SVMs

Callum Lau, 19102521

Numerical Optimisation

May 6, 2020

Contents

1 Mathematical derivation

In this preliminary section we derive the SVM problem. These derivations largely follow the presentations in [1] as well as the lecture notes in [2].

1.1 Primal Form for Hard-Margin Binary Classification

The support vector machine setup is as follows.

- Suppose we have a set of m training points $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m \in \mathbb{R}^n \times \{-1, 1\}$.
- Denote the **hyperplane** $H_{\boldsymbol{w},b} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\}$ where $\boldsymbol{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. In the **hard-margin** case we assume that the set $\mathcal{S}$ is linearly separable, i.e. $\exists \boldsymbol{w}, b$ such that

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) > 0, \qquad i = 1, \dots, m \tag{1.1}$$

  then $H_{\boldsymbol{w},b}$ is called a **separating** hyperplane. This means the vectors with the same label y lie on the same side of the hyperplane.

- Let the distance of a point $\boldsymbol{x}$ from the hyperplane $H_{\boldsymbol{w},b}$ be $\rho_{\boldsymbol{x}}(\boldsymbol{w}, b)$. Then

$$\rho_{\boldsymbol{x}}(\boldsymbol{w}, b) = \frac{|\boldsymbol{w}^\top \boldsymbol{x} + b|}{\|\boldsymbol{w}\|} \tag{1.2}$$

- If $H_{\boldsymbol{w},b}$ separates $\mathcal{S}$ we define the **margin** as

$$\rho_{\mathcal{S}}(\boldsymbol{w}, b) = \min_{1 \le i \le m} \rho_{\boldsymbol{x}_i}(\boldsymbol{w}, b) \tag{1.3}$$

  this is the distance between the hyperplane and the closest vector $\boldsymbol{x}_i$.

The optimisation problem is to find the **optimal** separating hyperplane, which corresponds to the hyperplane with maximum margin. This hyperplane $H_{\boldsymbol{w},b}$ therefore solves the problem:

$$\rho(\mathcal{S}) \coloneqq \max_{\boldsymbol{w},b} \min_{1 \le i \le m} \left\{ \frac{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|} \ : \ y_j(\boldsymbol{w}^\top \boldsymbol{x}_j + b) > 0,\, j = 1, \dots, m \right\}$$

Note that the choice of $(\boldsymbol{w}, b)$ is not unique up to scaling. We can therefore require that the hyperplane is normalised, i.e. $\|\boldsymbol{w}\| = 1$. However a more convenient choice is the data-dependent parametrisation - choose $\|\boldsymbol{w}\|$ such that $\rho_{\mathcal{S}}(\boldsymbol{w}, b) = \frac{1}{\boldsymbol{w}}$, in which case we require

$$\min_{1 \le i \le m} y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) = 1$$

The optimisation problem now becomes

$$\begin{aligned}
\rho(\mathcal{S}) &= \max_{\boldsymbol{w},b} \left\{ \frac{1}{\|\boldsymbol{w}\|} : \min_{1 \le i \le m} y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) = 1,\, y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + n) \ge 0 \right\} \\
&= \max_{\boldsymbol{w},b} \left\{ \frac{1}{\|\boldsymbol{w}\|} : y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \ge 1 \right\} \\
&= \frac{1}{\min_{\boldsymbol{w},b}\{\|\boldsymbol{w}\| : y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \ge 1\}}
\end{aligned}$$

which is equivalent to

Primal Form optimisation problem

$$\begin{aligned}
\min_{\boldsymbol{w}, b \in \mathcal{D} \subset \mathbb{R}^n \times \mathbb{R}} \quad & \frac{1}{2} \boldsymbol{w}^\top \boldsymbol{w} \\
\text{subject to} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \ge 1, \quad i = 1, \dots, m
\end{aligned} \tag{1.4}$$

This is the **primal** form of a convex optimisation problem with a quadratic critefrion and linear inequality constraints.

1.2 Dual Form for Hard-Margin Binary Classification

In order to derive the dual form of this problem we first write down the Lagrangian $\mathcal{L} : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}$:

$$\mathcal{L}(\boldsymbol{w}, b, \lambda) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{m} \lambda_i [y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1] \tag{1.5}$$

with Lagrange multipliers $\lambda \succeq 0$. The Lagrange dual function $g : \mathbb{R}^m \to \mathbb{R}$ is then defined as

$$g(\lambda) \coloneqq \inf_{\boldsymbol{w}, b} \mathcal{L}(\boldsymbol{w}, b, \lambda) \tag{1.6}$$

This can be found by differentiating with respect to $\boldsymbol{w}$

$$\frac{\partial}{\partial \boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \lambda) = \boldsymbol{w} - \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{x}_i = 0 \implies \boldsymbol{w} = \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{x}_i$$

and differentiating with respect to b

$$\frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{w}, b, \lambda) = \sum_{i=1}^{m} \lambda_i y_i = 0$$

Substituting these equalities back into 1.6 we get:

$$g(\lambda) = \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \lambda_i \lambda_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \tag{1.7}$$

Note that the Lagrangian dual function provides a lower bound on the solution p^* to (1.4), i.e. $p^* \geq g(\lambda)$. By maximising this lower bound $\max_\lambda g(\lambda) = d^*$ we form the dual form for the problem:

Dual Form optimisation problem

$$\begin{aligned} \max_{\lambda \in \mathbb{R}^m} \quad & g(\lambda) = \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \lambda_i \lambda_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\ \text{subject to} \quad & \lambda_i \geq 0 \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \lambda_i y_i = 0 \end{aligned} \tag{1.8}$$

1.3 The Non-Separable case

In the hard-margin case we assumed that the data were linearly separable. However, in even the high dimensional case this is not always possible. In the **soft-margin** case we add regularisation to permit training examples to have functional margin less than one, relaxing the separation constraints in (1.4). To this end we introduce the loss functional $L(y_i, \boldsymbol{w}^\top \boldsymbol{x}_i + b)$. The optimisation problem now becomes:

$$\min_{\boldsymbol{w}, b} \quad \frac{1}{2} \boldsymbol{w}^\top \boldsymbol{w} + C \sum_{i=1}^{m} L(y_i, \boldsymbol{w}^\top \boldsymbol{x}_i + b) \tag{1.9}$$

In general this problem is **non-convex**. A natural choice of loss functional is the misclassification loss $L(y, \hat{y}) = \mathcal{I}(y = \hat{y})$ which simply counts the number of misclassifications. However, this is NP-hard [5] and so instead hinge-loss is used: $L(y, \hat{y}) = \max(0, 1 - y\hat{y})$. This yields a convex problem with a unique solution. By introducing the **slack** variables $\xi_i = L_{\text{hinge}}(y_i, \boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0$ we can state the convex problem in primal form as

Primal Form Soft-Margin problem

$$\begin{aligned} \min_{\boldsymbol{w}, b \in \mathcal{D} \subset \mathbb{R}^n \times \mathbb{R}} \quad & \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C \sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \qquad\qquad i = 1, \ldots, m \end{aligned} \tag{1.10}$$

The parameter $C \in \mathbb{R}$ controls the relative weighting between the $\|\boldsymbol{w}^2\|$ and $\sum_{i=1}^{m} \xi_i$. We can derive the dual form of this problem in a similar fashion to Section 1.2. Again, we form the Lagrangian

$$\mathcal{L}(\boldsymbol{w}, b, \xi, \lambda, \mu) = \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \lambda_i [y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) + \xi_i - 1] - \sum_{i=1}^{m} \mu_i \xi_i \tag{1.11}$$

with Lagrange multipliers $\lambda, \mu \succeq 0$. The Lagrange dual function $g(\lambda, \mu)$ is just the infimum of $\mathcal{L}$ minimising over the parameters $(\boldsymbol{w}, b, \xi)$, and the dual solution is the supremum of g maximising over the multipliers λ, μ. As before, we differentiate with respect to $(\boldsymbol{w}, b, \xi)$:

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{m} \lambda_i y_i = 0 \tag{1.12}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{x}_i = 0 \implies \boldsymbol{w} = \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{x}_i \tag{1.13}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \lambda_i - \mu_i = 0 \implies 0 \leq \lambda_i \leq C \tag{1.14}$$

Substituting these back into (1.11) leads to the soft-margin dual form:

Dual Form Soft-Margin problem

$$\begin{aligned} \min_{\lambda \in \mathbb{R}^m} \quad & g(\lambda) = \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \lambda_i \lambda_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i=1}^{m} \lambda_i \\ \text{subject to} \quad & 0 \leq \lambda_i \leq C \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \lambda_i y_i = 0 \end{aligned} \tag{1.15}$$

1.4 Equivalence between primal and dual problems

The weak duality condition: $d^* \leq p^*$ always holds. However the strong duality statement $d^* = p^*$ only holds under constraint qualification - namely if Slater's condition is satisfied. Slater's condition states that strong duality holds for a convex program if there exists a x^* such that x^* is *strictly feasible* for the primal problem, i.e. the equality constraints are satisfied and the inequality constraints are satisfied with strict inequality. This holds for the SVM (and general Quadratic Programming) problem, and so solving the dual is equivalent to solving the primal.

