UNIVERSITY OF NOTTINGHAM

MATH4069: Statistical Machine Learning

An Exploration into Support Vector Machines with Comparisons to other Classification Methods

Group G

Jake Dorman

Timi Folaranmi

Anas Almhmadi

Rishabh Agarwal

We have read and understood the School and University guidelines on plagiarism. We confirm that this work is our own, apart from the acknowledged references.

Abstract

Support Vector Machines (SVMs) are a statistical machine learning technique traditionally used to tackle binary classification problems. This works by building a linear hyperplane to separate the data into the two separate classes. We see that for fully separable data, an Optimal Separating Hyperplane can be constructed by maximising the distance between the hyperplane and the closest points to it, referred to as the margin. For non-separable data, maximising the margin and minimising classification error can be balanced by a regularisation parameter C.

We then extend SVMs to data with a non-linear relationship by introducing kernels, which allow the efficient extension of data into higher dimensional spaces, so that data may be separated using a linear hyperplane. Kernalised SVMs are then applied to a real-life breast cancer classification dataset, in which we find that an SVM with a Gaussian Radial Basis Function (RBF) Kernel performs better than many other binary classification methods, such as Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA) and Logisitic Regression.

Contents

1 Introduction

Binary classification of categorical data is one of the most common tasks in machine learning. In this paper, we introduce a non-probabilistic method for solving this problem, the Support Vector Machine (SVM). This works by building a linear hyperplane to separate the data into the two binary classes. This hyperplane is found by maximising the margin, which is the distance between the hyperplane and the nearest points. In this paper, we aim to:

- Formalise the mathematics behind SVMs and find a numerically solvable algorithm to implement them,
- Show how SVMs can be extended to tackle data with nonlinear relationships,
- Test SVMs on a real data set to compare their use to other forms of classification,
- Explore how SVMs are being used to tackle real life problems in modern literature.

1.1 Historical Context

SVMs, as known to us in their current form, were introduced first by Vapnik et al in 1992 [1]. However, the history of SVMs goes much before this. The Generalized Portrait Algorithm was introduced by Vapnik and Lerner in 1963 [2]. The current SVM algorithm is a non-linear generalisation of this algorithm. In 1974, Vapnik developed the field of "Statistical Learning Theory" which contained some of the fundamental concepts used by SVMs in later years. In the 1992 paper, Vapnik introduced SVMs in their current form, but these only handled the 'Hard Margin' scenario, that is classes which are completely separable. Vapnik and Cortes introduced the 'Soft Margin' classifier in 1995 [3] to handle the issues of non-separable data. While these developments were taking place in the field of SVMs, the mathematics of kernels was a separate field of study. The fundamental contribution to the field of Kernels came in 1950 in the paper "Theory of Reproducing Kernels" by Aronszajn

[4]. In the 1992 paper by Vapnik, along with the hard-margin classifier, they also showcase how various kernels can be used to generalise SVMs to be non-linearly dependent on the input features. They present experimental results on optical character recognition problems demonstrating the good generalisation obtained compared to other learning algorithms.

1.2 Summary of main findings

In the linear section of the report, we find that SVMs can be formulated as a Dual Lagrangian Optimisation problem, which can be optimised using an algorithm known as the Sequential Minimal Optimisation (SMO) Algorithm. Furthermore, only data points found on or over the margins of the classifier contribute towards the parameter values of the hyperplane.

When extending our research to non - linearly separable data, we introduce the idea of kernels, and thus the kernel trick. We demonstrate that through the use of kernels such as the Polynomial Kernel and the Gaussian Radial Basis Function Kernel, higher dimensional classification of non - linearly separable samples can occur in a computationally efficient manner.

In applying SVMs to a Breast Cancer database, using K - fold cross validation, we find that when comparing with linear models on the same dataset, all 4 models (Naïve Bayes, LDA, Logistic, and SVM-Linear) give a strong 'accuracy' measure above 90%, with the linear SVM performing the best with an accuracy of 97.5%. When comparing with non - linear models (QDA, SVM (polynomial), SVM-RBF), we find that the SVM with a RBF kernel performs the highest, with an accuracy of 97.9%.

Critically, whilst SVMs perform well, when considering accuracy metrics, we observe that they are generally less preferred in settings where fast classification speed is needed. This is primarily due to the quantity of support vectors and decision boundaries being decided by the support vectors.

1.3 Report Structure

Section 2 introduces the mathematics governing SVMs, where we look at the hard margin and soft margin classifiers. In this section we formulate the problem as a numerically solvable optimization problem and outline a method to implement this. In Section 3 we cover the scenario when there is non-linearity present in the data and the decision boundaries can no longer be linear. The kernel trick is introduced and we see some examples to demonstrate the generalization of Linear Support Vector Classifiers to non-linear data. In Section 4, we compare the performance of Support Vector Classifiers to other classifiers with the help of a breast cancer classification dataset, to understand the benefits and limitations of SVMs. Finally, we look at some uses of SVMs in modern literature and the types of problems they are used to solve in practice.

2 Linear Support Vector Machines

We have access to a training set $S = \{(\mathbf{x}^{(i)}y^{(i)})\}_{i=1}^{N}$, where $\mathbf{x}^{(i)} \in \mathbb{R}^p, y^{(i)} \in \{-1, 1\}$, and aim to construct a hyperplane H that best separates the two classes. This hyperplane H will be of the form

$$\boldsymbol{\theta}^\top \mathbf{x} + \theta_0 = 0, \tag{2.0.1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^p$ is the normal vector to the hyperplane and $\theta_0 \in \mathbb{R}$ is a weight [5]. Classifications are then made using the following model:

$$\hat{y}^{(i)} = \text{sign}(\boldsymbol{\theta}^\top \mathbf{x}^{(\mathbf{i})} + \theta_0) \tag{2.0.2}$$

We will find the parameters $\boldsymbol{\theta}, \theta_0$ by maximising the distance m between the hyperplane and the nearest points to the hyperplane. This distance is known as the margin.

Note that the classes are -1 and 1 whereas for other (usually probabilistic) methods of binary classification, the classes are often 0 and 1. This is so that the classifier 2.0.2 can be defined using the sign function. Any classification problem can be written in either form by simply adding or subtracting 1 from the lower-valued class.

2.1 Hard Margins: The Fully Separable Case

First, we assume that the data is linearly separable, meaning that there exists at least one hyperplane that perfectly separates the data into the two classes. This can then be extended quite easily to the non-fully separable case in the next section.

Figure 1 shows an example of a fully separable dataset with two-dimensional input variables. The margin in Figure 1 is represented by the dotted lines between the circled points and the hyperplane. There is an infinite number of possible hyperplanes to separate this data, but we aim to find the **optimal separating hyperplane**, which is the hyperplane that maximising the margin. We will formalise this as a minimisation problem, and then rearrange it into a form that can be solved numerically [6].

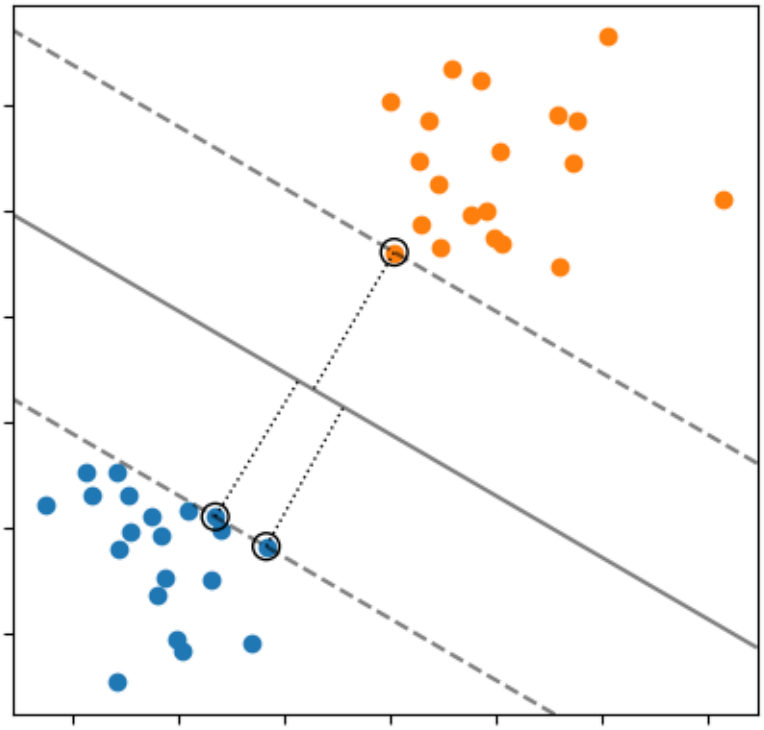

Figure 1: Fully separable data being split by the optimal separating hyperplane.

