a result, problem (4.5) will be cast into the generalized problem which can be solved in chapter 3.

4.2 Online learning for LLW SVM

Inspired by LaSVM and LaRank, in this section, an online algorithm for LLW machine is proposed, which replicates LaRank's concepts. Because S2DO has proven superior for batch learning [13], we from the beginning design our online solver to support the S2DO optimization scheme in order to preserve the advantages of this method. Similarly to LaRank, we will define some concepts that are used in online LLW SVM. To clarify, we remind of some definitions in LaRank: a support vector is a pair (x_i, y) whose alpha coefficient $\alpha_{i,y}$ is non-zero, and support pattern be pattern x_i such that (x_i, y) is support vector for some y, $1 \le y \le d$. In addition, we also define I and W , which are the set of current support vectors and support patterns, respectively.

4.2.1 Online step types

In 'classical' online learning [27], online steps only update variables that correspond to new, fresh patterns. However, this will lead to a slow optimization if online learning is employed as optimization heuristic in the dual representation. The reason is that the variables corresponding to the support vectors are not updated often enough [4]. Therefore, in LaSVM and LaRank, various online steps are defined and carried out that update not only variables corresponding to new pattern but also variables corresponding to old support patterns.
Furthermore, in LaRank, the CS-SVM [10] is employed to solve the multi-class classification problem. As a result, there is one equality constraint for each pattern. This restricts the two working variables of SMO in LaRank to operate on the same pattern. However, in the LLW-SVM without bias, there is no equality constraint. As a result, two working variables in S2DO might come from different patterns. These patterns might be support patterns or new patterns, since old non-support patterns are not stored and will become new patterns in the next epoch. Therefore, two working variables might be from the same new, fresh pattern; a new support

pattern and s old support pattern or from two old support patterns. As a result, equivalent online operations for the LLW SVM can be defined as follows:

- TWO_NEWS: This online step will select two variables from a fresh pattern which is not support pattern. This online operation are roughly equivalent to PROCESS_NEW in LaRank.
- ONE_OLD_ONE_NEW: This online step will select one variable from the pattern that has just been inserted into support pattern set. Another one will be chosen from old support patterns.
- TWO_OLD: Two variables will be selected from old support patterns. This operation is equivalent to PROCESS_OLD in LaRank.
- TWO_OLD_SUPPORTS: Two variables are support vectors that are selected from old support patterns. This operation is similar to OPTIMIZE in LaRank

It can be seen that ONE_OLD_ONE_NEW is a special case of TWO_OLD and with our experience, ONE_OLD_ONE_NEW contributes very small to the training process. Therefore, we exclude ONE_OLD_ONE_NEW from online step list.

4.2.2 Working set strategies for online LLW SVM

Online steps in some sense are part of working set selection, especially from the original perspective of solving batch optimization problem, because they restrict the variables from which the working set can be selected. However, in the online learning perspective, online steps are operations that online solvers will perform to adapt with the changes when a new, fresh example comes. Within a specific online step, the purpose of working set selection is to choose the variables that optimize this adaptation. It is also clear that the strategy of working set selection for online steps might affect the performance of a online optimization method. In case of LLW SVM, we will examine several working set selection strategies. Let V is set of possible variables which are determined by a specific online step. Working set selection strategies are defined as below.

- *S2DO_ALL*: Two variables are selected by S2DO over the set of all possible variables V, in particular one variable with highest gradient will be chosen, and the other variable will be selected by highest gain among V.
- *RANDOM_PATTERN_S2DO_ALL*: Firstly, a pattern p is randomly selected from W, then one variable is chosen according to this pattern with highest gradient. After that, the remaining variable is selected through V with highest gain.
- *RANDOM_PATTERN_S2DO_PATTERN*: A pattern p is randomly selected from W, then two variables are selected from this pattern by S2DO.

Because all working set strategies are the same in TWO_NEWS, these different strategies are only applied in ONE_OLD_ONE_NEW, TWO_OLD and TWO_OLD_SUPPORTS online steps. As a result, the working set selection algorithm for online steps is presented in Algorithm 6.

Algorithm 6 Working set selection for online LLW SVM steps

```
Input: step: online step, selectionMethod: working set strategy
Output: (m,p),(n,q)
 1: if step=TWO_NEWS then
 2:     Retrieve new pattern $x_i$
 3:     $m \leftarrow i,\, p \leftarrow \arg\max_{c \in \mathcal{Y}} g_{m,c}$
 4:     $n \leftarrow i,\, q \leftarrow \arg\max_{c \in \mathcal{Y}} computeGain((m,p),(n,c))$
 5: else if step=ONE_OLD_ONE_NEW then
 6:     Retrieve new support pattern $x_i$
 7:     $m \leftarrow i,\, p \leftarrow \arg\max_{c \in \mathcal{Y}} g_{m,c}$
 8:     if selectionMethod=RANDOM_PATTERN_S2DO_PATTERN then
 9:         $n \leftarrow i,\, q \leftarrow \arg\max_{c \in \mathcal{Y}} computeGain((m,p),(n,c))$
10:     else
11:         $(n,q) \leftarrow \arg\max_{j \in P, c \in \mathcal{Y}} computeGain((m,p),(j,c))$
12:     end if
13: else if step=TWO_OLD then
14:     Retrieve possible variables: $V = \{(m,p) | m \in P, p \in \mathcal{Y}\}$
15:     if selectionMethod=S2DO_ALL then
16:         $(m,p) \leftarrow \arg\max_{(i,c) \in V} g_{m,p}$
17:         $(n,q) \leftarrow \arg\max_{(j,e) \in V} computeGain((m,p),(j,e))$
18:     else if selectionMethod=RANDOM_PATTERN_S2DO_ALL then
19:         Select randomly a support pattern $i$ from $W$
20:         $m \leftarrow i,\, p \leftarrow \arg\max_{c \in \mathcal{Y}} g_{m,c}$
21:         $(n,q) \leftarrow \arg\max_{(n,q) \in V} computeGain((m,p),(n,q))$
22:     else if selectionMethod=RANDOM_PATTERN_S2DO_PATTERN then
23:         Select randomly a pattern $i$ from $W$
24:         $m \leftarrow i,\, p \leftarrow \arg\max_{c \in \mathcal{Y}} g_{m,c}$
25:         $n \leftarrow i,\, q \leftarrow \arg\max_{c \in \mathcal{Y}} computeGain((m,p),(n,c))$
26:     end if
27: else if step=TWO_OLD_SUPPORTS then
28:     Retrieve possible variables: $V = \{(m,p) | m \in P, p \in \mathcal{Y}, \alpha_{m,p} \neq 0\}$
29:     if selectionMethod=S2DO_ALL then
30:         $(m,p) \leftarrow \arg\max_{(i,c) \in V} g_{m,p}$
31:         $(n,q) \leftarrow \arg\max_{(j,e) \in V} computeGain((m,p),(j,e))$
32:     else if selectionMethod=RANDOM_PATTERN_S2DO_ALL then
33:         Select randomly a support pattern $i$ from $W$
34:         $m \leftarrow i,\, p \leftarrow \arg\max_{c \in \mathcal{Y}} g_{m,c}$, subject to $\alpha_{m,p} \neq 0$
35:         $(n,q) \leftarrow \arg\max_{(n,q) \in V} computeGain((m,p),(n,q))$
36:     else if selectionMethod=RANDOM_PATTERN_S2DO_PATTERN then
37:         Select randomly a pattern $i$ from $W$
38:         $m \leftarrow i,\, p \leftarrow \arg\max_{c \in \mathcal{Y}} g_{m,c}$, subject to $\alpha_{m,p} \neq 0$
39:         $n \leftarrow i,\, q \leftarrow \arg\max_{c \in \mathcal{Y}} computeGain((m,p),(n,c))$, subject to $\alpha_{n,p} \neq 0$
40:     end if
41: end if
```

4.2.3 Gradient information storage

As discussed earlier, two working variables in the LLW SVM can come from different patterns. Therefore, there might be more gradient calculations needed to find two working variables in online operations. For example, the combination of TWO_OLDS online step with S2DO_ALL strategy needs $|P| * d$ gradients in comparison with only d gradients in PROCESS_OLD in LaRank. Another point is that from (4.5), we have :

$$g_{m,c} = \frac{1}{d-1} - \sum_{n=1}^{|P|} \sum_{e=1}^{d} (\delta_{c,e} - \frac{1}{d})\alpha_{n,e} k(x_m, x_n)$$

Therefore, if gradients are computed on demand as in LaRank, they will take more time. As a result, gradients should be updated for all variables, both support vector and support vector, which correspond to the set of current support patterns. The gradient update rule is:

$$g_{i,c} = g_{i,c} - (\delta_{c,p} - \frac{1}{d})\mu_{m,p} k(x_m, x_i) - (\delta_{c,q} - \frac{1}{d})\mu_{n,q} k(x_n, x_i)$$

As a result, we only compute the gradients that belong to the new pattern in the TWO_NEWS step, and also we only keep and update the gradients if the new pattern becomes a support pattern. This strategy has an advantage that less time is needed to compute gradients. However, as a downside, more variables need to be treated after each update step and more memory is needed to store gradient information.