Furthermore, if strong duality holds the Karush-Kuhn-Tucker (KKT) are **necessary** and **sufficient** conditions for x^* to be an optimum. These provide helpful conditions for solving the optimisation problem and are stated below in the SVM context for $i = 1, \ldots, m$:

- Primal Feasibility

$$1 - \xi_i - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \leq 0 \tag{1.16}$$
$$-\xi_i \leq 0 \tag{1.17}$$

- Dual Feasibility

$$\lambda_i \geq 0 \tag{1.18}$$

- Complementary Slackness

$$\lambda_i(1 - \xi_i - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)) = 0 \tag{1.19}$$
$$\mu_i \xi_i = 0 \tag{1.20}$$

- Zero Derivatives (same as (1.12) - (1.14))

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{m} \lambda_i y_i = 0$$
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{x}_i = 0$$
$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \lambda_i - \mu_i = 0$$

1.5 The bias and classifier

In the discussions above we have so far neglected to talk about how the bias term b is calculated. This term can be derived from the KKT conditions. If $0 < \lambda_i < C$ then the complementary slackness condition (1.19) implies $1 - \xi_i - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) = 0$. Multiplying by $y_i = \pm 1$ and rearranging gives

$$b = y_i(1 - \xi_i) - \boldsymbol{w}^\top \boldsymbol{x}_i$$

Now if $0 < \lambda_i < C$ then the zero-derivative condition (1.14) implies $\mu_i = C - \lambda_i \neq 0$. Hence from the second complementary slackness condition (1.20) $\mu_i \xi_i = 0$ we get that $\xi_i = 0$. Then the equation can be simplified to

$$b = y_i - \boldsymbol{w}^\top \boldsymbol{x}_i$$

We could in theory pick any i to calculate the bias term, but it is sensible to take the average using all indices i which satisfy $0 < \lambda_i < C$. In fact all such terms $\boldsymbol{x}_i$ are called **non-margin support vectors**. Denote this set $\mathcal{S} = \{\boldsymbol{x}_i : 0 < \lambda_i < C\}$, in which case the bias term is now calculated by:

$$b = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (y_s - \sum_{i=1}^{m} \lambda_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x}_s \rangle)) \tag{1.21}$$

where we have used the zero-derivative condition (1.13): $\boldsymbol{w} = \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{x}_i$. Now suppose we are given a new point $\boldsymbol{x}_{\text{test}}$. We can classify this point with the label $\hat{y}$ by which side of the hyperplane it lies on, i.e. the classifier takes the form

$$\hat{y} = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_{\text{test}} + b) \tag{1.22}$$

1.6 Kernels

If we wish to better separate the non-separable training data $\boldsymbol{x}_i$ it may be helpful to apply a feature map $\phi : \mathbb{R}^n \to H, \boldsymbol{x} \to \phi(\boldsymbol{x})$ and perform classification in a higher dimensional Hilbert space H. Note that optimisation problem (1.15) involves $\boldsymbol{x}_i$ solely in terms of the inner product $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. Let $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be the positive semi-definite kernel such that $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$. By applying the feature maps we can therefore solve the kernelised version of the Dual Form Soft-Margin problem:

Kernelized Dual Form Soft-Margin problem

$$\begin{aligned} \min_{\lambda \in \mathbb{R}^m} \quad & g(\lambda) = \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \lambda_i \lambda_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{m} \lambda_i \\ \text{subject to} \quad & 0 \leq \lambda_i \leq C \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \lambda_i y_i = 0 \end{aligned} \tag{1.23}$$

By the *kernel trick* we can often compute the inner product $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ without having to explicitly form the (potenially infinite sized) feature maps ϕ.

1.7 Multi-class Classification

There exists many techniques for extending the binary SVM to handle multiclass data. The three main methods are One-Versus-All (OVA), One-Versus-One (OVO) and Directed Acyclic Graph SVM (DAGSVM) [3]. The OVA method constructs K binary SVMs if there are K classes in the data. The ith SVM is trained on all examples in class i given positive label $y_i = +1$ against all examples not in class i given negative label $y_i = -1$. Hence each binary SVM constructs a hyperplane between its class and all the others combined. Given K trained SVMs each with support vectors and biases $\boldsymbol{\lambda}^{(k)}, b^{(k)}, k = 1, \ldots, K$, the classification of a test point $\boldsymbol{x}_{\text{test}}$ is given by

$$\hat{y}_{\text{test}} = \underset{1 \leq k \leq K}{\operatorname{argmax}} \left(\sum_{i=1}^{m} y_i \lambda_i^{(k)} k(\boldsymbol{x}_i, \boldsymbol{x}_{\text{test}}) + b^{(k)} \right) \tag{1.24}$$

i.e. it is the label of the SVM whose classifier which produces the greatest value of the decision function.

Another framework for multiclass classification is the OVO method. The OVO method constructs an SVM classifier trained on data in every combination of the two classes $i, j \in \{1, \ldots, K\}, i \neq j$ so that we have $K(K-1)/2$ binary classifiers. After training all classifiers the test point $\boldsymbol{x}_{\text{test}}$ is classified via a voting strategy. Given each trained classifier $\text{SVM}^{(ij)}$ between classes i and j with support vectors $\lambda^{(ij)}$ and bias $b^{(ij)}$, then if $\text{sign}(\sum_{l=1}^{m} y_l \lambda_l^{(ij)} k(\boldsymbol{x}_l, \boldsymbol{x}_{\text{test}}) + b^{(ij)})$ says $\boldsymbol{x}_{\text{test}}$ is in class i then add one to the votes of class i, otherwise add one to the votes of class j. We perform this for all classifiers, and the final classification is given by the class with the most votes. The DAGSVM method is very similar. It first trains $K(K-1)/2$ binary SVMs, but the testing procedure is given by a directed acyclic graph with $K(K-1)/2$ internal nodes (corresponding to each SVM) and K leaves. The classification of a test point is given by the label of the leaf after traversing the binary classifiers in the DAG.

2 Study

2.1 The Historical Credit Rating Dataset

The dataset we will use for our experiments is the 'Historical Credit Rating' dataset available directly from the MATLAB client. The dataset consists of 3932 observations each with 6 predictors, and one response variable belonging to one of 7 classes deemed the credit rating of a corporate institution - 'AAA', 'AA', 'A', 'BBB', 'BB', 'B' or 'CCC':

The Historical Credit Rating Dataset							
Credit Rating	AAA	AA	A	BBB	BB	B	CCC
number of samples	580	385	575	1015	927	320	130

The task then is to learn a classifier to accurately predict the credit rating of an observation given its predictors. In the binary classification case we can simply pick the observations and labels of two classes and perform classification on just those two. In the interest of challenge, we choose the data belonging to credit ratings 'BBB' and 'BB', which it is fair to assume will be significantly more similar in nature than 'AAA' and 'C' for example. This resulted in a smaller dataset of 1942 observations with 1015 observations belonging to 'BBB' and 927 belonging to 'BB'.

2.2 Potential Issues

1. Although we have significantly trimmed down the size of the dataset, a dataset with 1942 observations is still moderately large. To this end it would be interesting to examine the performance of two algorithms: one solver not designed for larger datasets and another much more capable of scaling to a large number of datapoints. If we consider multiclass classification and subsequently even more datapoints this problem becomes exponentiated.