First, we must find an expression for the margin. The signed distance $M^{(i)}$ between any point i and the hyperplane is

$$M^{(i)} = \frac{\mathbf{w}^\top \mathbf{x}^{(i)} + b}{\|\mathbf{w}\|}. \tag{2.1.1}$$

We can assume that our selection of $\boldsymbol{\theta}$ and θ_0 is such that $M^{(i)}$ is positive on the side of the $y = +1$ class and is negative on the side of the $y = -1$ class, such that the equation

$$\tilde{M}^{(i)} = \frac{y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0)}{\|\boldsymbol{\theta}\|} \geq 0 \tag{2.1.2}$$

always gives the positive distance from H to point i. Then, the margin m is the distance between the hyperplane and the closest point, which is

$$\begin{aligned} m &= \min_i \tilde{M}^{(i)} \\ m &= \frac{1}{\|\boldsymbol{\theta}\|} \min_i [y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x} + b)]. \end{aligned} \tag{2.1.3}$$

From this, the optimisation problem can be constructed. The optimisation problem is to find the values of $\boldsymbol{\theta}$ and θ_0 that maximise the margin, under the constraint given in

Equation 2.1.2:

$$\max_{\boldsymbol{\theta},\theta_0} m \tag{2.1.4}$$
$$\text{subject to } y^{(i)}(\theta^\top \mathbf{x}^{(i)} + b) \geq 0.$$

Since m is the minimum distance of a point from the hyperplane, currently we are maximising a minimum. It would be convenient if we only had to deal with a minimum or a maximum. This is done by rearranging Equation 2.1.3. The magnitude of the normal vector $\boldsymbol{\theta}$ can be scaled without affecting the position of the hyperplane, so we can choose $|\boldsymbol{\theta}| = 1/m$. Hence, equation 2.1.3 becomes

$$1 = \min_i [y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + b)], \tag{2.1.5}$$

or equivalently

$$1 \leq [y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x} + b)], \, \forall i. \tag{2.1.6}$$

Note that this creates a stronger constraint than in 2.1.4, since the lower bound is now by 1 instead of 0. Using these adjustments, equation 2.1.4 becomes

$$\max_{\boldsymbol{\theta},\theta_0} \frac{1}{\|\boldsymbol{\theta}\|} \tag{2.1.7}$$
$$\text{subject to } y^{(i)}(\theta^\top \mathbf{x}^{(i)} + b) \geq 1, \, \forall i.$$

This can be rewritten as an equivalent minimisation problem:

$$\min_{\boldsymbol{\theta},\theta_0} \frac{1}{2}\|\boldsymbol{\theta}\|^2 \tag{2.1.8}$$
$$\text{subject to } 1 - y^{(i)}(\theta^\top \mathbf{x}^{(i)} + b) \leq 0, \, \forall i.$$

Due to the constraints, it is not possible to use gradient descent on this problem without changing it further first. Instead, Lagrangian duality will be used to solve the problem.

2.1.1 Hard Margins: Lagrangian Duality

We are trying to solve equation 2.1.8 using Lagrangian Duality. The Lagrangian for this problem is

$$\mathcal{L}(\boldsymbol{\theta}, \theta_0, \alpha^{(i)}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + \sum_{i=1}^{N} \alpha^{(i)}[1 - y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0)], \tag{2.1.9}$$

where $\alpha^{(i)} \geq 0, \forall i$. The objective is now

$$\min_{\boldsymbol{\theta},\ \theta_0} \mathcal{L}(\boldsymbol{\theta},\theta_0, \alpha) \tag{2.1.10}$$

$$\text{subject to } \alpha^{(i)} \geq 0.$$

We aim to find the dual objective of this, which is a minimisation objective depending on the $\alpha^{(i)}s$ instead of on the parameters $\boldsymbol{\theta}$ and θ_0, as this form of the problem is much easier to solve. We find this by setting the partial derivatives of $\mathcal{L}$ to be 0. This gives us two equations:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \boldsymbol{\theta} - \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} = 0 \implies \boldsymbol{\theta} = \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}, \tag{2.1.11}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = -\sum_{i=1}^{n} \alpha^{(i)} y^{(i)} = 0. \tag{2.1.12}$$

The first equation gives us an explicit formula for the weights $\boldsymbol{\theta}$, which can be used to eliminate this variable from equation 2.1.9, whilst the second equation gives a new constraint. The Lagrangian (Equation 2.1.9) becomes

$$\mathcal{L}(\alpha^{(i)}) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^\top \mathbf{x}^{(j)} \tag{2.1.13}$$

The full derivation of this can be seen in Appendix 8.1. This gives us a final minimisation problem:

$$\min_{\alpha^{(i)}} \mathcal{L}(\alpha^{(i)}) \tag{2.1.14}$$

$$\text{subject to: } \alpha^{(i)} \geq 0,$$

$$\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0. \tag{2.1.15}$$

This form of the problem can be solved numerically using an algorithm called the Sequential Minimal Optimisation (SMO) algorithm [7] (appendix 8.2), which gives explicit values for the optimal $\alpha^{(i)}$, denoted $\alpha^{*(i)}$. Note that by the Karush–Kuhn–Tucker (KKT) Complimentary Slackness condition [8], we have for all i,

$$\alpha^{(i)}[1 - y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0)] = 0. \tag{2.1.16}$$

This means that for each i, either $\alpha^{(i)} = 0$ or $1 - y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) = 0$. The points where $\alpha^{(i)} \neq 0$ are known as the **support vectors**, which are circled in Figure 1. The support vectors are the points that are on the margin, meaning they are exactly m units away from the hyperplane. These are the only points that determine the position of the hyperplane; all other points can be moved and the hyperplane will remain in the same position (provided the points are not moved onto or across the margin).

Using the support vectors, the optimal $\boldsymbol{\theta}^*$ can be found from 2.1.11, though the sum can be reduced to be just over the support vectors since all other values will be 0. All that is left now is to find θ_0. This can be found using the Complimentary Slackness condition 2.1.16. Assuming i is a support vector, we have

$$y^{(i)}((\boldsymbol{\theta}^*)^\top \mathbf{x}^{(i)} + \theta_0) = 1. \tag{2.1.17}$$

By multiplying this by $y^{(i)}$ and using the fact that $(y^{(i)})^2 = 1$, this can be rearranged for θ_0 :

$$\theta_0 = y^{(i)} - (\boldsymbol{\theta}^*)^\top \mathbf{x}, \tag{2.1.18}$$

which can be found by substituting in the values for $\mathbf{x}^{(i)}$ and $y^{(i)}$ from any of the support vectors.

To summarise, the optimal separating hyperplane in the hard margins case can be found using the following steps:

1. Solve Equation 2.1.10 using the SMO Algorithm (Appendix 8.2), to find the support vectors.

2. Solve Equation 2.1.11 for $\boldsymbol{\theta}$, summing over the support vectors.

3. Solve Equation 2.1.18 for θ_0, where i is one of the support vectors.

4. Then the classifier is

$$\text{sign}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0). \tag{2.1.19}$$

2.1.2 Problems with the Hard Margin

So far we have determined how to build a separating hyperplane in the case where the data is linearly separable, meaning we could assume that all data is on the correct side of the hyperplane. In many cases, data is not separable in this way, and any hyperplane we make will misclassify at least one point. In this case, the assumptions necessary to build the optimal hyperplane in the hard margin case are not satisfied. Furthermore, even if