4.2.4 Scheduling

A natural question is how online operations are scheduled to approximate efficiently the batch LLW SVM solver. There are two common strategies to mix different online steps, namely fixed scheduling and adaptive scheduling. Fixed scheduling [6] will successively run different atomic steps which consist of a number n_O of each online steps. n_O can be called 'multiplier' and different atomic steps can have different multipliers. These multipliers depend on a specific problem, which would be additional hyperparameters for the solver. In contrast, adaptive scheduling [4] will take into account both the computing time and the progress of the dual value to determine which atomic step should be run next and the multipliers of online steps are fixed for all datasets. There is still an open problem which kind of strategy is better, because there are no solid comparison studies investigating or comparing the effect of different step selection strategies. Due to the effectiveness in Larank, we adopt adaptive scheduling in our online solver, but differently, the multipliers of atomic steps are set to the number of classes of training set except for the multiplier of TWO_NEWS is set to 1. The main idea behind this strategy is to select atomic step which gets higher gain in shorter time for next step. Therefore, adaptive scheduling strategy for solving online LLW-SVMs is presented in Algorithm 7. Here, β is a decay parameter of adaptive scheduling algorithm. Our solver also replicate the strategy in LaRank that guarantees the selection of online steps with at least η in probability. The value of β and η are both set to 0.05 to have good performance in LaRank which is also inherited in our online solver.

Algorithm 7 Online LLW SVM solver with adaptive scheduling

```
Input: $\{(x_i, y_i)\}_{i=1}^{\ell}$: dataset, $O$: array of online steps, $multipliers$: array of multiplier values
 1: $P \leftarrow \emptyset$
 2: $\beta \leftarrow 0.05$
 3: $\forall step \in O : stepGains[step] \leftarrow 1$
 4: repeat
 5:     for $k = 1, .., \ell$ do
 6:         $step \leftarrow TWO\_NEWS$
 7:         repeat
 8:             for $j = 1, .., multipliers[step]$ do
 9:                 $(m, p) \leftarrow (0, 0), (n, q) \leftarrow (0, 0)$
10:                 Select two working variables $(m, p), (n, q) \leftarrow selectWorkingSet$
11:                 Perform two variable optimization: $Solve2D\,((m, p), (n, q))$
12:                 Update current support pattern set $W$ according to the value of $\alpha_{m,p}, \alpha_{n,q}$
13:                 Update gradient $\forall i \in P, \forall c \in \mathcal{Y}$ :
14:                 $g_{i,c} \leftarrow g_{i,c} - (\delta_{c,p} - \frac{1}{d})\mu_{m,p} k(x_i, x_m) - (\delta_{c,q} - \frac{1}{d})\mu_{n,q} k(x_i, x_n)$
15:                 Update gain: $stepGains[step] \leftarrow \beta * \frac{dual\_increase}{duration} + (1 - \beta) * stepGains[step]$
16:             end for
17:             Select a $step$ with probability according to $stepGains[step]$
18:         until $step = TWO\_NEWS$
19:     end for
20: until stopping condition is reached
```

4.2.5 Stopping conditions

We do not specify any stopping conditions in Algorithm 7. Normally, for reasonable choices of multipliers for the adaptive scheme, the algorithm gives good result after a single pass over the training dataset. We name it **OnlineLLW** $\times$ 1 to indicate that the algorithm will stop after one epoch.
In other case, we use the same stopping criteria with Shark framework [19] for multi-class SVMs where all gradient values of the variables that are inside feasible region, are less than $\epsilon = 10^{-3}$. We use the name **OnlineLLW** $\times \infty$ to indicate that the algorithm might take several epochs until the stopping conditions are reached. We also name **OnlineLLW** $\times$ 2 to indicate that the online learning will stop after two epoch.

4.2.6 Implementation details

In implementation, we will integrate our online LLW SVM algorithm into Shark [19] framework. Shark is a modular C++ library for the design and optimization of machine learning algorithms such as SVMs and neural network. Furthermore, our online solver still employs the Shark SVM cache mechanism to store kernel values on demand like in the batch SVM solver. The cache mechanism helps the online solver to handle training datasets of which the kernel matrix cannot fit into memory.

4.3 Experiments

Our goal is to derive an online LLW SVM solver that gets a fast approximation to the solution of the LLW SVM solvers, but it achieves comparable accuracy with the batch LLW SVM solvers. Therefore, we performed experiments on a benchmark set of datasets that are established in [4]. The reason is that they were used to establish the good performance of LaRank, thus, we also use them in our own studies. These benchmark datasets span an interesting range and cover several important settings: from mid-sized to larger-scale, and from few-features to many-features. After reporting the optimal hyper-parameters in model selection, we compare the performances of online LLW SVM with batch solvers. We also investigate the impact of kernel cache size on the performance of both online LLW and batch LLW solvers. Finally, we will compare the efficiencies of working set strategies.

4.3.1 Experiment setup

Experiments are carried out on four datasets, namely MNIST, INEX, LETTER and USPS. All four datasets are collected from [4]. Table 4.1 shows the properties of these datasets. We will do experiments with Gaussian kernel for USPS, LETTER and MNIST and with linear kernel for INEX. Our choice of kernels is also like in the original LaRank paper.
During experiments, we will compare the online LLW SVM solvers with the batch LLW SVM solvers, thus C and γ are kept the same for both solvers. We also name **batchLLW_shrink**

	Training size	Test size	Features	Classes
USPS	7291	2007	256	10
INEX	6053	6054	167295	18
LETTER	16000	4000	16	26
MNIST	60000	10000	780	10

Table 4.1: Datasets used in the experiments.

and **batchLLW_unshrink** to indicate the batch LLW SVM solver of Shark with two different settings: using or not using shrinking techniques. In addition, the kernel cache size is set to 3GB for overall experiments and the stopping condition ϵ is 10^{-3}.

4.3.2 Model selection

In order to find good parameters for online LLW-SVM learning, grid search and Jaakkola's heuristic [20] were used. We did grid search with **onlineLLW** $\times \infty$ for all datasets except for MNIST. MNIST was searched with **onlineLLW** $\times 2$ due to its intensive computation time with **onlineLLW** $\times \infty$ and **onlineLLW** $\times 2$ was normally enough to achieve stable accuracy. We did five-fold cross-validation on each point of the grid that we varied $\log_{10}(C) \in \{-1, .., 3\}, \log_2(\gamma) \in \{\log_2(\gamma_{Jaakkola}) - 2, .., \log_2(\gamma_{\text{Jaakkola}}) + 2\}$ for the machines with Gaussian kernel where γ_{Jaakkola} is estimated by Jaakkola's heuristic. With the linear kernel, the search grid was firstly created by $C \in \{10^{-1}, .., 10^3\}$ to find an initial value C_0, the search then was performed in a refine grid $C_0 \cdot 2^{\varphi}, \varphi \in \{-2, -1, 0, 1, 2\}$. In the case of equal cross validation errors, we preferred lower C and γ. If any chosen parameter hits the grid boundary, we shifted the grid such that the former boundary value would be inside the new grid and the new row of the grid was computed and compared. We repeated the shifting until the values of the best parameters did not improve further. The best parameters found in grid search are reported in table 4.2 The procedure of model

	Kernel $k(x, \overline{x})$	C	γ
USPS	$e^{-\gamma\|x-\overline{x}\|^2}$	100	0.0331574
INEX	$x^T \cdot \overline{x}$	5000	-
LETTER	$e^{-\gamma\|x-\overline{x}\|^2}$	1000	0.0512821
MNIST	$e^{-\gamma\|x-\overline{x}\|^2}$	1000	0.0189274

Table 4.2: best parameters found in grid search

selection here differs from in [4] where the optimal hyper-parameters are chosen by manually tuning.

4.3.3 Comparing online LLW with batch LLW

In this part, we compare **onlineLLW**$\times$1 with **batchLLW_shrink**, **batchLLW_unshrink** solvers. This comparison is carried out on the test sets after training both online and batch LLW solvers

on the training sets. We first evaluate the final performance of both solvers at their endpoints to compare the methods as whole, then we will look at the build-up curves and discuss the optimization behaviour in more detail.
As can be seen from the table 4.3, the test errors of **onlineLLW** $\times 1$ are comparable with those of **onlineLLW** $\times \infty$ and **batchLLW**, but it consumes far less computation and training time for all datasets. In detail, **onlineLLW** $\times 1$ is equal or better than **batchLLW** in three over four datasets in terms of test error, although it employs less than one fourth training time of **batchLLW**. Kernel evaluation in **onlineLLW** $\times 1$ is also dramatically lower than that in **batchLLW**. These encouraging results are clearly reflected in Figures 4.1 and 4.2 where the evolutions of the test errors as functions of time and computed kernels are drawn for all datasets. It is clear that one single pass over training sets is sufficient for online learning to nearly reach the performance of its counterpart batch solvers with regards to test error.
There is a drawback of **onlineLLW**$\times 1$ that it does not always approximate well the dual values of the batch solvers. This drawback can be seen from Figure 4.3 where the differences of dual values of **onlineLLW**$\times 1$ and **batchLLW** are significant in INEX and LETTER datasets. The reason is the poor performance of adaptive scheduler in these cases which can improved by increasing the multiplier values of the online operations: TWO_OLD and TWO_OLD_SUPPORTS.