2. Another challenge is the small dimensionality of the data. With just 6 predictors it may be difficult to use a linear classifier to separate the data. Therefore we propose to use kernels to map the data to a higher dimensional space and perform classification there.

3. For the multiclass case there exists a mild class imbalance. The largest class BBB has 1015 observations whilst the smallest (CCC) has just 130. This may affect the accuracy of a multiclass algorithm.

These issues motivate our choice of the **Log Barrier** and **Sequential Minimal Optimisation** (SMO) algorithms. The Log Barrier method is an **Interior Point** solver for general Quadratic Programming (QP) problems with both equality and inequality constraints. It is a Newton-type algorithm that in its simplest form relies on the inversian of the objective function's Hessian, which scales with the number of data points. Conversely SMO is an algorithm developed specifically for SVMs to tackle the problem of dealing with large datasets, and therefore should be much faster. In our analysis both algorithms will operate on the kernelized Dual Form Soft-Margin optimisation problem (1.23). Succinct derivations of the algorithms are given in the next section. Although the kernelisation introduces non-linearity in the data/classifier, the dual form is still expressed as a QP, and hence the resulting problem is

convex, **constrained**, **smooth** and **quadratic**

The study will therefore explore the performance of both algorithms with respect to two different kernels: the **Polynomial** and **Gaussian** kernel.

$$\begin{aligned} \text{Polynomial:} \qquad & k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle^p \\ \text{Gaussian:} \qquad & k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{\sigma^2} \right) \end{aligned}$$

As an extension, we will extend the SVM to the multiclass case, and analyse the effectivess of each kernel using the One-Vs-All method.

2.3 Metrics

For our experiments we will consider **four** metrics. Purely in terms of the optimisation problem, we wish to minimise the objective function $g(\lambda)$ subject to the constraints in (1.23). Hence we will firstly evaluate the performance of the two algorithms in minimising the objective function (with fixed value of C). However of course the original goal is to correctly separate the dataset so that the largest amount of labelled examples

lie on the correct side of the (non-linear) hyperplane. Therefore the second metric will be **training error** (number of misclassified examples in the training set). Ultimately we wish to generalise the SVMs performance to new unseen datapoints, which suggests the third metric of **testing error** (number of misclassified examples in the test set). Finally, we will consider the **computational performance** of the algorithms, namely convergence rates and CPU times.

3 Algorithms

3.1 Log Barrier

The Log Barrier algorithm [7] is an interior point method for convex optimisation problems with equality and inequality constraints. The general form of such a problem is

$$\begin{aligned} \min_x \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0 \quad i = 1, \ldots, m \\ & Ax = b \end{aligned} \tag{3.1}$$

where $f_0, f_1, \ldots, f_m$ are convex and twice differentiable. This can be rewritten by making the inequality constraints implicit in the objective

$$\begin{aligned} \min_x \quad & f_0(x) + \sum_{i=1}^{m} \mathcal{I}_-(f_i(x)) \\ \text{subject to} \quad & Ax = b \end{aligned} \tag{3.2}$$

where $\mathcal{I}_-(f_i(x))$ is an indicator function for the non positive reals which is 0 if $f_i \leq 0$ and infinite if $f_i > 0$. However, this objective is not differentiable and therefore we cannot employ standard gradient type algorithms such as Newton's method. Therefore replace $\sum_{i=1}^{m} \mathcal{I}_-(f_i(x))$ by a (convex) differentiable approximation $\phi(x) = -\sum_{i=1}^{m} \frac{1}{t} \log(-f_i(x))$. Hence (3.2) becomes

$$\begin{aligned} \min_x \quad & \phi(x) \equiv f_0(x) - \sum_{i=1}^{m} \frac{1}{t} \log(-f_i(x)) \\ \text{subject to} \quad & Ax = b \end{aligned} \tag{3.3}$$

An equality constrained (feasible) Newton's method can then be used to solve (3.3). Through linearisation of the optimality conditions, the feasible Newton method involves the solution of the following KKT system:

$$\begin{bmatrix} \nabla^2 \phi(x) & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_n \\ \nu \end{bmatrix} = \begin{bmatrix} -\nabla \phi(x) \\ 0 \end{bmatrix} \tag{3.4}$$

where Δx_n is a feasible descent direction and ν is the vector of Langrange multipliers for the equality constraint. Subsequently backtracking linesearch can be used to find a step length α that is optimal and satisfies the inequality constraints. Hence after one Newton step we have $x \leftarrow x + \alpha \Delta x_n$. This is feasible since $A(x + \alpha \Delta x_n) = Ax + \alpha A \Delta x_n = Ax + 0 = b$. Now define the Newton decrement $\lambda(x)$

$$\lambda(x) \equiv (\Delta x_n^\top \nabla^2 \phi(x) \Delta x_n)^{\frac{1}{2}} = \nabla \phi(x)^\top \Delta x_n \tag{3.5}$$

The value $\lambda^2(x)/2$ gives a good approximation of the difference between $\phi(x)$ and the true minimum of the quadratic model around x. It is therefore useful as a stopping condition $\lambda^2(x)/2 \leq \epsilon$ of the Newton iterations. The convergence properties of the feasible Newton method are exactly the same as applying Newton's method to the unconstrained problem.

Now back to problem (3.3) - it is clear that as t grows the quality of the log barrier approximation increases, in which case we would like to solve this system with as large a value of t as possible. However for larger t the objective function is more difficult to minimise using Newton's since the Hessian will vary

rapidly near the boundary of the feasible set [6]. The Log Barrier method therefore opts to solve a sequence of unconstrained, or linearly constrained, subproblems of the form (3.3), which involves increasing the value of t by a factor μ and successively Newton minimising starting from the solution for the previous value of t. The solutions $x^*(t)$ for each value of t, called central points, define a **central path**.

An important property is that every central point produces a dual feasible pair $\lambda^*(t), \nu^*(t)$ which minimises the Lagrangian

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \nu^\top (Ax - b)$$

Consequently, the *duality gap* between $f_0(x^*(t))$ and p^* (the primal optimal solution) satisfies

$$f_0(x^*(t)) - p^* \leq m/t \tag{3.6}$$

i.e. it converges to the optimal solution as $t \to \infty$. This suggests the stopping criteria to end the iterations if $m/t \leq \epsilon$ for some given tolerance ϵ.

Now, the dual form of the soft margin SVM problem is simply a Quadratic Program (QP) and therefore a special case of (3.1). If we define

$$\begin{aligned}
f_0(\lambda) &\equiv \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \lambda_i \lambda_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{m} \lambda_i \\
f_i(\lambda) &\equiv -\lambda_i \qquad\qquad i = 1, \ldots, m \\
f_{m+i}(\lambda) &\equiv \lambda_i - C \qquad\qquad i = 1, \ldots, m \\
A &\equiv \boldsymbol{y}_m^T \\
b &\equiv 0
\end{aligned}$$

where $\boldsymbol{y}$ is the m-dimensional stacked vector of class labels y_i, then the SVM problem is now in the form of (3.1). Additionally note that Barrier methods require a strictly feasible starting point $x^{(0)}$. When this is not known the method is preceeded by a phase I stage to find one. However, for the SVM case the equality and inequality constraints are of a particularly simple form as to make the phase I stage particularly simple. Our starting point $\lambda^{(0)}$ can be found by simply counting the number n of positive labels $y_i = 1$. Then assign all $2n$ multipliers associated with the positive labels with the value $C/2n$, and all $2(m-n)$ multipliers associated with the negative lables with the value $C/2(m-n)$. This ensures the equality constraint is satisfied and the multipliers are within the inequality bounds.

Algorithm 1 Log Barrier

initialise $t \leftarrow t^{(0)}$, μ, $\epsilon, \epsilon', \lambda(x)$
find feasible starting point $x \leftarrow x^{(0)}$
while $\frac{m}{t} \geq \epsilon$ **do**
    solve subproblem (3.3) for fixed t:
    while $\frac{\lambda^2(x)}{2} \geq \epsilon'$ **do**
        compute Newton step Δx_n by (3.4) and decrement $\lambda^2(x)$ by (3.5)
        choose step size α via *backtracking linesearch*
        update $x \leftarrow x + \alpha \Delta x_n$
    end while
    improve approximation:
    update $t \leftarrow \mu t$
end while

3.2 Sequential Minimal Optimisation

Training a support vector machine can involve solving a very large quadratic programming optimisation problem. The approach taken by the Log Barrier IP method above does not scale well since it directly solves

the QP problem that scales cubically with the number of examples. The Sequential Minimal Optimisation [4] (SMO) algorithm breaks the large QP problem into a series of smaller QP subproblems which minimises the dual soft-margin SVM problem (1.23), by choosing a working set of two λ_i's to optimise during an iteration and keeping the others fixed. We cannot optimise with respect to a single λ_i because of the equality constraint $\sum_{i=1}^m \lambda_i y_i = 0$. A brief derivation of the algorithm is now given.