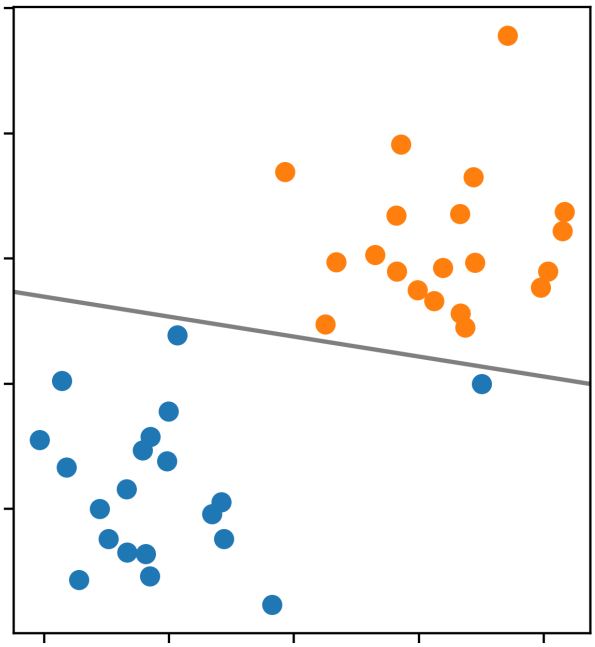

Figure 2: The optimal separating hyperplane is overfitted to an outlier data point.

the data is fully separable, using the hyperplane which maximises the margin still may not be the best classifier. This is because this classifier only takes into account the points

directly on the margin, meaning it can become incredibly overfit to this small number of points. Figure 2 shows the optimal separating hyperplane for some data, where the blue class appears to have an outlier. This does fit the data into the two separate classes, but it is clear that the hyperplane does not truly reflect how the data is separated. We need to build a hyperplane that best separates the data in both of these cases. We do this using a **soft margin classifier**.

2.2 Soft Margins: The Non-Separable Case

We aim to build a hyperplane that separates two classes of data that are not necessarily fully separable.

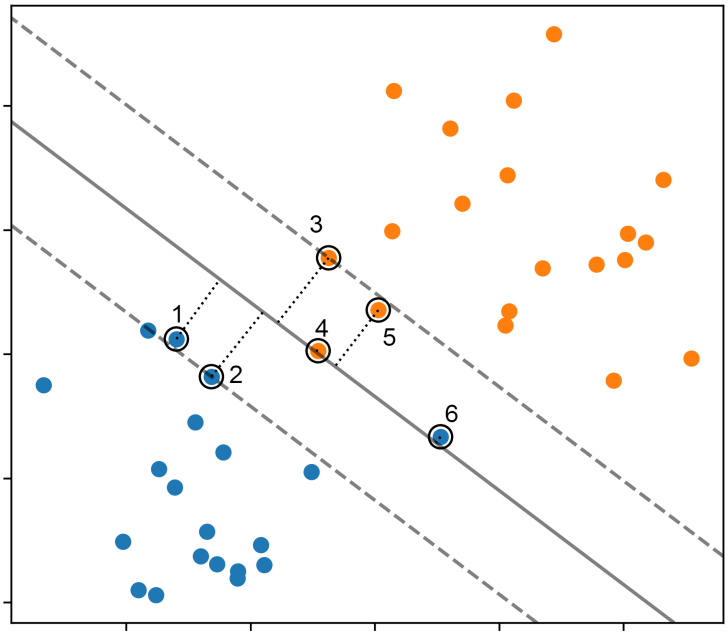

Figure 3: Data being separated by an SVM decision boundary. The numbered data points are the support vectors.

Figure 3 shows a hyperplane where points $1, 4, 5$ and 6 are within the margin, and point 6 is even on the wrong side of the hyperplane. This is known as a **soft margin classifier**, as it allows for mistakes to be made when building the decision boundary. As such, we

need a method to determine how much we tolerate a point being misclassified. To do this, we introduce slack variables $\xi^{(i)}$ which are measures of how much we allow a datapoint to be over the margin. These must be bounded below by $\xi^{(i)} = 0$, which represents the case where we do not allow points to be over the margin at all. We must also modify Innequality 2.1.6 to allow for these slack variables. As such, $\xi^{(i)}$ have the following constraints:

$$\xi^{(i)} \geq 0, \tag{2.2.1}$$

$$y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) \geq 1 - \xi^{(i)}. \tag{2.2.2}$$

Since in the hard margin case, we normalised the margin to 1, the second inequality ensures that no points are more than $\xi^{(i)}$ units over the margin. The slack is illustrated in Figure 4. Points 4 and 7 will have $\xi^{(i)} = 0$, since they are on the margin but not over it. Points 2 and 3 will have positive but relatively small values for the slack, since they are slightly over the margin, whilst points 1, 5, 6, 8 and 9 will have larger values. All other points will have a slack of 0, since they are not over the margins.

Now that we have introduced the slack variables, we can adjust the optimisation problem found in the hard margin case to include these. Now, we not only want to maximise the margin (which we saw was equivalent to the minimisation problem in 2.1.8), but also to minimise the slack variables, as we want to have as little tolerance for errors as possible. Hence, the optimisation problem is

$$\min_{\boldsymbol{\theta}, \theta_0, \xi^{(i)}} \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{i=1}^{N} \xi^{(i)} \tag{2.2.3}$$

$$\text{subject to: } \xi^{(i)} \geq 0$$

$$y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) + \xi^{(i)} - 1 \geq 0.$$

The first term in the minimisation problem corresponds to maximising the margin as before, whilst the second corresponds to minimising the slack. A hyperparameter C has been

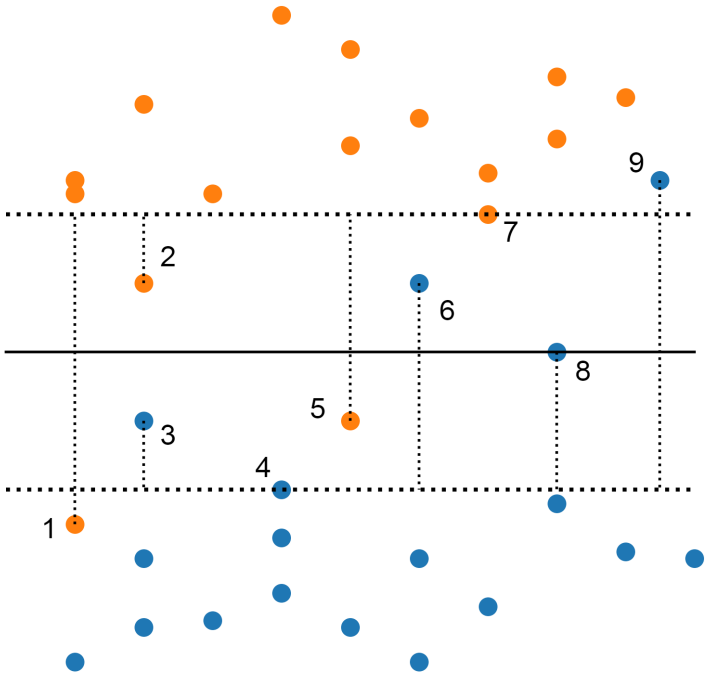

Figure 4: Data being separated by a hyperplane (full line), with a margin on either side (dashed line).

introduced which embodies the trade-off between maximising the margin and minimising the slack. C is known as the **regularisation parameter**. If C is large, it means we become less tolerant of errors and prioritise reducing the slack over maximising the margin. As such, the margin will decrease. The case where $C = \infty$ corresponds to the hard margin case, since any slack at all will cause the objective function to have infinite value.

Equation 2.2.3 can be solved in the same way as the soft margin problem, using Lagrangian Duality.