		USPS	INEX	LETTER	MNIST
onlineLLW $\times 1$	Test error (%)	**4.584**	28.6257	**2.25**	**1.49**
	Dual	58136.7	74671100	2262460	701103
	Training time (sec)	**218.611**	**923.528**	**5349.79**	**26631.9**
	Kernel evaluation ($\times 10^6$)	**39.84**	**34.74**	**227.38**	**11416.63**
batchLLW_shrink	Test errror (%)	**4.584**	**27.552**	2.35	**1.49**
	Dual	**58854.7**	**113078068**	**3527538**	**724837**
	Training time (sec)	946.16	18960.5	357575	322660
	Kernel evaluation ($\times 10^6$)	279.38	40.47	2658.46	167505.7
batchLLW_unshrink	Test errror (%)	**4.584**	**27.552**	2.35	**1.49**
	Dual	**58854.7**	113078000	**3527540**	**724837**
	Training time (sec)	608.896	7473.71	200402	359508
	Kernel evaluation ($\times 10^6$)	42.12	36.64	240.25	195585.42

Table 4.3: Comparing online LLW with batch LLW solvers. Here, the S2DO_ALL working set selection is employed for online learning

4.3.4 The impact of cache size

Undoubtedly, the kernel evaluation accounts for large part of the computation time in SVMs. Therefore, the kernel cache affects heavily the training time and the number of kernel evaluations on learning algorithm for SVMs. Here, we want to investigate how the kernel cache size affect the performance of online learning and batch learning. We compare the increasing curves of dual values of both online learning and batch learning for the LLW SVM when the cache size

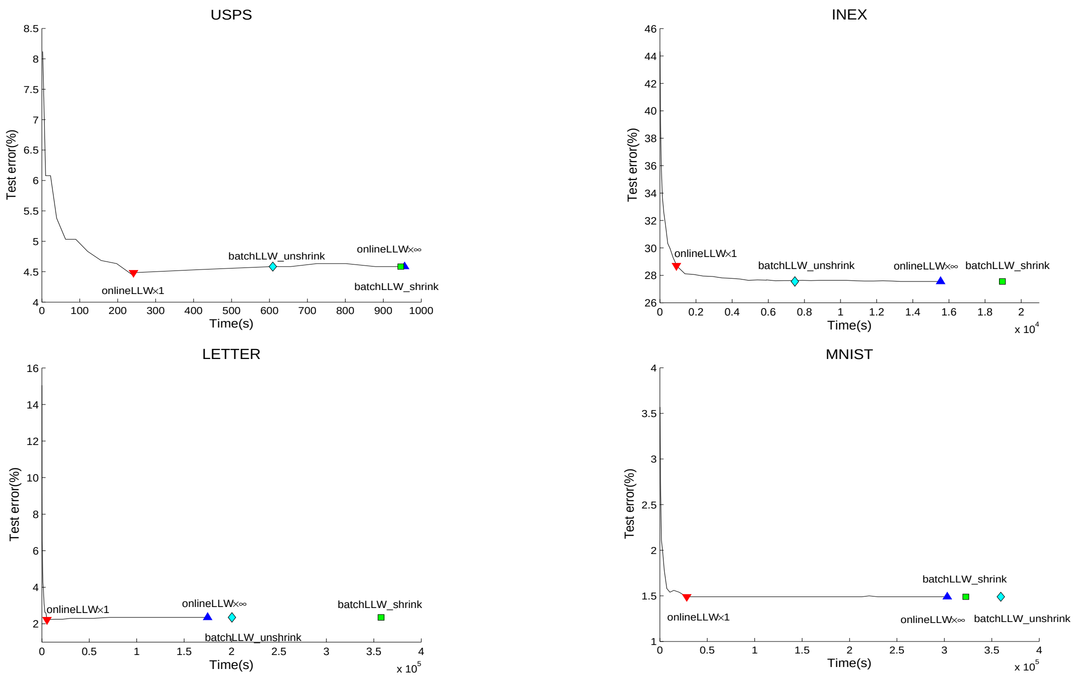


Figure 4.1: Test error of as a function of time

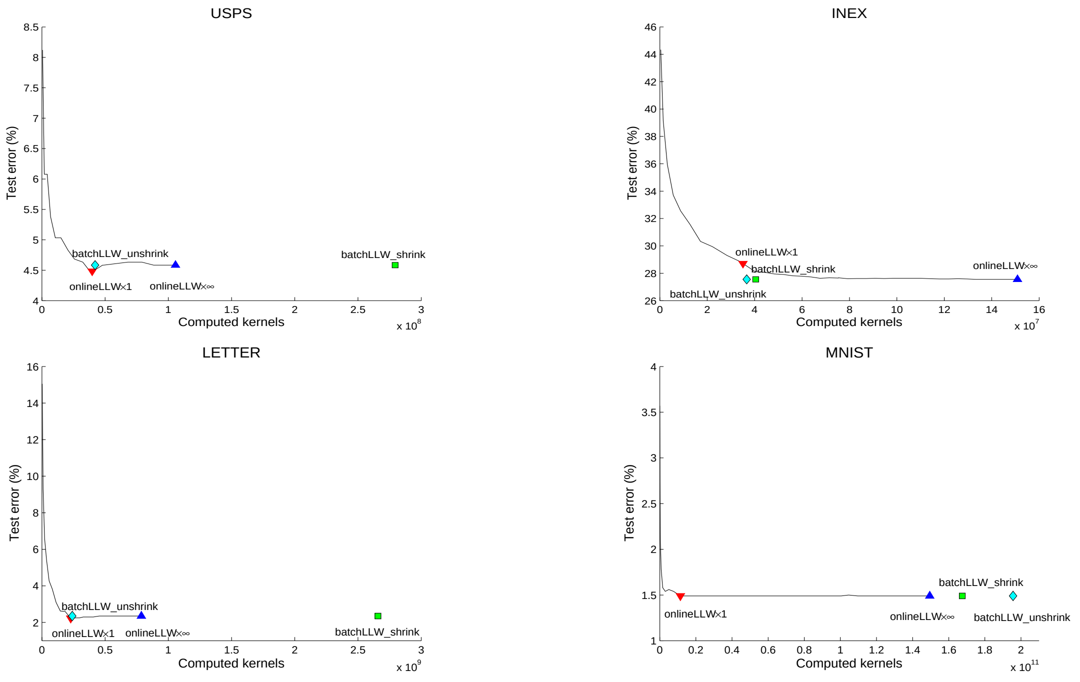


Figure 4.2: Test error as a function of computed kernel

varies at 20 and 120 percent of kernel matrix size. Figures 4.4 and 4.5 have shown that if kernel cache size is limited, in particular 20% of kernel matrix size, the dual value of **onlineLLW** $\times \infty$

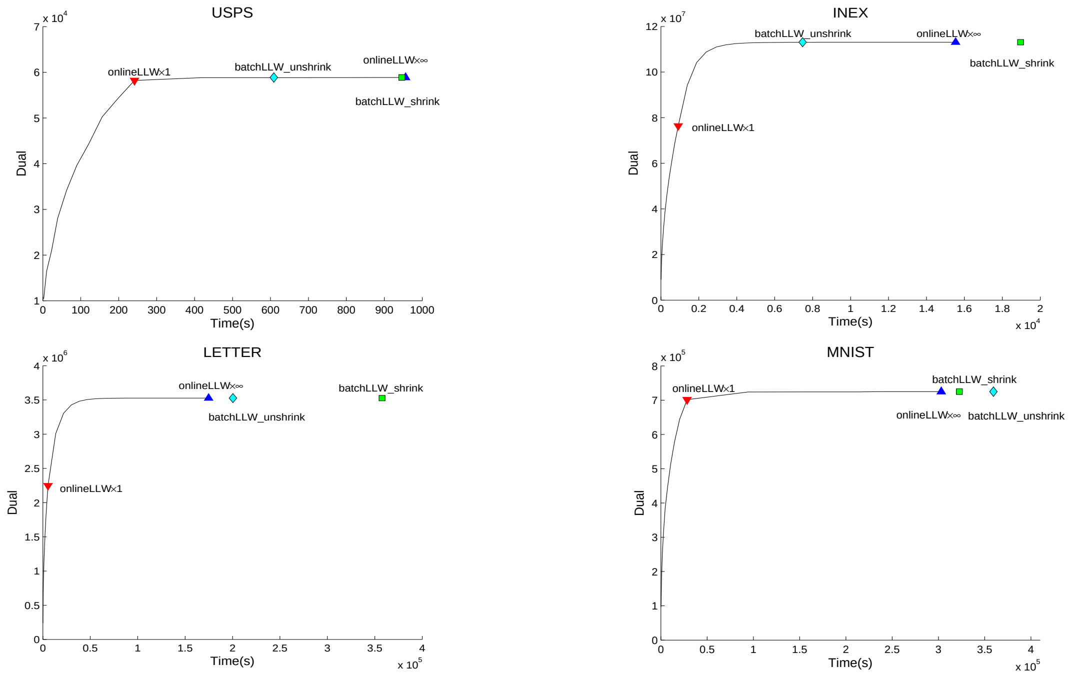


Figure 4.3: Dual as a function of time

increases significantly faster that of **batchLLW** for all experimented datasets. At the same time or the same number of computed kernels, the marked points where the online solvers stop after single pass over training sets, are higher than those of **batchLLW**. The reason is that online optimization heuristic is a kind of bottom up strategy that naturally fits the memory hierarchy design in computer architecture. Therefore, online optimization heuristic exploits the cache mechanism better than batch learning strategy when the memory is limited. This prove the potential applications of online heuristic for approximating the batch optimization problem in large scale problems.
Vice versa, batch learning heuristic can be considered as a top-down strategy when shrinking technique is used. As a result, batch learning has more '*global*' view which can be advantageous if the kernel cache size is enhanced. This explains why the dual values of **batchLLW** solvers increase better and sometimes they are even higher than the dual values of **onlineLLW** $\times$ 1 at the same time or the same computed kernels if the kernel cache size is set to 120% of the kernel matrix size.

4.3.5 Comparison of working set strategies

An interesting question is how do working set strategies perform in LLW model. It is clear from the table 4.4 that there are not much differences in the performance of three different strategies. However, RANDOM_PATTERN_S2DO_PATTERN and RANDOM_PATTERN_S2DO_ALL often give slightly better test errors due to randomization strategies, but they seem to consume more times than S2D0_ALL strategy.