Suppose that the two multipliers we wish to optimise are λ_1^{old} and λ_2^{old}. Since the other λ's are fixed, they are constant with respect to the minimisation and so the problem can be written as (ignoring all constant terms w.r.t the minimisation):

$$\begin{aligned} \min_{\lambda_1,\lambda_2} \quad & g(\lambda_1,\lambda_2) = \frac{1}{2}K_{11}\lambda_1^2 + \frac{1}{2}K_{22}\lambda_2^2 + sK_{12}\lambda_1\lambda_2 + y_1\alpha_1 v_1 + y_2\alpha_2 v_2 - \lambda_1 - \lambda_2 \\ \text{s.th} \quad & 0 \le \lambda_1 \le C \\ & 0 \le \lambda_2 \le C \\ & \lambda_1 y_1 + \lambda_2 y_2 = \lambda_1^{\text{old}} y_1 + \lambda_2^{\text{old}} y_2 \end{aligned}$$

where $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $s \equiv y_1 y_2$ and $v_j \equiv \sum_{i=3}^m \lambda_i y_i K_{ij}$. Multiplying by y_1, the equality constraint can be expressed as

$$\lambda_1 + s\lambda_2 = \lambda_1^{\text{old}} + s\lambda_2^{\text{old}} \equiv w$$

Hence the objective function $g(\lambda_1, \lambda_2)$ can be expressed as a function of λ_2 only:

$$\begin{aligned} g(\lambda_2) = \frac{1}{2}K_{11}(w - s\lambda_2)^2 + \frac{1}{2}K_{22}\lambda_2^2 + sK_{12}(w - s\lambda_2)\lambda_2 \\ + y_1(w - s\lambda_2)v_1 - w + s\lambda_2 + y_2\lambda_2 v_2 - \lambda_2 \end{aligned}$$

The minimum can be found via setting the differential to zero, which yields (after a lot of algebraic manipulation):

$$\lambda_2^{\text{new}} = \lambda_2^{\text{old}} + \frac{y_2(E_1 - E_2)}{\eta} \tag{3.7}$$

where $\eta = K_{11} + K_{22} - 2K_{12}$ and $E_j = \hat{y}_j - y_j$ (the error on example j). This is indeed a minimum since the g is convex. As a result of the inequality constraints λ_2^{new} possibly may not be feasible, and instead it must be clipped. If $y_1 = y_2$ then the inequality constraint implies

$$L = \max(0, \lambda_2 - \lambda_1) \le \lambda_2 \le \min(C, C + \lambda_2 - \lambda_1) = H$$

else when $y_1 \neq y_2$

$$L = \max(0, \lambda_2 + \lambda_1 - C) \le \lambda_2 \le \min(C, \lambda_2 + \lambda_1) = H$$

The minimiser $\lambda_2^{\text{new, clipped}}$ can then be selected according to the bounds $[L, H]$ if it lies outside the feasible region. Finally, the value of α_1 can then be determined by the formula:

$$\lambda_1^{\text{new}} = \lambda_1^{\text{old}} + s(\lambda_2^{\text{old}} - \lambda_2^{\text{new, clipped}}) \tag{3.8}$$

Of course, in the analysis above we have assumed that a suitable working pair λ_1, λ_2 has been chosen to performe the minimisation. Choosing the pair significantly affects the performance of the algorithm. However the choice is largely down to heuristics. The most common is the *maximal violating pair* heuristic. This aims to choose the Lagrange multiplier which will increase the minimisation step taken the most. However, this is expensive to calculate and so by proxy the multiplier which produces the largest value of $|E_1 - E_2|$ is chosen. The indices i, j of the multipliers to be updated are therefore:

$$i \in \underset{t \in I_{\text{up}}(\boldsymbol{\lambda}^k)}{\operatorname{argmax}} -y_t \nabla g(\boldsymbol{\lambda}^k)_t, \qquad j \in \underset{t \in I_{\text{low}}(\boldsymbol{\lambda}^k)}{\operatorname{argmin}} -y_t \nabla g(\boldsymbol{\lambda}^k)_t \tag{3.9}$$

where

$$I_{\text{up}}(\boldsymbol{\lambda}) = \{t | \lambda_t < C, y_t = 1 \text{ or } \lambda_t > 0, y_t = -1\}$$
$$I_{\text{low}}(\boldsymbol{\lambda}) = \{t | \lambda_t < C, y_t = -1 \text{ or } \lambda_t > 0, y_t = 1\}$$

A cache of evaluated errors is often employed to reduce kernel evaluations. There are many other heuristics and more sophisticated techniques to improve the performance of the SMO algorithm but they are not explored here.

Algorithm 2 Sequential Minimal Optimisation (simplified)

1: $\lambda_i \leftarrow 0$ for $i = 1, \ldots, m$
2: **while** any λ_i violate KKT conditions **do**
3: choose maximal violating pair indices (1, 2) according to (3.9)
4: update λ_2 according to (3.7), clipping if needed.
5: update λ_1 according to (3.8)
6: **end while**

4 Experiments

- The primary experiments applied the binary kernelized SVM algorithm to separate the data in the Credit Rating dataset belonging to classes 'BBB' and 'BB'. We performed an 80/20 training/test split of the data. Hence we trained on 812 examples from 'BBB' and 741 from 'BB', and tested on the remaining 203 and 186 examples in 'BBB' and 'BB'.

- Before carrying out the experiments we first note there exists a wide range of hyperparameters that could potentially be optimised. For the Barrier method these include the backtracking sufficient decrease and curvature condition parameters $\texttt{c}_1$ and $\texttt{c}_2$, as well as initial log barrier coefficient $t^{(0)}$ and multiplier μ. For both algorithms, we note that the regularisation parameter C is far more important since it controls the tolerance for misclassified examples. Hence a discussion of this parameter in the context of the Polynomial and Gaussian kernels is handled in depth. For now we display the parameters of the Barrier method that were held fixed for the rest of the experiments.

	$t^{(0)}$	μ	α_0	$\texttt{c}_1$	$\texttt{c}_2$	ϵ	ϵ_{newton}
`Barrier`	1	20	1	0.1	0.9	`1e-4`	`1e-3`

- The parameter $t^{(0)}$ controls the trade off between the number of inner iterations in the first Newton step and the outer iterations. The choice of μ is discussed in Section 4.2.1. Changes to the backtracking parameters $\texttt{c}_1$ and $\texttt{c}_2$ generally did not impact the performance of the algorithms and hence the standard values for Newton-type methods were used. The initial step length α_0 was chosen as 1 so that full Newton steps can be taken. The tolerance of the Newton method ϵ_{newton} was set high at `1e-3` since the Barrier method does not require highly accurate centering steps to succeed. Finally the overall tolerance ϵ was set reasonably high since we are more interested in the training/test error rather than a highly accurate solution.

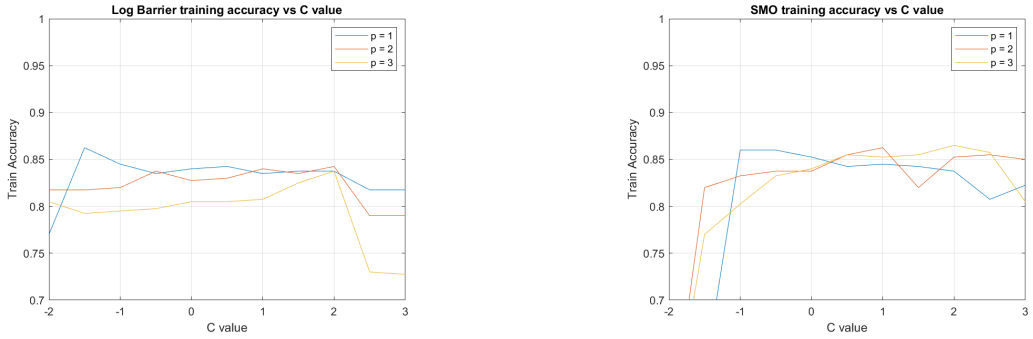
- The SMO algorithm requires almost no hyper-parameters (apart from the regularisation term C). The algorithm terminates when the number of KKT violating multipliers is zero. However, this can sometimes be hard to achieve in practice so we set $\texttt{maxIter} = 3000$.