2.2.1 Soft Margins: Lagrangian Duality

The Lagrangian for 2.2.3 is:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \theta_0, \xi^{(i)}, \alpha^{(i)}, \beta^{(i)}) = &\frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{i=1}^{N}\xi^{(i)} \\ &- \sum_{i=1}^{N}\alpha^{(i)}\left(y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0\right) + \xi^{(i)} - 1) \\ &- \sum_{i=1}^{N}\beta^{(i)}\xi^{(i)}, \end{aligned} \tag{2.2.4}$$

and the optimisation problem is

$$\begin{aligned}&\min_{\boldsymbol{\theta}, \theta_0, \xi^{(i)}} \mathcal{L}(\boldsymbol{\theta}, \theta_0, \xi^{(i)}, \alpha^{(i)}, \beta^{(i)}) \\ &\text{subject to } \alpha^{(i)}, \beta^{(i)} \geq 0.\end{aligned} \tag{2.2.5}$$

We aim to find the dual of this problem so that we can use the SMO algorithm (Appendix 8.2) to solve it. We do this as before, by differentiating with respect to $\boldsymbol{\theta^{(i)}}, \theta_0$ and $\xi^{(i)}$ and setting these to 0. These partial derivatives are:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \boldsymbol{\theta} - \sum_{i=1}^{n}\alpha^{(i)}y^{(i)}\mathbf{x}^{(i)} = 0 \implies \boldsymbol{\theta} = \sum_{i=1}^{n}\alpha^{(i)}y^{(i)}\mathbf{x}^{(i)} \tag{2.2.6}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = -\sum_{i=1}^{n}\alpha^{(i)}y^{(i)} = 0. \tag{2.2.7}$$

$$\frac{\partial \mathcal{L}}{\partial \xi^{(i)}} = C - \alpha^{(i)} - \beta^{(i)} = 0. \tag{2.2.8}$$

The first two of these are identical to equations 2.1.11 and 2.1.12 in the hard margins section. The third allows us to use $\beta^{(i)} = C - \alpha^{(i)}$ so that we can formulate the dual problem in terms of just $\alpha^{(i)}$. Since $\beta^{(i)} \geq 0$, this gives us an additional constraint: $\alpha^{(i)} \leq C$. These can be inserted into 2.2.4 and rearranged to find the dual Lagrangian:

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N}\alpha^{(i)} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}(\mathbf{x}^{(i)})^\top \mathbf{x}^{(j)}. \tag{2.2.9}$$

This is shown in Appendix 8.1. Hence, the final optimisation problem is

$$\min_{\alpha^{(i)}} \mathcal{L}(\alpha^{(i)}) \tag{2.2.10}$$
$$\text{subject to: } 0 \leq \alpha^{(i)} \leq C$$
$$\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0.$$

Notice that the dual Lagrangian and optimisation problem are identical to the ones from the soft margin case, except for the addition of the $\alpha^{(i)} \leq C$ constraint. This can then be solved numerically for $\alpha^{(i)}$ using the SMO Algorithm, detailed in Appendix 8.2.

Again, the support vectors can be defined from the KKT Complimentary Slackness conditions. For the soft margin case, these are:

$$\alpha^{(i)} \left(y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) + \xi^{(i)} - 1 \right) = 0, \tag{2.2.11}$$
$$\beta^{(i)} \xi^{(i)} = (C - \alpha^{(i)}) \xi^{(i)} = 0. \tag{2.2.12}$$

From these, there are three cases:

- $\alpha^{(i)} = 0$, so by 2.2.12, $\xi^{(i)} = 0$. This is the case when the point lies outside of the margin, hence why there is no slack. In this case, $y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) > 1$
- $0 \leq \alpha^{(i)} \leq C$, so by 2.2.12 $\xi^{(i)} = 0$. This is the case where the point lies on the margin, hence why there is no slack. In this case, $y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) = 1$.
- $\alpha^{(i)} = C$, so by 2.2.12 $\xi^{(i)} > 0$. This is the case where the point lies over the margin, so the slack must be positive. In this case, $y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) < 1$.

This time the support vectors are any data points that lie on or over the margin, so any data points with $\alpha^{(i)} > 0$. Again, only these support vectors affect the position of the hyperplane, and all other points can be moved without issue, provided they are not moved on or over the margin.

As in the hard margin case, the optimal weights $\boldsymbol{\theta}$ can be calculated using Equation 2.2.6. The bias θ_0^* can be found using the fact that for any point i that lies on the margin, we have

$$y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) = 1, \tag{2.2.13}$$

which as before in the hard margin case, can be rearranged for

$$\theta_0^* = y^{(i)} - \boldsymbol{\theta}^{*\top} \mathbf{x}^{(i)}. \tag{2.2.14}$$

Finally, we can construct an algorithm for finding the SVM soft margin classifier:

1. Solve Equation 2.2.10 using the SMO Algorithm (Appendix 8.2) to find the support vectors.

2. Solve Equation 2.2.6 for $\boldsymbol{\theta}$, summing over the support vectors.

3. Solve Equation 2.2.14 for θ_0, where i is one of the support vectors.

4. Then the classifier is

$$\text{sign}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0). \tag{2.2.15}$$

We now have a working method to build a linear hyperplane to classify data into two classes. However, this is only beneficial if the data has a linear boundary between the two classes, which many datasets do not. In the next section, we will see how kernels can be utilised to extend the methodology developed in this section to non-linear problems.

3 Non - linearity in Data

To motivate this concept, it is helpful to think about the fact that data in the real world often exhibits complex, non - linear relationships. The linear decision boundaries introduced in the previous section, which are straight lines in two dimensions, or hyperplanes in higher dimensions, may indeed fail to capture these intricate patterns. Such situations where this may be the case include where you have circular or curved relationships between data points.

3.1 Kernels

We have seen that SVMs are very powerful tools when it comes to classification, but are mainly designed for when data exhibit linear relationships. However, functions called Kernels address this limitation by mapping input data into a higher - dimensional space. This is like looking at a flat piece of paper to examine a three - dimensional object. In doing so, SVMs can handle non - linear data far more effectively. Kernels are the functions that can help us achieve this. Primarily, this is done through a technique called the 'Kernel trick'. Essentially, rather than transforming the data explicitly into the higher dimensional space, which is usually a computationally expensive process, it takes a shortcut. The kernel trick allows SVMs to just focus on calculating similarities between data points, as opposed to finding out where every data point maps to in the higher - dimensional space. This is done by computing the dot product between points, thus achieving the kernel "trick".

3.1.1 Kernel Trick

In general, kernel methods aim to implicitly map our data into a higher - dimensional feature space, so that the linear machine learning algorithm can be used in this feature space, just as if the original data was linearly separable. In this case, since we know how to apply SVMs in a linear setting efficiently, we can access high, and sometimes infinite

dimensional feature spaces efficiently in the same way. We can then introduce a feature mapping $\boldsymbol{\phi}(\mathbf{x})$, which is a function of the original inputs.

Therefore, to acquire a more complex, non - linear decision boundary, we may want to apply the SVM algorithm to learn about the features $\boldsymbol{\phi}(\mathbf{x})$, as opposed to $\mathbf{x}$ in isolation. In doing so, we replace $\mathbf{x}$ everywhere it is previously mentioned, especially in Chapter 2 where the optimisation procedure is introduced.

To explore the effect on the optimisation problem to find the optimal classifier in Chapter 2, we use the more general case of the soft margin hyperplane, as there is no guarantee that the problem will still be linearly separable in the new space. The optimisation problem stays the same

$$\min_{\boldsymbol{\theta},\theta_0,\xi^{(i)}} \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{i=1}^{N}\xi^{(i)}$$

However, the constraints are now defined in the new space

$$y^{(i)}\left(\boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{x}^{(i)}) + \theta_0\right) \geq 1 - \xi^{(i)} \tag{3.1.1}$$

Which is a transformation of equation (2.2.2). Once the optimisation problem has incorporated this constraint transformation, the dual Lagrangian, equation (2.2.9), becomes

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N}\alpha^{(i)} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{\phi}(\left(\mathbf{x}^{(i)}\right))^\top\boldsymbol{\phi}(\mathbf{x}^{(j)}) \tag{3.1.2}$$

subject to: $0 \leq \alpha^{(i)} \leq C$ and $\sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0$, as before.

Thus, the main idea in kernel machines is to replace the inner product of the basis functions $\boldsymbol{\phi}\left(\boldsymbol{x}^{(i)}\right)^T$ and $\boldsymbol{\phi}\left(\boldsymbol{x}^{(j)}\right)$, by a kernel function $K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$, between data points in the original input space. Rather that mapping two instances $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ to the new, higher dimensional space and doing a dot product there, we can apply directly the kernel function in the original space. This can lead to greater computational efficiency, when implementing in practice.

3.1.2 Popular Kernels: Polynomial and Radial Basis Function

In the following section, let $\mathbf{x}$ and $\mathbf{y}$ denote data points/samples in the original space. An intuitive perspective of kernels would be that they can be interpreted as functions which measure similarity between data points $\mathbf{x}$ and $\mathbf{y}$. Therefore, when $\mathbf{x}$ and $\mathbf{y}$ are very similar, the kernel function will output a large value, and when they are dissimilar, the kernel will be small. In order for a kernel to be a valid kernel, it must satisfy some conditions, which are explored in Appendix 8.3.