		USPS	INEX	LETTER	MNIST
S2DO_ALL	Test error (%)	4.584	28.6257	2.25	1.49
	Dual	58136.7	74671100	2262460	701103
	Training time (sec)	**218.611**	923.528	**5349.79**	**26631.9**
	Kernel evaluation ($\times 10^6$)	39.84	**34.74**	227.38	11416.63
RANDOM_PATTERN_ S2DO_ALL	Test errror (%)	4.5341	28.64	2.275	**1.44**
	Dual	**58122.6**	74946300	**2265390**	**701059**
	Training time (sec)	220.266	925.101	5413.05	27589.7
	Kernel evaluation ($\times 10^6$)	39.88	34.86	227.54	**11376.33**
RANDOM_PATTERN_ S2DO_PATTERN	Test errror (%)	**4.4843**	**28.5761**	**2.225**	1.52
	Dual	58130.6	**75572700**	2263770	701042
	Training time (sec)	220.317	**914.479**	5454.12	28262.6
	Kernel evaluation ($\times 10^6$)	**39.69**	34.91	**227.36**	11391.79

Table 4.4: Comparing working set strategies of online learning for LLW SVMs

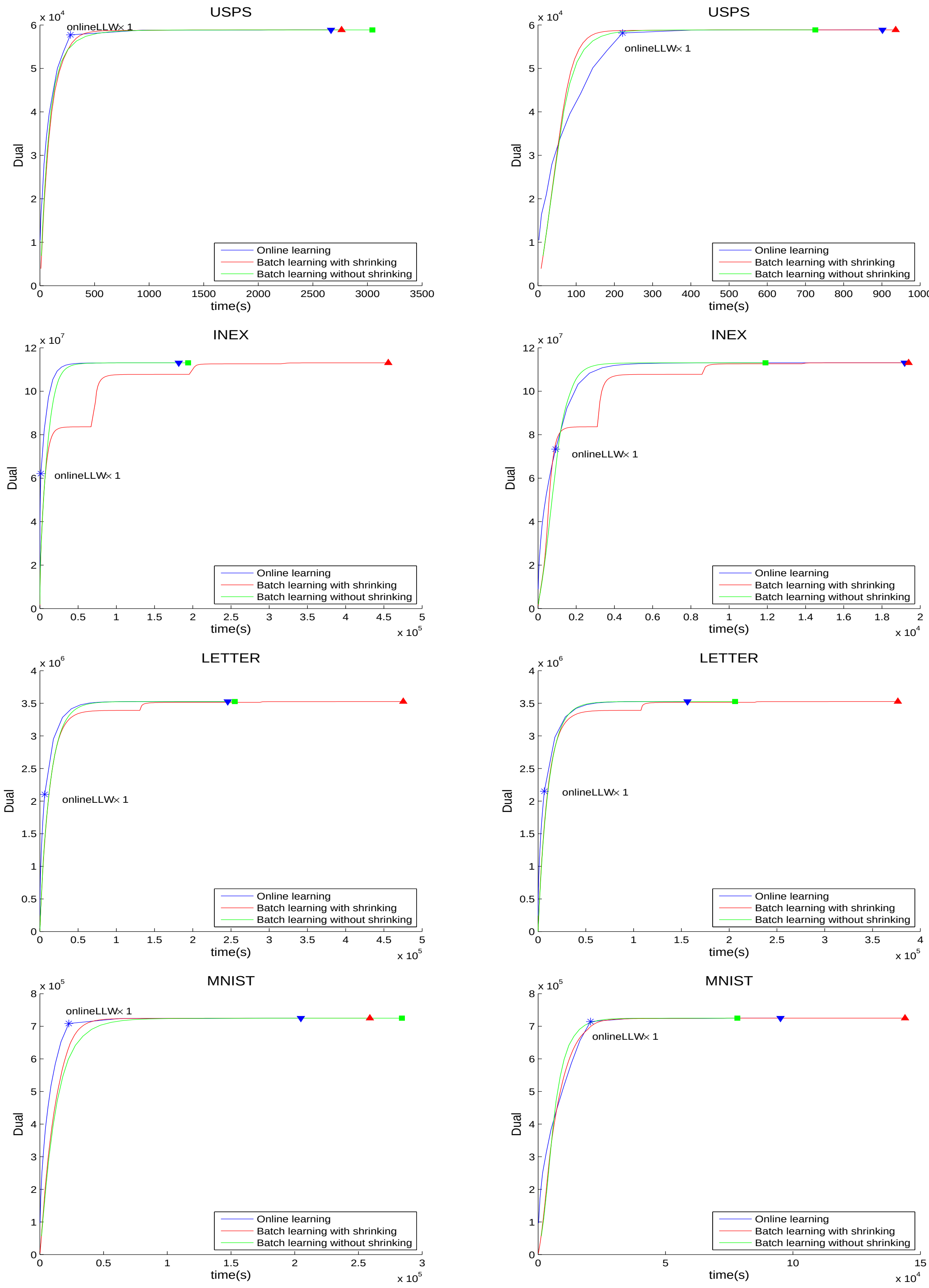


Figure 4.4: the dual curves with respect to time when the cache sizes are 20% and 120% of the kernel matrix sizes, respectively. Here, the left is of 20% the kernel matrix size and the right is of 120% the kernel matrix size

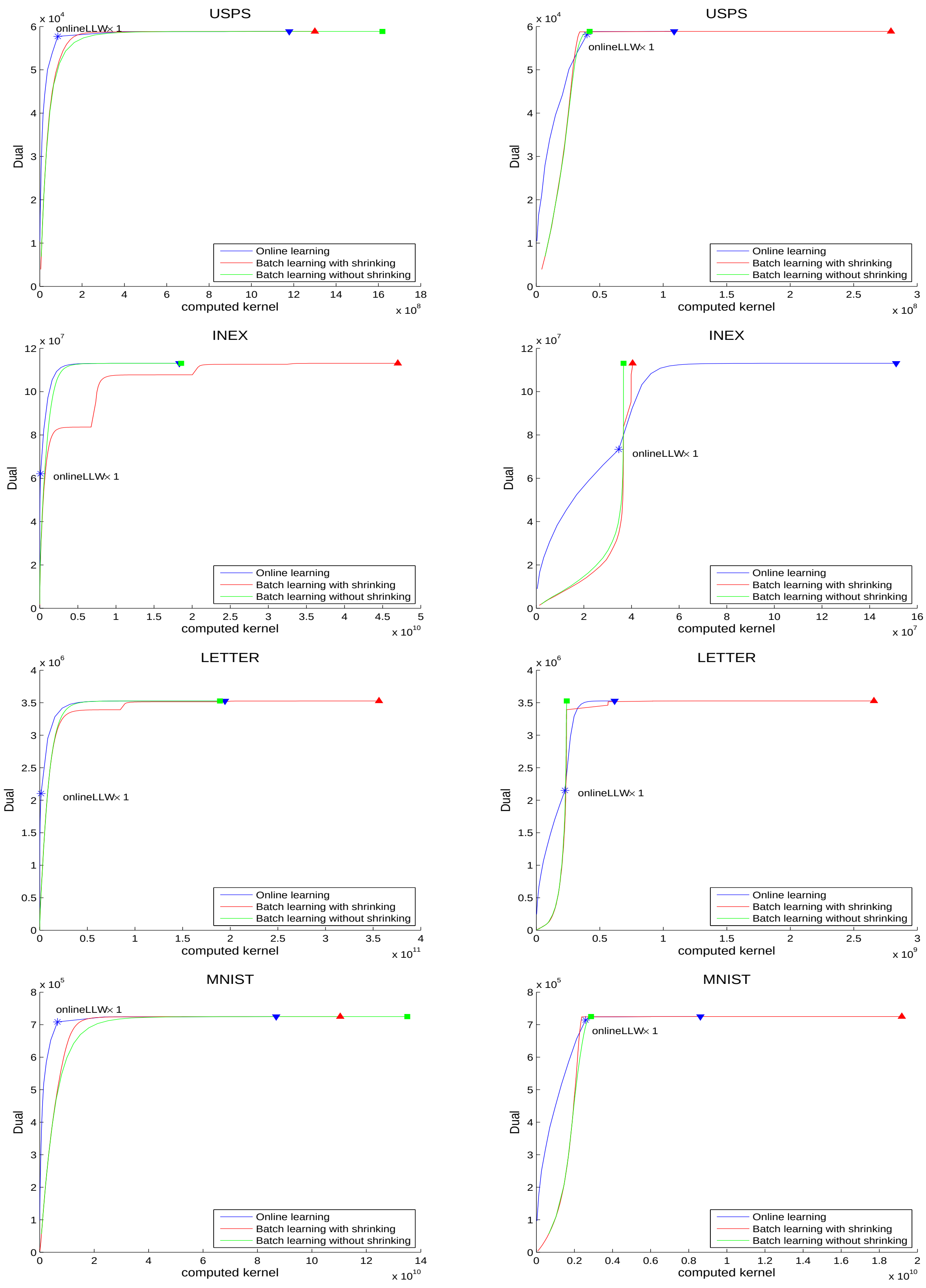


Figure 4.5: the dual curves with respect to kernel evaluation when the cache sizes are 20% and 120% of kernel matrix sizes, respectively. Here, the left is of 20% the kernel matrix size and the right is of 120% the kernel matrix size

Chapter 5

Unified framework for online multi-class SVMs

As discussed in chapter 2, there are various conceptually slightly different proposals for adapting SVMs to the multi-class case. There are the three all-in-one approaches presented in this thesis, namely CS, WW and LLW. They both fit the regularized risk minimization, and their differences come from the different interpretations of the margin concept and the different ways of formulating a loss function on them. The unified framework of these different concepts is recently established, which is discussed in Section 2.2.7. In this chapter, we will first review the dual problem of the unified framework and then apply the online optimization heuristic in Chapter 4 to the unified multi-class SVMs. After that, we perform experiments with the CS SVM as one example of the versatility of the online solver for the unified framework for all-in-one multi-class SVMs.