4.1 Kernels

In this section we seek to examine the performance of both algorithms with respect to the Polynomial and Gaussian Kernels. The discussion keeps in mind the dual objective of low test-error (a machine learning perspective), and low training-error and relative objective function values (an optimisation perspective). Extensive cross-validation is not performed as we are not solely interested in the former persepective. However, we do investigate the performance of the kernels with respect to the regularisation parameter C as well as the kernel parameter p (polynomial degree) or σ (gaussian width).

4.1.1 Polynomial Kernel

Firstly, we explore a suitable choice of the parameter C. For both algorithms, we use *training accuracy* to assess the best value of C. Hence we plot training accuracy as a function of $C = 10^d$ for $d = -2, -1.5, \ldots, 2.5, 3.0$.



(a) Log Barrier (b) SMO

Figure 4.1: Training accuracy for varying C (base 10)

- Both algorithms are somewhat robust against the choice of C. For $C < 10^{-1}$ and $C > 10^2$ the training accuracy generally decreases. However, for values within that range the training accuracy does not differ significantly.
- A value of $C = 10$ seems to be a reasonable choice resulting in high training accuracy over all polynomials and algorithm choice. A cross-validated grid search over a fine grid of possible values could be performed to find an absolutely optimal value of C but this is not explored here.

We now proceed to measure the performance of the polynomial kernels for both algorithms using a regularisation parameter of $C = 10$. The performance indicators are the value of the objective function, the training accuracy and the test accuracy. The results are given in the Table below.

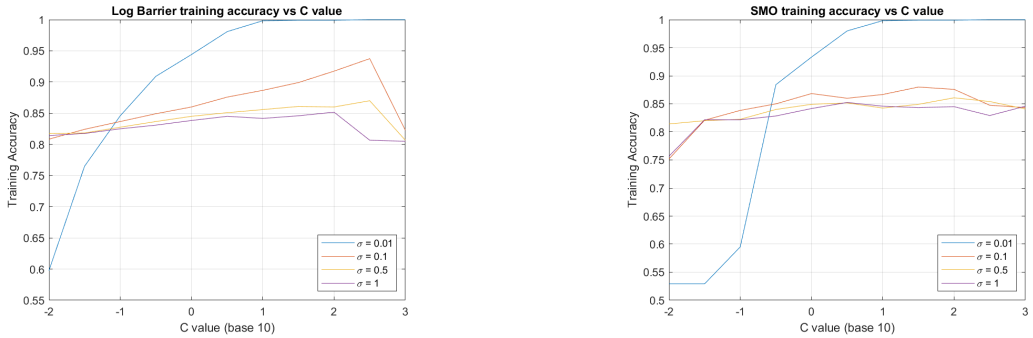
		Performance Indicators			
		Objective	Training Accuracy	Test Accuracy	best
$p = 1$	LogBarrier	**-1.4583e+04**	0.8300	0.8008	
	SMO	-1.3007e+04	0.8142	0.8183	
$p = 2$	LogBarrier	**-1.4391e+04**	**0.8500**	**0.8237**	✓
	SMO	-1.0653e+04	0.8142	0.8116	
$p = 3$	LogBarrier	**-1.4207e+04**	0.8100	0.7981	
	SMO	-1.0212e+04	0.8183	0.7880	

- The Objective function value over all polynomial degrees is lower for the Log Barrier method than SMO. Purely in terms of the minimisation problem (1.23) both algorithms are trying to solve, we can therefore conclude that Log Barrier is significantly more accurate than SMO. This conforms with the expectation that Interior Point methods are highly accurate in comparison to decomposition methods like SMO.
- Log Barrier method with $p = 2$ had the best training and test accuracy. However, there is no clear trend of higher degree kernels outperforming the linear kernel, and the performance of the $p = 2$ kernel is marginally better than the linear case.
- Testing accuracy *decreases* for $p = 3$ polynomial kernels. It is possible that for $p = 3$, the SVM is **overfitting** to the training data. Since the SVM is fitting in a richer feature space, a kernelized SVM is much more likely to overfit.

In conclusion we observe that the Log Barrier method outperforms SMO in terms of training and test accuracy. Furthermore the polynomial degree $p = 2$ produces the best training and test accuracy.

4.1.2 Gaussian Kernel

In this section we investigate the performance of the Gaussian kernel with kernel width $\sigma = 0.01, 0.1, 0.5, 1$. Again, we first explore the effect of the regularisation parameter C on the training accuracy over the same range of values. This is plotted in Figure 4.2.



(a) Log Barrier

(b) SMO

Figure 4.2: Training accuracy for varying C

- For both algorithms the Gaussian kernel with width $\sigma = 0.01$ achieves close to 100% training accuracy for values of C greater than 10.
- The general trend is that training accuracy increases for C in the range $[10^1, 10^{2.5}]$, and that a smaller kernel width σ increases the training accuracy for both algorithms. A sensible choice of C is 10^2, although a more rigourous search could be carried out.

Therefore the objective function values, training accuracy and test accuracy for the value of $C = 100$ over the different kernel widths is provided below.

		Performance Indicators			
		Objective	Training Accuracy	Test Accuracy	best
$\sigma = 0.01$	LogBarrier	-1.0182e+03	**0.9992**	0.7591	✓
	SMO	-983.8036	0.9967	0.7618	
$\sigma = 0.1$	LogBarrier	-2.5957e+04	0.9175	0.7766	
	SMO	-1.8889e+04	0.8675	0.7429	
$\sigma = 0.5$	LogBarrier	-3.9409e+04	0.8725	**0.8456**	✓
	SMO	-2.8002e+04	0.8442	0.7927	
$\sigma = 1$	LogBarrier	-4.2430e+04	0.8517	0.8116	
	SMO	-3.1060e+04	0.8417	0.7833	

- Again note that the Log Barrier method produces a lower objective value than SMO for all kernel widths and is therefore more accurate in the pure optimisation sense.
- Although $\sigma = 0.01$ achieves 99.92% training accuracy, its test accuracy is far lower at 75.91%. Roughly, a small kernel width means that "small scale" dependencies between samples are more important in the feature space, which is excellent for fitting tightly and accurately to the training data. However, for test data this overfitting will result in much higher test error. There is a noticable trend that smaller kernel width results in higher training accuracy but also higher generalisation error (lower test accuracy).
- Kernel width $\sigma = 0.5$ and the Log Barrier method produces the best test accuracy at 84.56%.

Similar to the polynomial case, the Log Barrier method outperforms SMO. Smaller kernel width seems to increase train accuracy but at the expense of test accuracy. A good compromise seems to be $\sigma = 0.5$ which produces reasonable training accuracy and the highest test accuracy.