Polynomial Kernel

In general, polynomial kernels, up to degree p, are of the form:

$$K(\mathbf{x}, \mathbf{y}) = \left(1 + \mathbf{x}^T\mathbf{y}\right)^p$$

Where p is a constant, the degree of the polynomial, selected by the user. For example, if we consider a quadratic mapping $\boldsymbol{\phi} : \mathbb{R}^2 \to \mathbb{R}^3$ that maps $\mathbf{x} = (x_1, x_2)^2$ as follows

$$\boldsymbol{\phi}(\mathbf{x}) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right)^T$$

The dot product between the mapping, for any two given data points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ is given as

$$\boldsymbol{\phi}(\mathbf{x})^T\phi(\mathbf{y}) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2$$

Then, we can re arrange the above to obtain the quadratic kernel function as follows:

$$\phi(\mathbf{x})^T\phi(\mathbf{y}) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2 = (x_1y_1 + x_2y_2)^2 = \left(\mathbf{x}^T\mathbf{y}\right)^2 = K(\mathbf{x}, \mathbf{y}) \qquad (3.1.3)$$

We can therefore see that the dot product in the higher dimensional feature space can be computed by evaluating the kernel in the input space, without explicitly mapping the points into the feature space. An example justification is explored in Appendix 8.3. This is the Kernel Trick.

The below figure demonstrates the margins found by a polynomial kernel of degree 2. The circled data points are the support vectors.

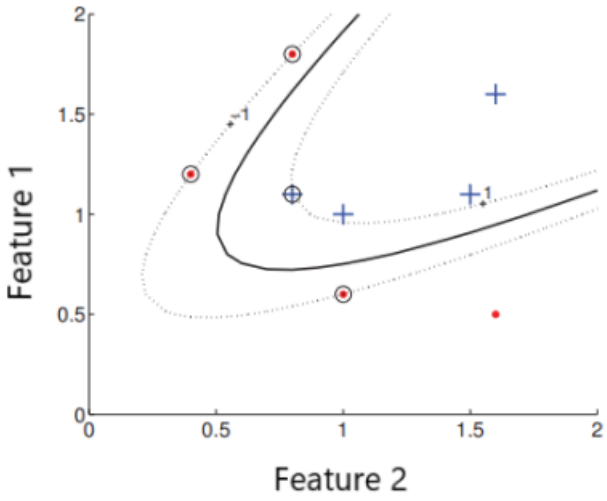

Figure 5: The classifier and margins found by a polynomial kernel of degree 2. Circled instances are the support vectors.

Consider the previous example above with the kernel in equation 3.1.3. In the transformed, higher dimensional feature space, we map

$$x_1, x_2 :\rightarrow z_1, z_2, z_3$$

$$z_1 = \sqrt{2}x_1x_2 \quad z_2 = x_1^2 \quad z_3 = x_2^2$$

The left pane plot in Figure 6 demonstrates the non - linearity of data points. The features are x_1 and x_2. Therefore, in order to find a suitable classifier to separate the points, the data is projected into a higher dimensional space, as shown on the right pane. On this plot, the data has been transformed in a quadratic fashion, and the hyperplane is plotted accordingly. See Appendix 8.3 for corresponding code for the plot.

3.1.3 Gaussian Radial Basis Function (RBF) Kernel

In general, Gaussian Radial Basis kernels are of the form:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right) = \exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{y}\|^2\right) \tag{3.1.4}$$

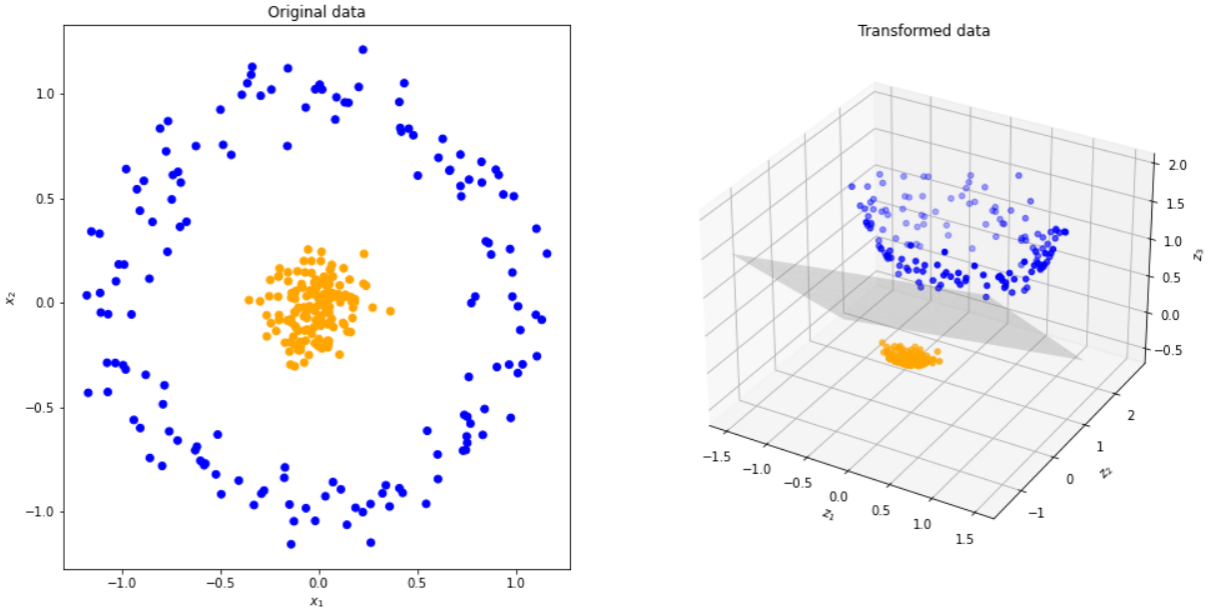

Figure 6: **Left pane**: Original data plot, exhibiting non - linear relationship. **Right pane**: Transformed data plot, after quadratic (polynomial) transformation, with a hyperplane

Where σ^2 is the bandwidth parameter. More specifically σ denotes the kernel width. This primarily determines the feature space on which the samples will be mapped, and heavily influences the accuracy of the classification task. As $\sigma \to 0$, all training samples can be classified in the correct fashion, but generalisation of the learning machine becomes poor, and the SVM will reduce in its ability to classify new samples; as $\sigma \to \infty$, the entire training sample set is classified as one class. This property is justified mathematically in Appendix 8.3.

The best value for σ^2 is found by cross - validation. Note that when there are two parameters to be optimised using cross validation, for instance C and σ^2, one should do a grid (factorial) search in the two dimensions.

We can observe an implementation of Gaussian RBF kernels in Figure 7.

Suppose we had a large feature space. How can we prevent overfitting in practice when implementing a kernelised SVM? In general, when the SVM algorithm runs, it seeks a

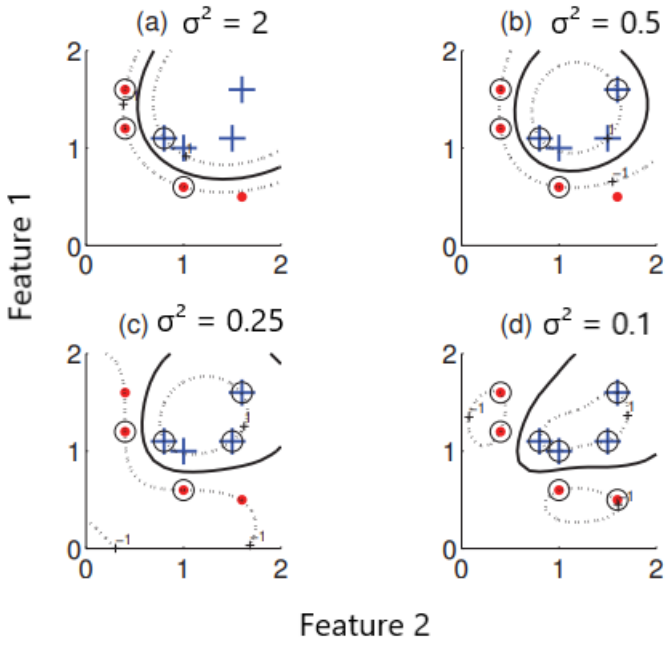

Figure 7: The boundary and margins obtained by the Gaussian kernel with different bandwidth values σ^2. As the bandwidth gets larger, the boundaries get smoother.

solution with a large margin. Large margins can lead to good generalisation. To understand where this comes from, small margins generally draw highly complex decision boundaries taking many turns. Large margins would not allow this, as the margin would not fit between the data points that you are trying to "move around". Therefore, as the decision boundary increases in complexity, it will also increase in its ability to represent where single data points lie in the training data, and less likely to be representative of data that is not in the training set.