5.1 The dual problem for the unified framework

The primal problem for the unified framework can be re-stated as:

$$\min_{w,b,\xi} \quad \frac{1}{2}\sum_{c=1}^{d} ||w_c|| + C\sum_{i=1}^{\ell}\sum_{r\in R_{y_i}} \xi_{i,r} \tag{5.1}$$

$$\begin{aligned}
\text{Subject to} \quad & \sum_{c=1}^{d} \nu_{y_i,p,c}\left(\langle w_c, \phi(x_i)\rangle + b_c\right) \geq \gamma_{y_i,p} - \xi_{i,s_{y_i}(p)} \quad 0 \leq i \leq \ell,\ p \in P_{y_i} \\
& \xi_{i,r} \geq 0 \quad 0 \leq i \leq \ell,\ r \in R_{y_i} \\
& \sum_{c=1}^{d} \left(\langle w_c, \phi(x)\rangle + b_c\right) = 0 \quad \Leftrightarrow \quad \sum_{c=1}^{d} w_c = 0 \; and \; \sum_{c=1}^{d} b_c = 0
\end{aligned} \tag{5.2}$$

Normally, the problem (5.1) is solved in its dual form. Transforming equation (5.1) using Lagrangian multipliers like in chapter 4, we have the dual representation:

$$\begin{aligned}
\max_{\alpha} \quad & \sum_{i=1}^{l} \sum_{p \in P_{y_i}} \gamma_{y_i,p} \alpha_{i,p} - \frac{1}{2} \sum_{i=1}^{l} \sum_{p \in P_{y_i}} \sum_{j=1}^{l} \sum_{q \in P_{y_j}} M_{y_i,p,y_j,q} \cdot k(x_i, x_j) \cdot \alpha_{i,p} \cdot \alpha_{j,q} \\
\text{Subject to} \quad & \xi_{i,p} \geq 0 \quad \forall i \in \{0, .., \ell\},\ \forall p \in R_{y_i} \\
& \sum_{p \in P^r_{y_i}} \alpha_{i,r} \leq C \quad \forall i \in \{0, .., \ell\},\ \forall r \in R_{y_i} \\
& \sum_{i=1}^{l} \sum_{p \in P_{y_i}} \nu_{y_i,p,c} \alpha_{i,p} = 0 \quad \forall c \in \{1, .., d\} \qquad (5.3)
\end{aligned}$$

With $M_{y_i,p,y_j,q}$ is defined as:

$$M_{y_i,p,y_j,q} = \sum_{m=1}^{d} \sum_{n=1}^{d} (\delta_{m,n} - \frac{1}{d}) \nu_{y_i,p,m} \nu_{y_j,q,n}$$

And, the set P^r_y, $r \in R_y$, are defined as $P^r_y = s_y^{-1}(\{r\}) = \{p \in P_y \,|s_y(p) = r\}$.
As discussed earlier, because of the minor importance with universal kernel and the difficulties in solving, bias terms should be remove. Using machines without bias term results in deleting the equality constraints (5.3). As a result, the unified problem can be written as:

$$\max_{\alpha} D(\alpha) = \sum_{i=1}^{l} \sum_{p \in P_{y_i}} \gamma_{y_i,p} \alpha_{i,p} - \frac{1}{2} \sum_{i=1}^{l} \sum_{p \in P_{y_i}} \sum_{j=1}^{l} \sum_{q \in P_{y_j}} M_{y_i,p,y_j,q} \cdot k(x_i, x_j) \cdot \alpha_{i,p} \cdot \alpha_{j,q} \qquad (5.4)$$

$$\begin{aligned}
\text{Subject to} \quad & \xi_{i,p} \geq 0 \quad \forall i \in \{0, .., \ell\},\ \forall p \in R_{y_i} \\
& \sum_{p \in P^r_{y_i}} \alpha_{i,r} \leq C \quad \forall i \in \{0, .., \ell\},\ \forall r \in R_{y_i} \qquad (5.5)
\end{aligned}$$

5.2 Online learning for the unified framework

In this section, we will derive a learning algorithm that uses an online optimization heuristic to approximate the solution of the unified problem (5.4). In detail, we will adapt the online operations and the dynamic scheduling presented in the chapter 4 to solve the unified problem. However, because of the more general formulation, the working set selection for online operations and the procedure for solving the sub-problems will be changed and presented in the next parts of this section.

5.2.1 Online step types and working set selection

It can be seen that two working variables can come across patterns due to the absence of equality constraints. Therefore, online steps which were introduced in the section 4.2.1, are still applied.

However, as before, we do not include ONE_OLD_ONE_NEW since it contributes little to the learning process in practice.
With working set selection strategies for online operations, we still employ different strategies that are defined in the section 4.2.2. However, there is a slight change in the way that two working variable are selected due to the inequality constraints (5.5). This change is inherited from the modified S2DO in the bath learning for the unified problem which is presented in [14]. In the batch learning, the two working variables are selected as follows.

- If $\max\{|g_{m,c}|\} \geq \max\{g_{m,c} - g_{n,e}\}$, then selecting the first working variable as $(i,p) = \arg\max\{|g_{m,c}|\}$, and the second working variable is chosen as $\arg\max$ *computeGain* $((i,p)-(n,e))$. Here, the procedure *computeGain* is the same as in section 3.2.
- If $\max\{|g_{m,c}|\} \leq \max\{(g_{m,c}-g_{n,e})\}$, then the two working variables are selected as $((i,p),(j,q)) = \arg\max\{(g_{m,c} - g_{n,e})\}$.

As a result, let W be the set of current support patterns and V be the set of possible variables. The working set selections for online operations are presented in Algorithms 8, 9, 10.

Algorithm 8 Working set selection for TWO_NEWS online step

```
Input: selectionMethod: working set strategy
Output: (m,p),(n,q)
1: Retrieve new pattern $x_i$
2: if $|P^r_y| = 1$ then
3:     $m \leftarrow i,\ p \leftarrow \arg\max_{c\in\mathcal{Y}} g_{m,c}$
4:     $n \leftarrow i,\ q \leftarrow \arg\max_{c\in\mathcal{Y}} computeGain((m,p),(n,c))$
5: else
6:     $m \leftarrow i,\ p \leftarrow \arg\max_{c\in\mathcal{Y}} g_{m,c}$
7:     $n \leftarrow i,\ q \leftarrow \arg\max_{c\in\mathcal{Y}} \left(g_{m,p} - g_{n,c}\right)$
8: end if
```

5.2.2 Parameter update

When the working set has been selected, we want to solve the problem 5.4 with respect to the variables in the working set. Let denote the working set be denote by two variables $B = \{(i,p),(j,q)\}$ and the quadratic matrix restricted to the working set

$$Q_B = \begin{pmatrix} Q_{(i,p),(i,p)} & Q_{(i,p),(j,q)} \\ Q_{(j,q),(i,p)} & Q_{(j,q),(j,q)} \end{pmatrix}$$

Here, $Q_{(i,p),(j,q)} = M_{y_i,p,y_j,q} \cdot k(x_i,x_j)$. Also, let the gradient be $g_B = (g_{(i,p)}, g_{(j,q)})$ where $g_{(i,p)} = \partial D(\alpha)/\partial\alpha_{(i,p)}$. The sub-problem is to find the update $\mu_B = (\mu_{i,p}, \mu_{j,q})$ that maximizes the gain: $\max_\mu D(\alpha_B + \mu_B) - D(\alpha_B) = g_B^T\mu_B - \frac{1}{2}\mu_B^T Q_B \mu_B$, where $\alpha_B = (\alpha_{i,p}, \alpha_{j,q})$. There are two cases to be considered as follows.

Algorithm 9 Working set selection for TWO_OLDS online step

```
Input: step: online step, selectionMethod: working set strategy
Output: (m,p),(n,q)
 1: Retrieve possible variables: $V = \{(m, p) \mid m \in W, p \in \mathcal{Y}\}$
 2: if selectionMethod=S2DO_ALL then
 3:     if $|P^r_y| = 1$ then
 4:         $(m, p) \leftarrow \arg\max_{(i,c)\in V} g_{m,p}$
 5:         $(n, q) \leftarrow \arg\max_{(j,e)\in V} \mathit{computeGain}((m, p), (j, e))$
 6:     else
 7:         $g_1 \leftarrow \max_{(i,c)\in V}\{g_{i,c}\}$
 8:         $g_2 \leftarrow \max_{(i,c),(i,e)\in V}\{(g_{i,c} - g_{i,e}) \mid \sum_{b\in R^r_{y_i}} \alpha_{i,b} = C\}$
 9:         if $g_1 \geq g_2$ then
10:             $(m, p) \leftarrow \arg\max_{(i,c)\in V} g_{m,p}$
11:             $(n, q) \leftarrow \arg\max_{(j,e)\in V} \mathit{computeGain}((m, p), (j, e))$
12:         else
13:             $((m, p), (n, q)) \leftarrow \arg\max_{(i,c),(i,e)\in V}\{(g_{i,c} - g_{i,e}) \mid \sum_{b\in R^r_y} \alpha_{i,b} = C\}$
14:         end if
15:     end if
16: else if selectionMethod=RANDOM_PATTERN_S2DO_ALL then
17:     Select randomly a support pattern $i$ from $W$
18:     if $|P^r_y| = 1$ then
19:         $m \leftarrow i$, $p \leftarrow \arg\max_{c\in\mathcal{Y}} g_{m,c}$
20:         $(n, q) \leftarrow \arg\max_{(n,e)\in V} \mathit{computeGain}((m, p), (n, e))$
21:     else
22:         if $\sum_{b\in R^r_{y_i}} \alpha_{i,b} = C$ then
23:             $((m, p), (n, q)) \leftarrow \arg\max_{(i,c),(i,e)\in V}\{(g_{i,c} - g_{i,e}) \mid \sum_{b\in R^r_y} \alpha_{i,b} = C\}$
24:         else
25:             $m \leftarrow i$, $p \leftarrow \arg\max_{c\in\mathcal{Y}} g_{m,c}$
26:             $(n, q) \leftarrow \arg\max_{(n,c)\in V} \mathit{computeGain}((m, p), (n, c))$
27:         end if
28:     end if
29: else if selectionMethod=RANDOM_PATTERN_S2DO_PATTERN then
30:     Select randomly a pattern $i$ from $W$
31:     if $|P^r_y| = 1$ then
32:         $m \leftarrow i$, $p \leftarrow \arg\max_{c\in\mathcal{Y}} g_{m,c}$
33:         $n \leftarrow i$, $q \leftarrow \arg\max_{c\in\mathcal{Y}} \mathit{computeGain}((m, p), (n, c))$
34:     else
35:         $m \leftarrow i$, $p \leftarrow \arg\max_{c\in\mathcal{Y}} g_{m,c}$
36:         $n \leftarrow i$, $q \leftarrow \arg\max_{c\in\mathcal{Y}} \left(g_{m,p} - g_{n,c}\right)$
37:     end if
38: end if
```