4.1.3 A comparison

Based on our experiments in the previous two sections we conclude that the Gaussian kernel outperforms the Polynomial kernel for this dataset. The Gaussian kernel exhibits the lowest test accuracy score of 84.56% with $\sigma = 0.5$ and $C = 100$ whilst the highest test accuracy for the polynomial kernel was 82.37% for $p = 2$ and $C = 10$. Furthermore the Gaussian kernel generally achieves much higher training accuracy than the Polynomial kernel, and gets close to 100% training accuracy for $\sigma = 0.01$ whereas the highest training accuracy for the ploynomial is 85.00% for $p = 2$. However, we do note that the Gaussian kernel is much more likely to overfit to the data, and high training accuracy using a small kernel width comes with the price of higher test error. For the rest of the report, therefore, we analyse the convergence rates of these algorithms with parameters:

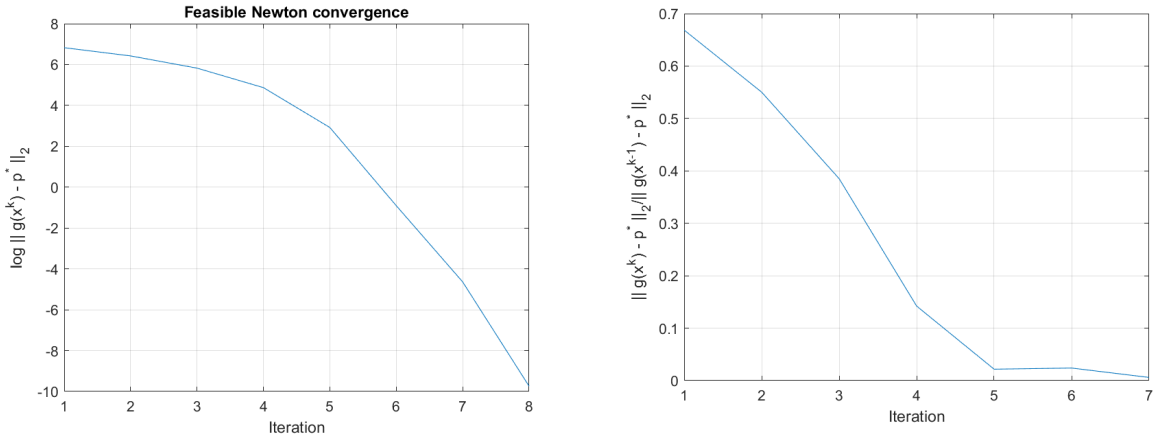
- Polynomial Kernel: $p = 2$ and $C = 10$
- Gaussian Kernel: $\sigma = 0.5$ and $C = 100$

4.2 Convergence Rates

In this section we analyse the convergence rates of both algorithms, exploring what we observe with respect to the theoretical convergence statements.Since we work with the dual form of the SVM problem we would expect to see very similar performance using polynomial and Gaussian kernels (up to precomputation of the kernel matrix).

4.2.1 Barrier Method Convergence - Theoretical vs Practical

- There are two sets of convergence statements for the Log Barrier method. Going back to Algorithm 1, the first concerns the *inner* loop consisting of the feasible Newton step (lines 4-8). The second concerns the *outer* loop consisting of the trajectory of points on the central path. The theoretical convergence properties of the feasible Newton steps are the same as for applying Newton's method to the unconstrained problem. Hence we would expect **local quadratic convergence**. We verify this by plotting the values $\log \|g(\lambda^k) - g(\lambda^*)\|$ and ratio $\frac{\|g(\lambda^k)-g(\lambda^*)\|}{\|g(\lambda^{k-1})-g(\lambda^*)\|}$ of the Newton iterations during the *last* centering step for the Gaussian kernel in Figure 4.3.

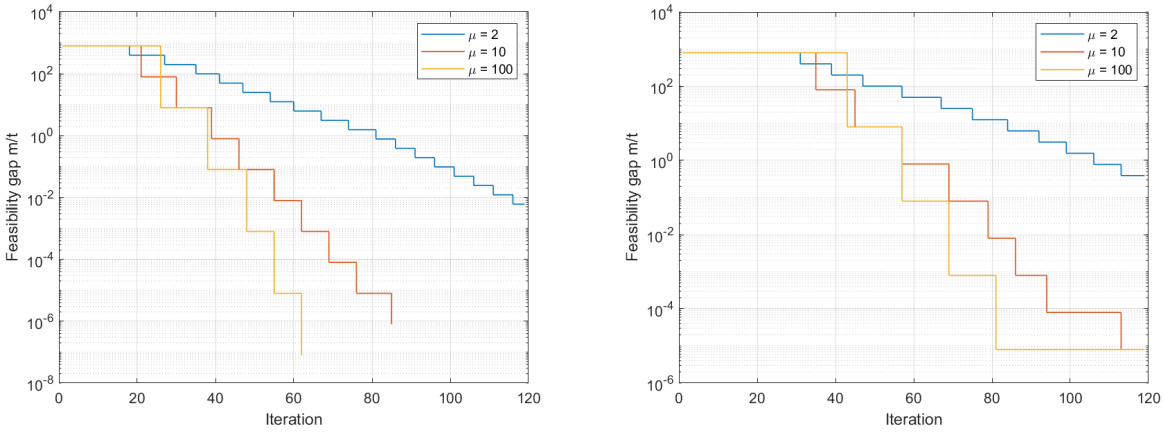


(a) $\log \|g(\lambda^k) - g(\lambda^*)\|$ (b) $\frac{\|g(\lambda^k)-g(\lambda^*)\|}{\|g(\lambda^{k-1})-g(\lambda^*)\|}$

Figure 4.3: Last Centering step

- The shape of the graph indicates quadratic local convergence as well as the ratio $\frac{\|g(\lambda^k)-g(\lambda^*)\|}{\|g(\lambda^{k-1})-g(\lambda^*)\|} \approx 0$ and thus verifies that the practical convergence rate seems to match the theoretical rate. Since at some point the algorithm will take a last centering step, we rather trivially have quadratic *local* convergence.

- Secondly, we wish to analyse the convergence properties of the outer loops. Although a superlinear convergence rate of the objective function has been proven for a linear objective [10], no such guarantee has been proven for the quadratic case we have here. However, equation (3.6) shows that the duality gap between the objective at a central point $g(\lambda^*(t))$ and the primal optimum p^* is bounded by m/t, which implies global convergence as $t \to \infty$. In the algorithm, t is increased by a factor μ so that $t \leftarrow \mu t$ after a centering step. This suggests a **trade-off** between a small $\mu \approx 1$ which leads to a good initial guess for the Newton method - resulting in fewer inner iterations, but a large number of outer iterations to achieve $m/t \leq \epsilon$ accuracy. Conversely a large value of μ will require fewer outer iterations but more inner iterations as a result of stepping far past the central path. This trade-off is demonstrated in Figure 4.4. The feasibility gap m/t is plotted as a function of iteration number for $\mu \in [2, 10, 100]$. We set $t^{(0)} \leftarrow 1$ and have $m = 2 \times \mathcal{T}$ where $\mathcal{T}$ is the size of the training set.



(a) Polynomial (b) Gaussian

Figure 4.4: Feasibility Gap, varying μ

- Observe that a smaller multiplying factor $\mu = 2$ leads to numerous centering steps (the vertical portions), but few Newton steps *during* centering (the horizontal portions), whereas a larger factor $\mu = 100$ requires many Newton iterations but takes fewer centering steps to reach tolerance. This demonstrates the trade-off implied by the theoretical convergence result. In fact, a large range of μ values result in a very similar number of total Newton steps. We plot the number of Newton steps versus the value of μ below in Figure 4.5. For $\mu < 10$ the number of Newton steps does increase significantly, but for values $\mu > 10$ the difference in performance is largely random. This motivated the choice of $\boldsymbol{\mu = 20}$ in all our experiments.

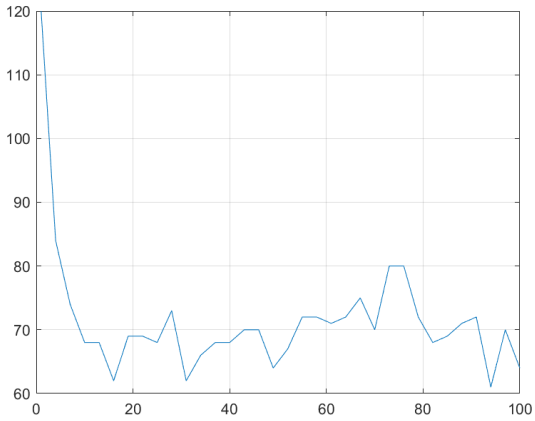
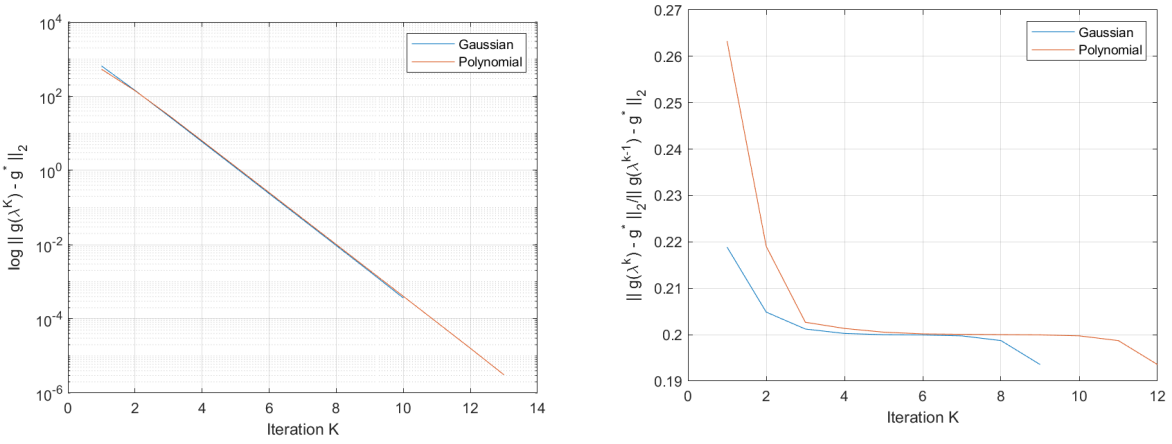