Now that we have seen how kernels can extend SVMs to higher dimensional spaces to tackle more complex problems, we will explore this further with a real life example in the next section.

4 Comparison and Model Evaluation

In this section, we compare the performance of Support Vector Classifiers with other linear and non-linear models on a real dataset. We use the k-fold cross-validation technique to tune the hyper-parameters and compare all the models. This ensures our results are robust and reproducible.

4.1 Breast Cancer Database

To test the performance of SVMs and other classifiers, we use the breast cancer database, which is publicly available at the UC Irvine Machine Learning repository. The data consists of 569 suspected cancer tumors, and 30 features are "computed from a digitized image of a fine needle aspirate (FNA) of a breast mass" [13].

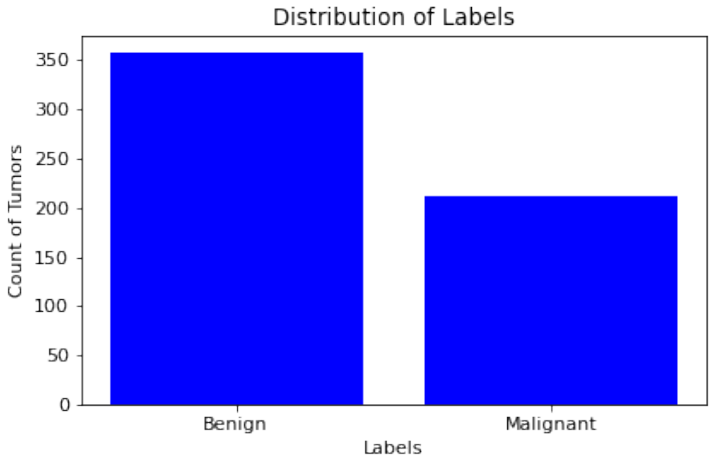

Figure 8: Distribution of 569 tumors by their labels: Malignant and Benign

These features measure the various characteristics of the cell nuclei like Radius, Area Textures, etc extracted from the image. A list of all the features is detailed in the appendix Table 3. Each tumor has a binary label of 0 and 1 where the label 0 implies the tumor is malignant and label 1 means the tumor is benign. The classes of malignant and benign tumors are almost balanced in a 40-60 split as illustrated in Figure 8.

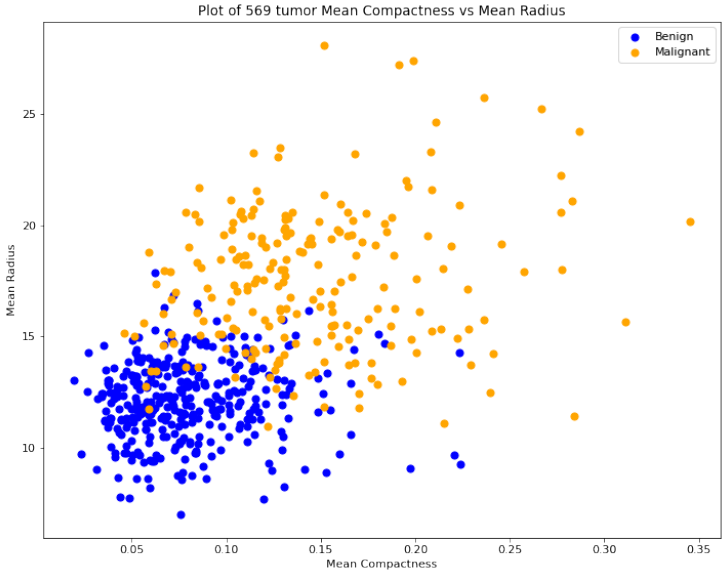

Figure 9: Plot of Mean Radius vs Mean Compactness

Figure 9 shows the dataset in 2 dimensional space of Mean Radius and Mean Compactness measurement. We can see the two classes are overlapping in some regions and therefore it is a non-separable case. Furthermore, in Figure 10 we can see the distribution of the first 10 features in the dataset. The two classes clearly exhibit different data distributions for many of the features. Next, to find the best model we need to define the model evaluation criteria.

4.2 Performance Metrics

Data classification tasks can be classified into binary, multiclass and multi-labelled classification [14]. We discuss some of the common metrics used in binary classification tasks since it is the focus of our report. Performance metrics are useful for two main purposes:

1. Comparing the performance of two different models,

2. Tuning model parameters [15].

In general, the metric choice is determined by the type of problem, the distribution of the data and the goal of the project. Some of the common metrics are:

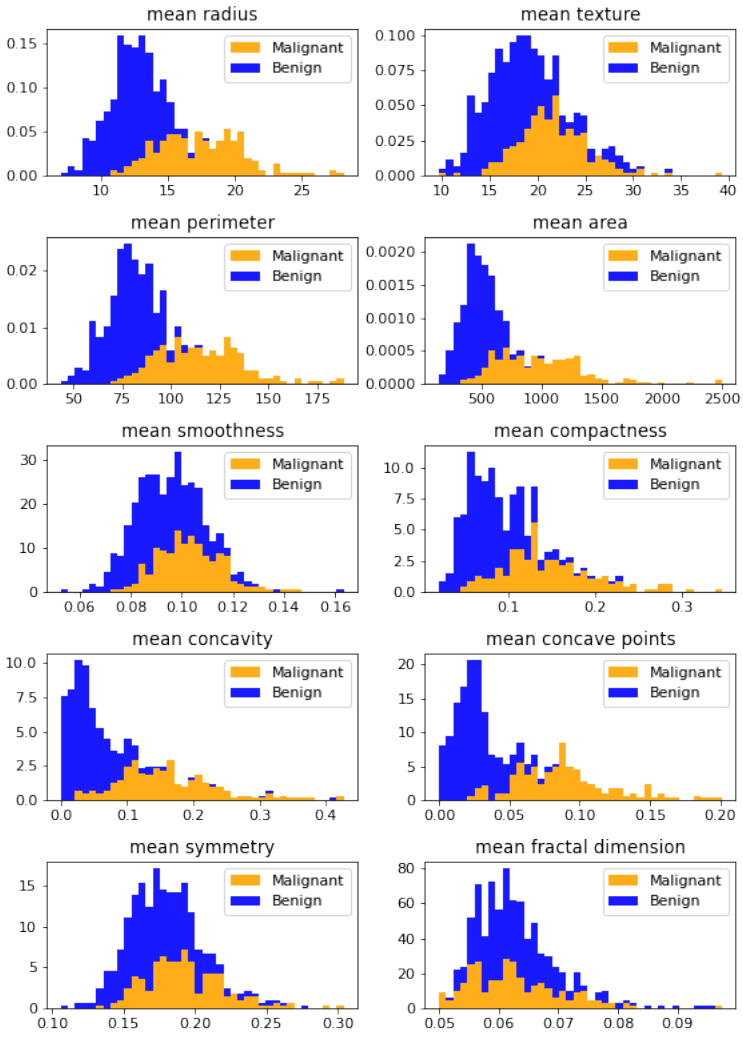

Figure 10: Histogram of first 10 features of Breast Cancer Dataset

1. **Confusion Matrix:** For binary classification, the confusion matrix is a 2 x 2 matrix which compares the predicted class from the trained model against the actual classes in the dataset as shown in Figure 11.

2. **Precision:** The precision value tells us, out of all the points predicted as positive by the model, how many of them were actually positive. In other words, how precise our predictions on a class are.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2.1}$$

		Predicted Class	
		Normal	Attack
Actual Class	Normal	True Negative (TN)	False Positive (FP)
	Attack	False Negative (FN)	True Positive (TP)

Figure 11: Example of confusion matrix for binary classification

3. **Recall:** The recall tells us, out of all the points which had an actual class as true, how many of them were correctly predicted as true by the model.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.2.2}$$

4. **Accuracy:** The accuracy is a ratio of all the labels correctly predicted by the model, whether positive or negative, by the total number of predictions made.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2.3}$$

5. **F1-Score:** This is calculated as the harmonic mean of the precision and recall defined above. It is a useful measure when precision and recall are equally important and we need to balance the trade-off.

$$\text{F1-Score} = 2 \cdot \left(\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \right) \tag{4.2.4}$$

The accuracy metric can sometimes be misleading and biased towards the majority class if the classes are imbalanced. In our dataset the class labels are nearly balanced and it is important to predict both malignant and benign tumors accurately hence we chose accuracy as performance metric. Next, we look at the use of cross validation techniques to get reliable model metrics from our test data.