Algorithm 10 Working set selection for TWO_OLD_SUPPORTS online step

Input: step: online step, selectionMethod: working set strategy

Output: (m,p),(n,q)

```
1: Retrieve possible variables: $V = \{(m, p) \mid m \in W, p \in \mathcal{Y}, \alpha_{m,p} \neq 0\}$
2: if selectionMethod=S2DO_ALL then
3:     if $|P^r_y| = 1$ then
4:         $(m, p) \leftarrow \arg\max_{(i,c)\in V} g_{m,p}$
5:         $(n, q) \leftarrow \arg\max_{(j,e)\in V} computeGain((m, p), (j, e))$
6:     else
7:         $g_1 \leftarrow \max_{(i,c)\in V}\{g_{i,c}\}$
8:         $g_2 \leftarrow \max_{(i,c),(i,e)\in V}\{(g_{i,c} - g_{i,e}) \mid \sum_{b\in R^r_{y_i}} \alpha_{i,b} = C\}$
9:         if $g_1 \geq g_2$ then
10:            $(m, p) \leftarrow \arg\max_{(i,c)\in V} g_{m,p}$
11:            $(n, q) \leftarrow \arg\max_{(j,e)\in V} computeGain((m, p), (j, e))$
12:        else
13:            $((m, p), (n, q)) \leftarrow \arg\max_{(i,c),(i,e)\in V}\{(g_{i,c} - g_{i,e}) \mid \sum_{b\in R^r_y} \alpha_{i,b} = C\}$
14:        end if
15:    end if
16: else if selectionMethod=RANDOM_PATTERN_S2DO_ALL then
17:    Select randomly a support pattern $i$ from $W$
18:    if $|P^r_y| = 1$ then
19:        $m \leftarrow i$, $p \leftarrow \arg\max_{c\in\mathcal{Y}}\{g_{m,c} \mid (m, c) \in V\}$
20:        $(n, q) \leftarrow \arg\max_{(n,e)\in V} computeGain((m, p), (n, e))$
21:    else
22:        if $\sum_{b\in R^r_{y_i}} \alpha_{i,b} = C$ then
23:            $((m, p), (n, q)) \leftarrow \arg\max_{(i,c),(i,e)\in V}\{(g_{i,c} - g_{i,e}) \mid \sum_{b\in R^r_y} \alpha_{i,b} = C\}$
24:        else
25:            $m \leftarrow i$, $p \leftarrow \arg\max_{c\in\mathcal{Y}}\{g_{m,c} \mid (m, c) \in V\}$
26:            $(n, q) \leftarrow \arg\max_{(n,e)\in V} computeGain((m, p), (n, e))$
27:        end if
28:    end if
29: else if selectionMethod=RANDOM_PATTERN_S2DO_PATTERN then
30:    Select randomly a pattern $i$ from $W$
31:    if $|P^r_y| = 1$ then
32:        $m \leftarrow i$, $p \leftarrow \arg\max_{c\in\mathcal{Y}}\{g_{m,c} \mid (m, c) \in V\}$
33:        $n \leftarrow i$, $q \leftarrow \arg\max_{c\in\mathcal{Y}} \{computeGain((m, p), (n, c)) \mid (n, c) \in V\}$
34:    else
35:        $m \leftarrow i$, $p \leftarrow \arg\max_{c\in\mathcal{Y}}\{g_{m,c} \mid (m, c) \in V\}$
36:        $n \leftarrow i$, $q \leftarrow \arg\max_{c\in\mathcal{Y}} \left\{(g_{m,p} - g_{n,c}) \mid (n, c) \in V\right\}$
37:    end if
38: end if
```

1. If $i \neq j$ or $s_{y_i}(p) \neq s_{y_j}(q)$, then the sub-problem becomes:

$$\begin{aligned}\max_{\mu_B} D(\alpha_B + \mu_B) - D(\alpha) =& g_B^T \mu_B - \frac{1}{2}\mu_B^T Q_B \mu_B \\ \text{Subject to} \quad & 0 \leq \alpha_{i,p} + \mu_{i,p} \leq U_1 \\ & 0 \leq \alpha_{j,q} + \mu_{j,q} \leq U_2\end{aligned}$$

   Here, $U_1 = C - \sum_{e \in P_{y_i}^{r_1}} \{\alpha_{i,e} \mid e \neq p\}$, $r_1 = s_{y_i}(p)$, and $U_2 = C - \sum_{e \in P_{y_j}^{r_2}} \{\alpha_{j,e} \mid e \neq q\}$, $r_2 = s_{y_j}(q)$. As can be seen, this case is solved by the technique in 3.3.

2. If $i = j$ and $s_{y_i}(p) = s_{y_j}(q)$, the sub-problem become:

$$\max_{\mu_B} D(\alpha_B + \mu_B) - D(\alpha) = g_B^T \mu_B - \frac{1}{2}\mu_B^T Q_B \mu_B \tag{5.6}$$

$$\text{Subject to} \quad \alpha_{i,p} + \mu_{i,p} \geq 0 \tag{5.7}$$

$$\alpha_{j,q} + \mu_{j,q} \geq 0 \tag{5.8}$$

$$\alpha_{i,p} + \mu_{i,p} + \alpha_{j,q} + \mu_{j,q} \leq U \tag{5.9}$$

   Here, $U = C - \sum_{e \in P_{y_i}^{r}} \{\alpha_{i,e} \mid e \neq p,\ e \neq q\}$, $r = s_{y_i}(p) = s_{y_j}(q)$. In this case, we have three smaller cases to be considered:

   - if $Q_B = 0$, the sub-problem become a linear optimization problem in which the optimal solution must be at the point where two of the three inequality constraints are activated to be equality constraints. Therefore, by examining the feasible solutions at points where two of the three inequality constraints are active, we select the optimal one that maximizes the gain.
   - if $\det Q_B = 0$ and $Q_B \neq 0$, the objective function is convex with respect to μ_B. In addition, the unconstrained optimum must be in the geometric line $g_B = Q_B \mu_B^T$. Therefore, by convexity, the constrained optimum must be in line when one of the three inequality constraints becomes an equality constraint. As a result, we can reduce the two dimensional sub-problem to several single variable problems. Solving the single variable problem gives us one feasible solution, and then we select the optimal one that maximizes the gain.
   - if $\det Q_B > 0$, the unconstrained optimum is $\widehat{\mu}_B = Q_B^{-1} g_B$. If this unconstrained optimum satisfies all three constraints, it is the optimal solution. Otherwise, the optimal solution in the geometric line when one of the three inequality constraints becomes equality constraint as was the case above.

   It can be seen that in all cases, we can reduce the problem (5.6) to single variable optimization problems. As a result, the procedure for solving the problem (5.4) when $i = j$ and $s_{y_i}(p) = s_{y_j}(q)$, is presented in Algorithm 11

Algorithm 11 Solving the two-dimensional sub-problem when $i = j$ and $s_{y_i}(p) = s_{y_j}(q)$

```
Input: $B = \{(i, p), (j, q)\}$, $g_{i,p}$, $g_{j,q}$, $Q_{(i,p),(i,p)}$, $Q_{(i,p),(j,q)}$, $Q_{(j,q),(j,q)}$, $\alpha_{i,p}$, $\alpha_{j,q}$, U
Output: $\mu_{i,p}, \mu_{j,q}$
 1: Let $Q_B$ be matrix $[Q_{(i,p),(i,p)}\ Q_{(i,p),(j,q)}\ ;\ Q_{(i,p),(j,q)}\ Q_{(j,q),(j,q)}]$
 2: $\overline{\mu}_{i,p} \leftarrow \frac{1}{\det Q_B}\left(Q_{(j,q),(j,q)}g_{i,p} - Q_{(i,p),(j,q)}g_{j,q}\right)$
 3: $\overline{\mu}_{j,q} \leftarrow \frac{1}{\det Q_B}\left(Q_{(i,p),(j,q)}g_{i,p} - Q_{(i,p),(i,p)}g_{j,q}\right)$
 4: if $(\overline{\mu}_{i,p} + \alpha_{i,p} \geq 0)$ & $(\overline{\mu}_{j,q} + \alpha_{j,q} \geq 0)$ & $(\overline{\mu}_{i,p} + \overline{\mu}_{j,q} \leq U - \alpha_{i,p} - \alpha_{j,q})$ then
 5:     $\mu_{i,p} \leftarrow \overline{\mu}_{i,p}$
 6:     $\mu_{j,q} \leftarrow \overline{\mu}_{j,q}$
 7: else
 8:     $\overline{\mu}^1_{i,p} \leftarrow -\alpha_{i,p}$
 9:     $\overline{\mu}^1_{j,q} \leftarrow$ solveSingleVariable($\mu_{j,q}$) subject to $0 \leq \mu_{j,q} + \alpha_{j,q} \leq U$
10:     $\overline{\mu}^2_{j,q} \leftarrow -\alpha_{j,q}$
11:     $\overline{\mu}^2_{i,p} \leftarrow$ solveSingleVariable($\mu_{i,p}$) subject to $0 \leq \mu_{i,p} + \alpha_{i,p} \leq U$
12:     $\overline{\mu}^3_{i,p} \leftarrow$ solveSingleVariable($\mu_{i,p}$) subject to $0 \leq \mu_{i,p} + \alpha_{i,p} \leq U - \alpha_{j,q}$
13:     $\overline{\mu}^3_{j,q} \leftarrow U - \mu_{j,q} - \alpha_{j,q} - \alpha_{j,q}$
14:     $\left(\mu_{i,p}, \mu_{j,q}\right) \leftarrow \arg\max_{v=\overline{1,3}}$ computeGain($\mu^v_{i,p}, \mu^v_{j,q}$)
15: end if
```