Figure 4.5: Number of iterations for varying μ

- Furthermore we plot the convergence of the objective function $\log \|g(\lambda^{(K)}) - g(\lambda^*)\|$ and ratio $\frac{\|g(\lambda^{(K)}) - g(\lambda^*)\|}{\|g(\lambda^{(K-1)}) - g(\lambda^*)\|}$ in Figure 4.6 where K is the index of an **outer** iteration. As noted above, there is only a theoretical guarantee of *superlinear* convergence in the case of a linear objective, which in our case will not hold since our objective is quadratic. In fact we observe that the objective function on a log scale decreases at a linear rate, and the ratio tends to a value $0.05 \in (0, 1]$. Both these results indicate *linear* convergence. This is to be expected since we'd expect a quadratic objective to have a rate of convergence just as worse as a linear objective.



(a) $\log \|g(\lambda^{(K)}) - g(\lambda^*)\|$ (b) $\frac{\|g(\lambda^{(K)}) - g(\lambda^*)\|}{\|g(\lambda^{(K-1)}) - g(\lambda^*)\|}$

Figure 4.6: Outer iteration convergence

4.2.2 SMO Convergence - Theoretical vs Practical

- When the indices (i, j) of the Lagrange multipliers to be updated are a maximal violating pair, [8] states that the dual objective function $g(\lambda)$ in (1.23) is guaranteed to strictly decrease, i.e. at iteration k: $g(\lambda^{k+1}) < g(\lambda^k)$. Furthermore, under the following assumptions:

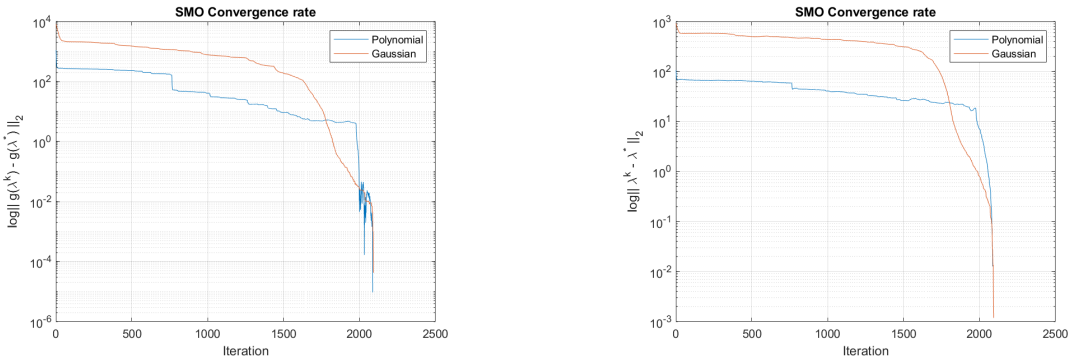
  (1) The kernel matrix K is positive definite

  (2) Strict complementarity

  the theoretical convergence rate of the objective function for this algorithm is **linear** [9]. They state that there exists $c < 1$ such that for k large enough

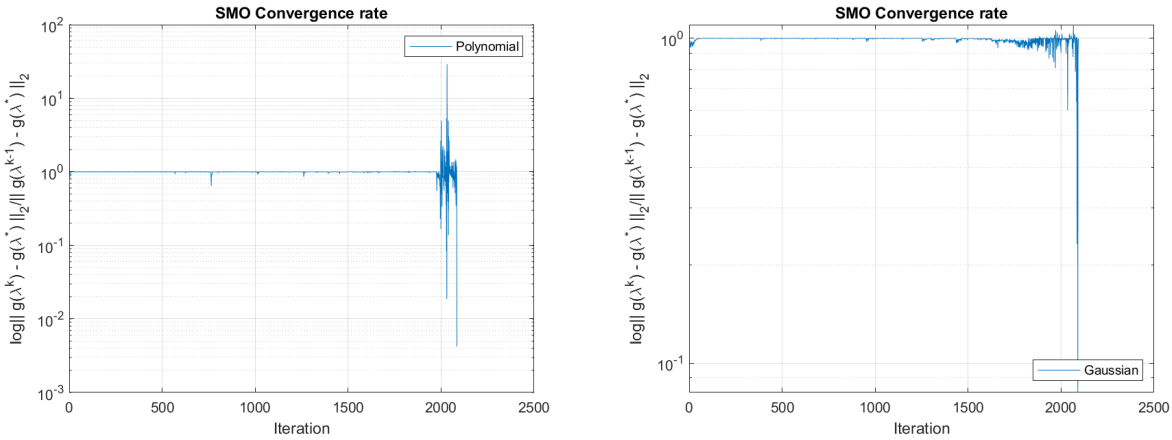
$$g(\lambda^{k+1}) - g(\lambda^*) \leq c(g(\lambda^k) - g(\lambda^*)) \tag{4.1}$$

- Note that condition (1) will generally be observed as long as there are no repetitions of data and condition (2) will require closer inspection of the solutions. We now determine whether the theoretical convergence rate holds in a practical context by plotting the convergence of the objective function $\log \|g(\lambda^k) - g(\lambda^*)\|$ and ratio $\log \frac{\|g(\lambda^k) - g(\lambda^*)\|}{\|g(\lambda^{k-1}) - g(\lambda^*)\|}$ as well as the convergence of the iterates $\log \|\lambda^k - \lambda^*\|$. This is done for both the Gaussian and Polynomial kernel in Figures 4.7 and 4.8



(a) Convergence rate of objective function (b) Convergence rate of iterates

Figure 4.7: SMO convergence rates



(a) Polynomia kernel (b) Gaussian

Figure 4.8: SMO convergence ratio $\log \frac{\|g(\lambda^k)-g(\lambda^*)\|}{\|g(\lambda^{k-1})-g(\lambda^*)\|}$

- In Figure 4.7a we first note that the first theoretical convergence statement - a strict decrease in the objective function - is not always observed. This is likely due to the iterations where the maximal violating pair working set selection criterio fails. In this case a random pair of indices to optimise over are chosen and therefore the strict decrease in objective function is not obeyed.
- Secondly, we note that in Figures 4.7a and 4.7b that the decrease in objective function and iterate differences converge slowly to the solution. For the first 1500 iteration or so the decrease in both values is quite slow. However, it is hard to analyse the exact convergence rate.
- If Figures 4.8 we show the convergence ratios $\log \frac{\|g(\lambda^k)-g(\lambda^*)\|}{\|g(\lambda^{k-1})-g(\lambda^*)\|}$. We note that the ratio stays mostly at the value 1, but then becomes erratic at the later iterations - again this is likely due to the overly strict stopping condition (all KKT conditions must be obeyed) causing the algorithm to revert to a random working subset. We therefore note that these graphs suggest that the convergence rate is indeed *linear* - the general trend of the plots displays a ratio of mostly 1.

4.2.3 CPU Performance

- In this section we present some results about the practical performance of the algorithms. The hyperparameters were chosen as in the previous section and the linear kernel was used. In particular we study how the performance of the algorithms change with respect to the size of the training set. The total dataset consists of 1942 samples. For the experiment we measure the number of iterations, CPU time and CPU time per iterations for both algorithms until convergence, using a training set of size $\mathcal{T} = 300, 600$ and 900. The results are presented in the table below.