4.3 K-fold Cross Validation

Typically for regression and classification problems, the data is split into two parts: the training data for training the model and the testing data for estimating the model performance on unseen data. However, with only one fixed train and test data set, the model's performance can be sensitive to the way the split is made. K-fold Cross-validation addresses this by generating K chunks of input data and fitting the the model K-times, in each iteration on $(K-1)$ chunks and testing on the left out chunk to estimate the performance metric.

We used a 5-fold cross validation approach which leads to almost 100 data points in each fold. The **scikit-learn** library in Python is used, specifically the **"KFold"**, **"cross_val_score"** and **"GridSearchCV"** classes for performing cross-validation and hyper-parameter tuning. The Python code for cross-validation of our models can be found in Appendix 8.4.5.

4.4 Comparison with Linear Models

A 5-fold cross validation technique is used to compare and tune our model hyper parameters. We compare the support vector classifier with a linear kernel against some modules including Naive Bayes, Linear Discriminant Analysis and Logistic Regression. The models are trained on all the 30 input features without any feature selection. The results are summarised in the table below.

All 4 models give a good accuracy of above 90%, among them the Naive Bayes has the lowest accuracy of 92.9%. LDA performs better than the baseline Naive Bayes with 95.6% accuracy. The logistic regression model further improves the accuracy to 97.2%. The optimal regularization parameter (C) for an SVM with a linear kernel was found using the 5-fold cross-validation technique do be at $C = 0.1$. This model gave the best accuracy among the linear models, which was 97.5%.

Table 1: Linear Model Comparison

Model	**Accuracy**
Naïve Bayes	0.929
LDA	0.956
Logistic	0.972
SVM (Linear)	0.975

4.5 Comparison with Non-Linear Models

As could be seen when we visualized the data, the classes were overlapping and the decision boundary could be non-linear. Therefore, we fit a Quadratic Discriminant Analysis (QDA) model. This should theoretically give us more complex and non-linear decision boundaries which should lead to better accuracy. We utilize non-linear kernels such as a Polynomial Kernel and an RBF Kernel and use 5-fold cross validation to tune the respective parameters and finally compare the model accuracy.

Table 2: Non-Linear Model Comparison

Model	**Accuracy**
QDA	0.965
SVM (Polynomial)	0.965
SVM (RBF)	0.979

QDA has a regularisation parameter C that was tuned using cross-validation and the best model accuracy was found at $C = 0.1$. For the SVM with a polynomial kernel, 3 parameters had to be tuned: the regularisation parameter C, the degree of the polynomial and γ, with γ defined as in equation 3.1.4. The optimal parameters found after cross-validation were $C = 0.01, \text{degree} = 3, \gamma = 0.5$ Similarly, the RBF kernel has two parameters that we tuned and the optimal parameters found were $C = 2, \gamma = 0.04$.

Table 2 shows that both QDA and the SVM with a Polynomial kernel have the same accuracy of 96.5%. However, when we use the RBF kernel, we get an accuracy of 97.9% which is the best among all the linear and non-linear models we tried. Therefore, the use of kernels helped us to model the complex non-linear relationships between our data points and it gives us the optimal model.

5 SVMs: Assumptions and Limitations

Support Vector Machines, although accurate, are not preferred in applications requiring great classification speed due to the large number of support vectors that need to be found, since the decision boundaries are decided by the support vectors. Several factors impact the complexity of a Support Vector Machine [17].

1. Kernel Choice. SVMs use both linear and nonlinear (polynomial, radial basis function, Gaussian, etc.) kernels and therefore the choice of the kernel affects the computational and time complexity of the classifier in mapping input data to higher-dimensional spaces. With complex nonlinear kernels the model complexity increases as well.

2. Kernel Parameters. Kernel performance depends on kernel parameters. For example, for the Radial Basis Function (RBF) kernel, the scale and shape parameters are crucial in the effectiveness of the kernel in estimating the decision hyperplane(s). For large datasets with high volatilities, this can lead to more complex models and corresponding time and space complexity.

3. Regularisation Parameter (C). This controls the trade-off between the model accuracy and the complexity of the decision boundary. With large values of C, we can achieve a more accurate but complex model that could have explainability issues. Whereas for small , the simpler model would not be able to generate an accurate decision boundary.

4. Dataset Size. With hundreds to thousands of parameters in the model, time and space complexity become issues for SVM classifiers. In datasets with up to, say 15,000 instances, the full kernel matrix is stored using cache, this significantly reduces the time complexity for hyperparameter tuning. The cache approach can store those values from the kernel matrix that are not used for decision boundaries while working with support vectors.

5. Class Imbalance. If there is a serious bias between classes in a classification problem, the resulting classifier can be more complex than one with balanced classes due to working with nonlinear kernels and large regularisation values [18].

6 Connections to modern literature

Support Vector Machines are highly suitable for classification and regression problems from a variety of different fields. SVMs are also equipped to compute on high-dimensional datasets, making them ideal for areas such as bioinformatics and computational linguistics.

6.1 Support Vector Machines for Cancer Genomics

Support Vector Machines have been used in the cancer classification process ever since high-throughput microarray gene expression data became available in the early 2000s [19]. The enormous dimensionality of the data in genomics makes it difficult to develop an appropriate classifier. The relatively small sample size (n = 20 to 50 patients) of these experiments is another challenge. Initially, Golub [20] tried to utilise a linear SVM to separate two different types of leukemia based on gene expression microarray data. A simple learning method called weighted voting was developed to discriminate between two recognized (labeled) forms of leukemia. The Affymetrix Hgu6800 hardware chips covered a total of 7129 gene expression probes (gene features), and the weighted contribution of each entire gene feature to the two groups was calculated. SVMs have proven to perform better when it comes to classifying high-dimensional (gene features) and low-sample-size data (for example, n= 20 to 60 patients). Vapnik and Mukherjee [21] improved the weighted voting mechanism of the SVM to lower the error rate from approximately 6 percent (2 errors out of 34) to 0 percent.

Nevertheless, feature selection was not carried out in this study before the model was created. SVM classifiers perform feature selection to extract differential expression genes among the thousands of genes in the collection.
Certain tissues that are associated with colon cancer are also classified using SVM classifiers. Moler [22] employed a set of twenty-two normal colon tissues and forty colon cancer tumors.

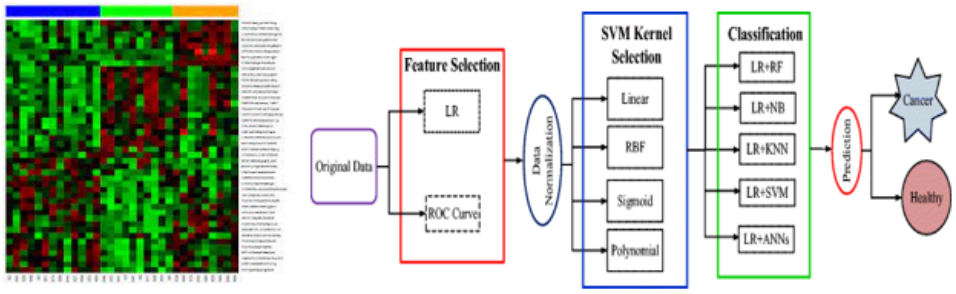

Figure 12: High Throughput data Genomic data and the SVM pipeline approach

First, a feature selection metric based on the likelihood that a patient has (or does not have) colon cancer was proposed: the Naïve Bayes relevance score.

6.2 Support Vector Machines in Financial Markets

Support Vector Machines have several applications in financial markets, such as stock exchange price prediction and credit risk management [23], [24], [25]. In this section, we will look at one example about loan applications.

To raise the standard of living for the population, banks are businesses that receive funds from the community in the form of savings and then disburse them back to the community in the form of credit or other means. To get credit, one must apply and submit a proposal to the bank. When deciding to approve or reject a loan proposal, the regulators take into account five factors: the customer's character (personality), capacity (ability to repay the debt), capital (capital of the customer), collateral (credit risk guarantee), and condition (financial standing of the customer).