5.3 Experiments

In this section, we will do experiments about the CS SVM as an example of the unified framework. The purpose of these experiments is to investigate the performance of online leaning for CS SVM as an approximative solver for the batch CS SVM.
As in chapter 4, the experiments are carried out on four datasets, namely MNIST, INEX, LETTER and USPS since they were established as a benchmark suite in the LaRank paper [4]. After showing the optimal hyper-parameters selected for each dataset, we report the performance of online learning through the comparison with that of batch learning for the CS SVM. We also investigate the effect of working set selection strategies for online steps on the performance of online learning for the CS SVM.
In order to easy distinguish, we name **onlineCS** $\times$ 1 to indicate that the online learning for the CS SVM will stop after single pass over dataset, **onlineCS** $\times\ \infty$ indicate that the online learning will take more epochs until stopping conditions are reached. We also name **batchCS_shrink** and **batchCS_unshrink** to indicate the batch learning with and without shrinking techniques for the CS SVM that are developed in Shark.
It is clear that C and γ are crucial hyper-parameter that affects the performance of SVMs. Therefore, we employ the same strategy and heuristic as in section 4.3.2 to find the optimal hyper-parameters for each dataset. We also exploit the grid search with shifting heuristic where as in section 4.3.2, we shift the boundary of the grid as necessary to make sure the optimal hyper-

parameters are always inside the grid. The optimal hyper-parameters are reported in table 5.1.

	Kernel $k(x, \overline{x})$	C	γ
USPS	$e^{-\gamma\|x-\overline{x}\|^2}$	10	0.0331574
INEX	$x^T \cdot \overline{x}$	500	-
LETTER	$e^{-\gamma\|x-\overline{x}\|^2}$	10	0.0512821
MNIST	$e^{-\gamma\|x-\overline{x}\|^2}$	100	0.0378548

Table 5.1: Best parameters for CS found in grid search

5.3.1 Comparing online CS with batch CS solvers

In this section, we will compare the performance of the **onlineCS** $\times$ 1 with the batch CS solvers **batchCS_shrink**, **onlineCS_unshrink**. For fairly comparison, we set the kernel cache is 3GB. In addition, RANDOM_PATTERN_S2DO_ALL working set selection strategy is used for online learning for CS SVM.
As can be seen from table 5.2, the **onlineCS** $\times$ 1 performs better than both the batch CS solvers in terms of training time in three over four benchmark datasets. While the **onlineCS** $\times$ 1 solver achieves nearly the same dual values as the batch CS solvers, its test errors are comparable with the batch CS solvers. This results are also reflected in Figures 5.1 and 5.3 where the evolution of test error as a function of time and a function of computed kernel are plotted.

		USPS	INEX	LETTER	MNIST
onlineCS $\times$ 1	Test errror (%)	**4.6338**	**26.0489**	**1.9**	**1.42**
	Dual	1734.42	997797	9231.64	9972.75
	Training time (sec)	58.8231	**355.564**	**241.13**	**3260.99**
	Kernel evaluation ($\times 10^6$)	28.01	**33.32**	147.37	**1219.91**
batchCS_shrink	Test errror (%)	**4.6338**	26.3297	2.0	1.45
	Dual	**1738.95**	987710	**9326.2**	**10001.3**
	Training time (sec)	161.013	894.974	818.044	11249.6
	Kernel evaluation ($\times 10^6$)	90.75	85.51	1049.15	5512.04
batchCS_unshrink	Test errror (%)	**4.6338**	26.115	2.0	1.45
	Dual	**1738.96**	**1010980**	**9326.37**	**10001.3**
	Training time (sec)	**38.9461**	378.035	429.231	7689
	Kernel evaluation ($\times 10^6$)	**15.61**	33.40	**112.43**	4288.74

Table 5.2: Comparing online CS with batch CS solvers.