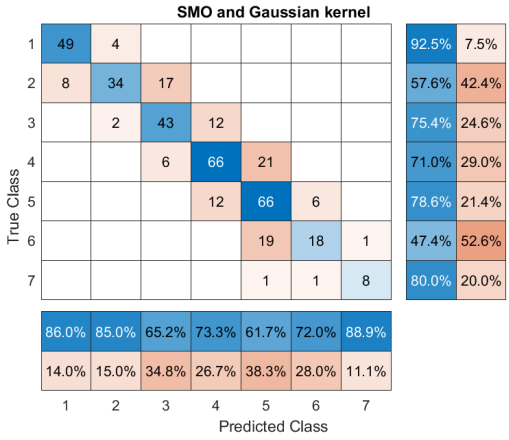
		Performance Indicators			
		NumIter	CPU time (s)	time/iteration	best
$\mathcal{T} = 300$	LogBarrier	53	0.680	0.012	
	SMO	901	**0.051**	0.000057	✓
$\mathcal{T} = 600$	LogBarrier	52	4.418	0.0847	
	SMO	930	**0.124**	0.0001	✓
$\mathcal{T} = 900$	LogBarrier	54	14.451	0.267	
	SMO	1824	**0.315**	0.0001	✓

- Across all sizes, the SMO algorithm has the lowest CPU time. The difference in time increases as the training set size grows. This is because the Log Barrier method relies on the inversion of a Hessian which grows with the size of training set, as in Equation (3.4). More sophisticated interior point methods reduce this bottleneck by approximating the Hessian. Conversely, SMO's decomposition strategy scales efficiently with the size of the training set and is therefore suited for large scale SVM problems.

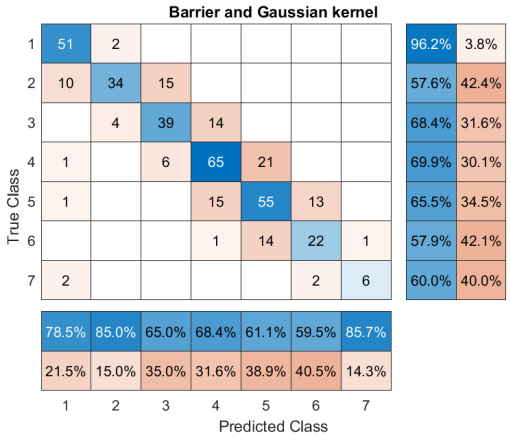
4.3 Multiclass

- As an extension we generalise the SVM to the multiclass case using the One-Versus-All (OVA) method discussed in section 1.7. Hence we trained K classifiers over the classes 'AAA', 'AA', 'A', 'BBB', 'BB', 'B' an 'CCC'. We use a random 80/20 training/test split over the entire dataset. In order to simplify the analysis we do not cross-validate over the hyperparameters again and instead select the optimal parameters we discussed in the previous section. Namely, for the Gaussian kernel we use a kernel width $\sigma = 0.5$ and set $C = 100$, and for the Polynomial kernel we use degree $p = 2$ and set $C = 10$.

- We used the Log Barrier and SMO methods the train each of the $K = 7$ binary classifiers on the training data. We then used the test data to compute the test error (assigning the prediction of each test point according to equation (1.24) and produce a confusion matrix. The results are displayed in the Figure 4.9 and 4.10 (class 1 corresponds to 'AAA' and class 7 to 'CCC' etc.).

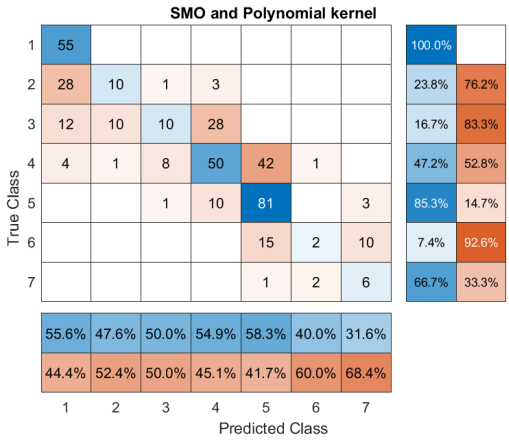


(a) SMO - test accuracy 72.08%

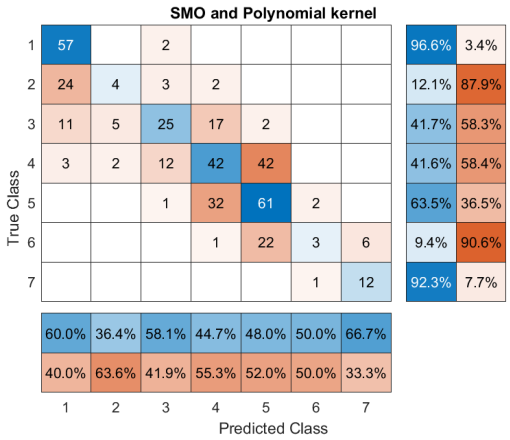


(b) Barrier - test accuracy 69.10%

Figure 4.9: Confusion matrix - Gaussian kernel



(a) SMO - test accuracy 54.31%



(b) Barrier - test accuracy 51.78%

Figure 4.10: Confusion matrix - Polynomial kernel

Comments

- Both kernels struggle to separate the data, with the highest test accuracy for the Gaussian kernel being 72.08% and the Polynomial kernel 54.37% (both from the SMO algorithm). A more thorough search strategy to yield more optimal hyperparameters C, σ and p could increase these accuracies. However, we note that we would not expect to see a significant performance increase - the SVMs struggle to classify the data between adjacent classes (e.g. between 'BBB' and 'BB' or 'BB' and 'B'). Hence we conclude that the data is highly non-separable.
- Under the best setting of the parameters, the Gaussian kernel outperforms the polynomial kenrnel significantly - by more than 15%, suggesting it is more capable of separating this data.
- The dataset is quite imbalanced. This may hinder the performance of our SVM classifiers and a modified version may be more suitable as suggested in [11].
- The SMO method outperforms the Barrier method for both choices of kernel. Additionally its training time is significantly lower, hence we conclude that SMO is much more suited to solve the SVM optimisation problem.

References

[1] Cristianini, N. Shawe-Taylor, J (1999). An Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods. Cambridge University Press

[2] Pontil, M. Verri, A. (2001). Properties of Support Vector Machines. http://www0.cs.ucl.ac.uk/staff/m.pontil/reading/sv_prop.pdf

[3] Chih-Wei, H. and Chih-Jen, L. (2002). "A comparison of methods for multiclass support vector machines," in IEEE Transactions on Neural Networks, vol. 13, no. 2, pp. 415-425

[4] Platt, John. (1999). Fast Training of Support Vector Machines Using Sequential Minimal Optimization. Advances in Kernel Methods: Support Vector Learning. 185-208.

[5] Cortes, C., Vapnik, V. (1995). Support-vector networks. Mach Learn 20, 273–297.

[6] Boyd, S., & Vandenberghe, L. (2004). Convex Optimization. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511804441

[7] Fiacco, A., McCormick, G., & Danskin, J. (1967). The Sequential Unconstrained Minimization Technique (SUMT) without Parameters. Operations Research, 15(5), 820-829. Retrieved April 6, 2020, from www.jstor.org/stable/168637

[8] Pai-Hsuen Chen, Rong-En Fan, and Chih-Jen Lin. 2006. A study on SMO-type decomposition methods for support vector machines. Trans. Neur. Netw. 17, 4 (July 2006), 893–908. DOI:https://doi.org/10.1109/TNN.2006.875973

[9] Lin, Chih-Jen. (2003). Linear Convergence of a Decomposition Method for Support Vector Machines

[10] Wright, S. (2001). On the convergence of the Newton/log-barrier method. Math. Program. 90, 71–100. https://doi.org/10.1007/PL00011421

[11] Akbani R., Kwek S., Japkowicz N. (2004) Applying Support Vector Machines to Imbalanced Datasets. In: Boulicaut JF., Esposito F., Giannotti F., Pedreschi D. (eds) Machine Learning: ECML 2004. ECML 2004. Lecture Notes in Computer Science, vol 3201. Springer, Berlin, Heidelberg