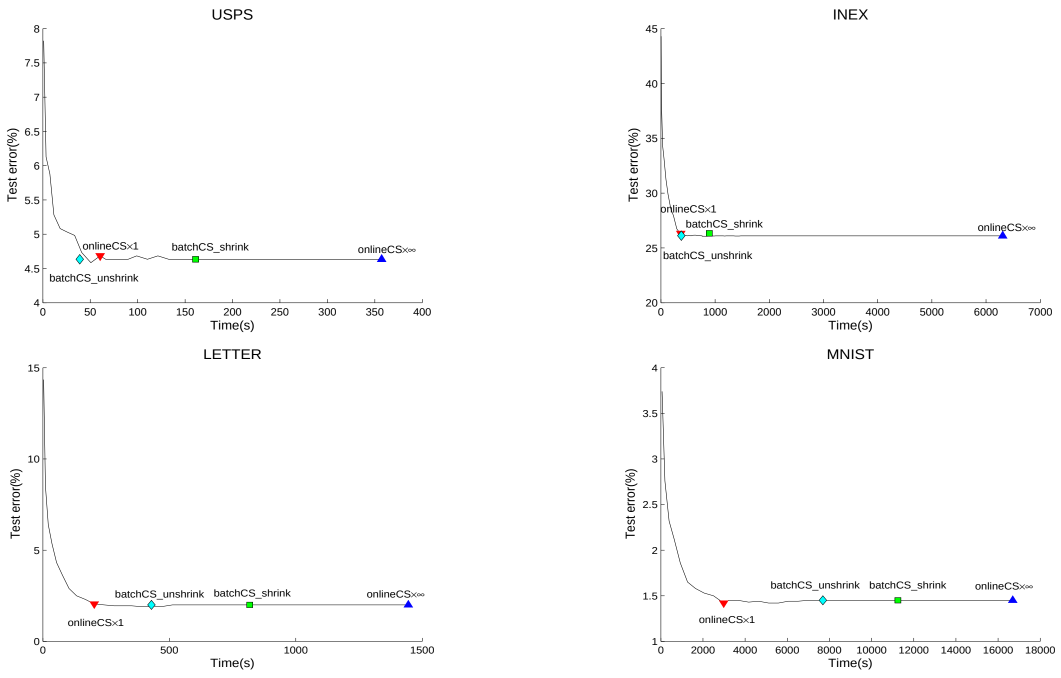


Figure 5.1: Test error of as a function of time

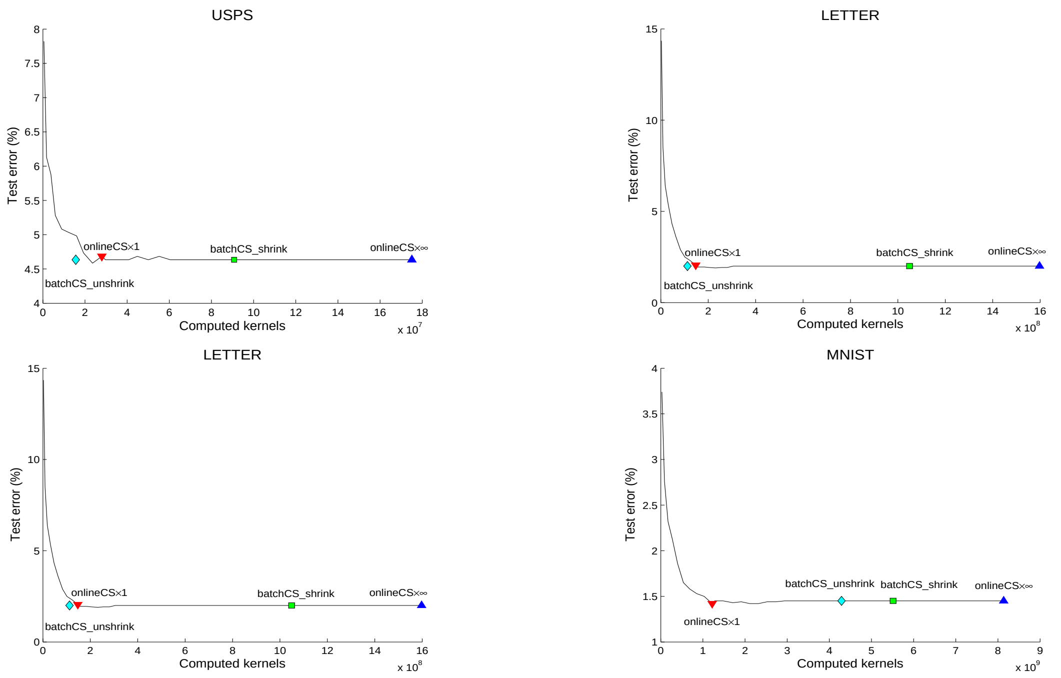


Figure 5.2: Test error as a function of computed kernel

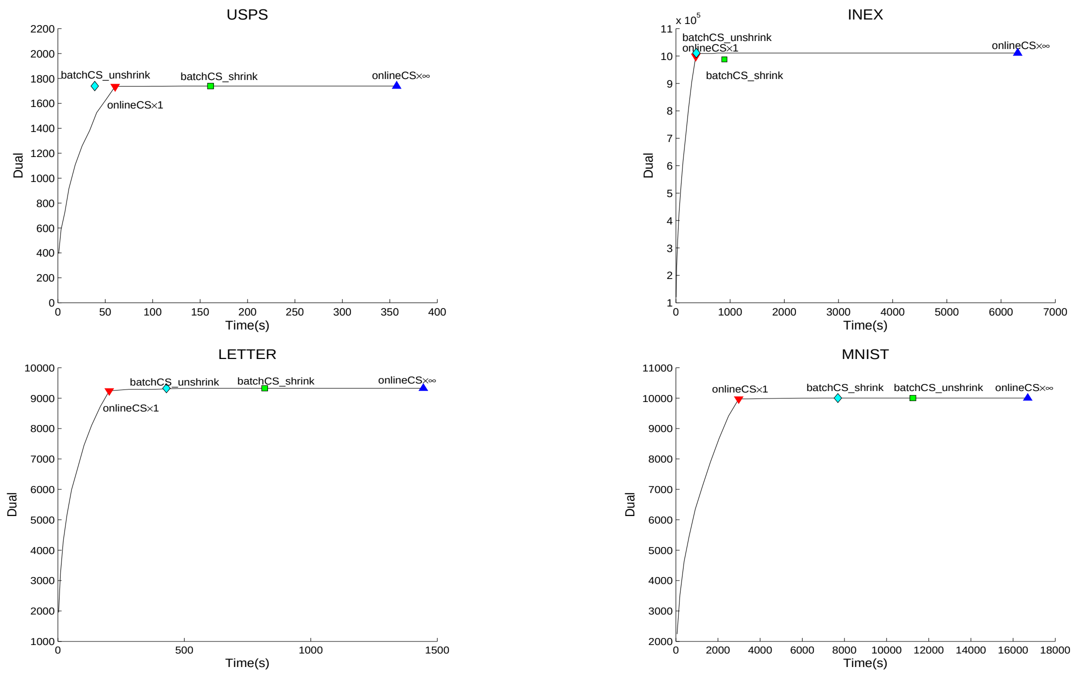


Figure 5.3: Dual as a function of time

5.3.2 Comparison of working selection strategies

As can be seen from the table 5.3, although S2DO_ALL achieves highest dual values and lowest test error among working set strategies, RANDOM_PATTERN_S2DO_ALL to be the best to gain a trade-off between optimization time and reaching a sufficiently high dual value. This can be explained by the optimization structure of CS SVM in dual form. Some of the inequality constraints 5.5 will be active to become equality constraints, which will restrict the two working variables corresponding to the same support pattern. In addition, some of them are still inactive, which leads to the fact that the two working variables might come from different support patterns. As a result, in one side, RANDOM_PATTERN_S2DO_ALL saves time when it restricts one of the two working variables to the set of variables that correspond to a randomly support pattern. In other side, RANDOM_PATTERN_S2DO_ALL still has high possibility of finding the two working variables that have fairly good gain by searching the gain maximization over current set of support patterns. RANDOM_PATTERN_S2DO_PATTERN seems to be the worst working set selection strategy.

		USPS	INEX	LETTER	MNIST
S2DO_ALL	Test error (%)	4.6338	26.0984	**1.9**	1.43
	Dual	**1736.51**	**100738**0	**9271.38**	**9975.35**
	Training time (sec)	87.5257	442.647	515.859	3775.38
	Kernel evaluation ($\times 10^6$)	**26.73**	**32.74**	**140.46**	**1206.77**
RANDOM_PATTERN_ S2DO_ALL	Test errror (%)	4.6338	**26.0489**	**1.9**	**1.42**
	Dual	1734.42	997797	9231.64	9972.75
	Training time (sec)	**58.8231**	**355.564**	241.13	**3260.99**
	Kernel evaluation ($\times 10^6$)	28.01	33.32	147.37	1219.91
RANDOM_PATTERN_ S2DO_PATTERN	Test errror (%)	**4.584**	26.181	2.1	1.47
	Dual	1673.85	971815	8816.29	9919.89
	Training time (sec)	77.9901	**355.268**	**186.703**	3834.19
	Kernel evaluation ($\times 10^6$)	41.78	35.22	206.12	1884.26

Table 5.3: Comparing working set strategies in CS model

Chapter 6

Conclusion and Future Work

This thesis has employed online learning heuristic to derive online learning algorithms for training problems of all-in-one multi-class SVMs. In particular, an online solver for a machine proposed by Lee, Lin and Wahba (LLW) with the preferable theoretical property of Fisher consistency has been developed. Furthermore, we have extended the technique in LLW SVM case to propose an online solver for the unified framework of all all-in-one approaches. To show the efficiency of our online solvers, experiments have been carried out on a benchmark suite of pattern recognition problems in comparison with batch learning. The experiment results have shown that our online solvers are efficient and flexible to approximate the solution of batch learning problems for multi-class SVMs. In detail, our online solvers achieve test errors that are competitive with their counterpart batch solvers, but they consume significant less training time and computation. It is especial true in case of cache limitation when the dual increasing curves of online learning are clearly higher than those of batch learning for all experimented datasets in LLW SVM. This suggest that online learning will be further dominant over batch learning when dataset cannot fit memory, which need further careful research. Another direction for future work is to limit the growth of number of support vector during training that is proposed in [12]. This might limit the number of support patterns which might reduce the training time and memory storage.

Bibliography

[1] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68, 1950.

[2] P.L. Bartlett, M.I. Jordan, and J.D. McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

[3] A. Bordes. *New Algorithms for Large-Scale Support Vector Machines*. PhD thesis, Pierre and Marie Curie University, 2010.

[4] A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In *Proceedings of the 24th international conference on Machine learning*, pages 89–96. ACM, 2007.

[5] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *The Journal of Machine Learning Research*, 6:1579–1619, 2005.

[6] A. Bordes, N. Usunier, and L. Bottou. Sequence labelling svms trained in one pass. In W. Daelemans, B. Goethals, and K. Morik, editors, *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 5211 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 2008.

[7] L. Bottou and Y. LeCun. Large scale online learning. In S. Thrun, L. K. Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2003.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[9] V. R. Carvalho and W. W. Cohen. Single-pass online learning: performance, voting schemes and online feature selection. In T. Eliassi-Rad, L. H. Ungar, M. Craven, and D. Gunopulos, editors, *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery*, pages 548–553. ACM, 2006.

[10] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[11] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, Cambridge, U.K., 2000.

[12] O. Dekel and Y. Singer. Support vector machines on a budget. In B. Schölkopf, J. C. Platt, and Thomas Hoffman, editors, *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems*, pages 345–352. MIT Press, 2006.

[13] Ü. Dogan, T. Glasmachers, and C. Igel. Fast training of multi-class support vector machines. Technical report, Department of Computer Science, University of Copenhagen, 2011.

[14] Ü. Dogan, T. Glasmachers, and C. Igel. A unified view on multi-class support vector classification. In preparation, 2012.

[15] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[16] R. E. Fan, P. Chen, and C. J. Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[17] T. Glasmachers and C. Igel. Maximum-gain working set selection for SVMs. *Journal of Machine Learning Research*, 7:1437–1466, 2006.

[18] T. Glasmachers and C. Igel. Second-order SMO improves SVM online and active learning. *Neural Computation*, 20(2):374–382, 2008.

[19] C. Igel, V. Heidrich-Meisner, and T. Glasmachers. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.

[20] T. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies. In T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes, and R. Zimmer, editors, *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158. AAAI, 1999.

[21] G. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95, 1971.

[22] Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, pages 67–81, 2004.

[23] Y. Lin. A note on margin-based loss functions in classification. *Statistics & probability letters*, 68(1):73–82, 2004.

[24] Y. Liu. Fisher consistency of multicategory support vector machines. In *Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*, pages 289–296, 2007.

[25] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London*, 1909.

[26] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

[27] F. Rosenblatt. *Principles of Neurodynamics: Perceptron and Theory of Brain Mechanisms*. Spartan Books, 1962.

[28] B. Schölkopf, R. Herbrich, and Alex J. Smola. A generalized representer theorem. In *Proceedings of the 14th Annual Conference on Computational Learning Theory and and 5th European Conference on Computational Learning Theory*, COLT '01/EuroCOLT '01, pages 416–426, London, UK, UK, 2001. Springer-Verlag.

[29] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In Z. Ghahramani, editor, *Proceedings of the 24th international conference on Machine learning*, volume 227 of *ACM International Conference Proceeding Series*, pages 807–814. ACM, 2007.

[30] B. Sriperumbudur, K. Fukumizu, and G. Lanckriet. Universality, characteristic kernels and rkhs embedding of measures. *Journal of Machine Learning Research*, 12:2389–2410, 2011.

[31] A. Tewari and P.L. Bartlett. On the consistency of multiclass classification methods. *The Journal of Machine Learning Research*, 8:1007–1025, 2007.

[32] M. Tuma and C. Igel. Improved working set selection for LaRank. In *Computer Analysis of Images and Patterns*, volume 6854 of *Lecture Notes in Computer Science*, pages 327–334. Springer, 2011.

[33] U. von Luxburg and B. Schölkopf. Statistical learning theory: Models, concepts, and results. In D. M. Gabbay and J. Woods, editors, *Inductive Logic*, volume 10 of *Handbook of the History of Logic*, pages 651–706. North-Holland, 2011.

[34] J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the Seventh European Symposium On Artificial Neural Networks*, pages 219–224, 1999.

[35] T. Zhang. Statistical analysis of some multi-category large margin classification methods. *The Journal of Machine Learning Research*, 5:1225–1251, 2004.

[36] T. Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *Annals of Statistics*, pages 56–85, 2004.