If a consumer has poor credit and is therefore unable to make payments on time, there may be an increase in credit risk. The SVM algorithm has been the subject of some studies to categorise potential clients of lending organisations based on their ability to repay loans or not. Accuracy and misclassification metrics, such as Area under the Characteristic curve

(AUC), F1-Score, accuracy, sensitivity, specificity, and precision, are utilised to evaluate the model. In addition to credit status, which was the dependent variable, the independent factors included gender, interest rate, duration of time, employment, income, principal amount, warranty, and loan history. Global companies such as Experian work on credit risk modelling for individual citizens, while Standard & Poor$'s$ 500 and Moody's work at the macro-economic level and cover credit risk modelling for companies and countries. Both the micro and macro level financial data classification problems include the use of support vector machines in binary and multi-class classification problems. Multiple SVM models with varying kernels and cross-validation approaches are tested before finding the optimal model. Data preparation also impacts the performance of SVM models [26].

6.3 Support Vector Machines in Image Processing

Like other application areas, SVMs are widely used in image processing and computer vision. There are a wide variety of topics in image processing, such as image segmentation, object detection, and image classification. The following are some essential details about the application of SVMs in image processing [27] [28]:

Image segmentation: SVMs can be used to categorise and classify images into several groups. The SVM may be trained to recognise various items or patterns within the images by using a set of labeled images. With a large training dataset, SVMs can implement separation between the objects. Being a supervised learning model, the model is presented with a number of different objects, and the SVM generates hyperplanes to segment them. Challenges include object scale, orientation, pixels, shades, and similar details [29].

Object identification: SVMs are also used to identify objects in pictures. Their ability to recognize particular items or areas of interest within an image can be trained, which make them valuable in computer vision and in autonomous systems applications (see [30]).

Feature Extraction: SVMs are suited for image processing jobs that require extracting

and interacting with a large number of features since they frequently perform well with high-dimensional data. Extracting features is essential to deciphering an image's content [31].

Image Segmentation: The objective of image segmentation is to divide an image into meaningful segments or areas. SVMs can be used for this task. This has applications in remote sensing, medical imaging, and similar areas [32].

Due to the Kernel trick, SVMs can implicitly map the input data into a high-dimensional space. When working with intricate linkages and non-linear patterns in photographs, this is advantageous. It is crucial to remember that, despite their strength, SVM model performance can be affected by high dimensionality, including the quality of the training set, kernel selection, and parameter tuning. When using SVMs for image processing, image processing professionals frequently pre-process the images, extract relevant features, and fine-tune the parameters to get the best results for the given task.

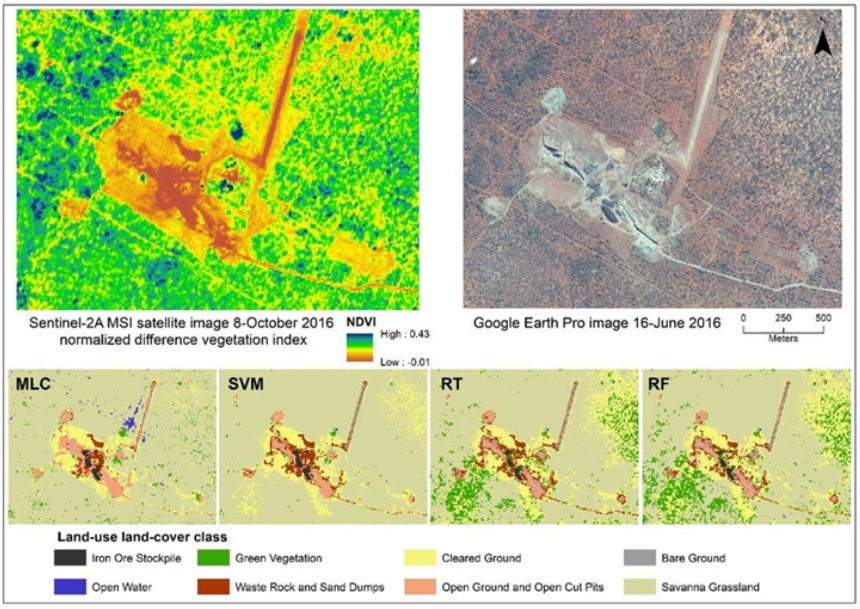

Figure 13: Vegetation area detection in Satellite Images using SVM. Feature Extraction/ subset selection using SVM.

7 Conclusion

7.1 Summary

We have shown that Support Vector Machines are an effective tool for tackling binary classification problems. The main results of this paper are:

- SVMs work by maximising the margin and minimising the slack. This idea can be formulated as a dual Lagrangian optimisation problem, which can be solved numerically using the SMO algorithm.
- For data with non-linear relationships, the Kernel Trick can be used to efficiently project data into higher dimensional spaces such that a linear hyperplane is sufficient to separate this data in the new space.
- SVMs can outperform other machine learning models in binary classification problems, and are still used today to tackle classification problems in a variety of different fields.

Finally, we will reflect upon the work put into this paper and what topics would be worth investigating in the future.

7.2 Critical Reflection

7.2.1 Linear data

Overall, this section of the report was successful in what it set out to do: to explore the mathematical foundations behind SVMs and motivate their use. We showed that the problem can be transformed into a numerically solvable optimisation problem using the dual Lagrangian for both the soft margin and hard margin cases.

If this was done again, it could be useful to have an exploration into the different heuristics used to select the α parameters in the SMO algorithm, so that readers have all

of the information necessary to implement SVMs entirely from scratch after reading this section of the report.

7.2.2 Non - linear data

Whilst two highly popular kernels were explored in the report, it would have been beneficial to explore additional kernels used in practice to appreciate a greater breadth of working with non - linear data. Examples include, but are not limited to, the Chi - Squared Kernel, which is commonly used in computer vision tasks, specifically for object recognition and image classification; the String Sub - sequence Kernel, which is useful when dealing with text data or in particular data which can be represented as sequences. This is most commonly used in Natural Language Processing (NLP) tasks, which is a growing sub - field of machine learning, and Graph Kernels, such as the Graphlet Kernel and Random Walk Kernel. The use cases for this kernel include solving problems in graph - structured data, particularly social network analysis. Furthermore, the general class of RBFs as well as custom kernels implemented in practice would have been beneficial to explore.

More applied examples may have been beneficial, where various kernel functions could have been applied to real data sets. However, this would have vastly increased the length of the report.

7.2.3 Comparison to other models

A real database on cancer tumor prediction was used to successfully perform a comparison study between SVMs and other frequently used classifiers. Cross validation was employed to ensure robust and accurate comparison. However, there were other classification models that we did not include in our comparison study, such as tree based models, for instance. These models could have performed better than all the models.

It would have also been additionally informative to see the results using other performance metrics such as F1-score or recall. The optimal parameters found during cross validation

may have differed for different performance metrics. Their behaviour could be analysed and compared.

Finally, we did not perform any feature selection. This may have had an impact on the final model performance. However, feature selection is a significant topic which might have deviated the focus of the report away from the main topic of SVMs, the kernel methods and comparison to other models.

7.3 Suggestions for future work

For future work, it would be interesting to explore how SVMs can be extended to multiclass classification problems. This could include explanations of the one-to-one and one-to-many approaches, and a comparison between these approaches to assess the benefits and drawbacks of each. Other future directions include how Support Vector Machine ideas can be implemented in deep learning approaches. In addition to this, also exploring improvements in quantum Support Vector Machines (QSVMs). Besides classification, extending SVMs to regression problems can be yet another area of potential future work.

8 Appendices

8.1 Deriving the Dual Objective Function

We have the Lagrangian

$$\mathcal{L}(\boldsymbol{\theta}, \theta_0, \alpha^{(i)}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + \sum_{i=1}^{N} \alpha^{(i)}[1 - y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0)], \tag{8.1.1}$$

as well as the following first-order conditions

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \boldsymbol{\theta} - \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} = 0 \implies \boldsymbol{\theta} = \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}, \tag{8.1.2}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = -\sum_{i=1}^{n} \alpha^{(i)} y^{(i)} = 0. \tag{8.1.3}$$

and aim to derive the following dual objective function

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^\top \mathbf{x}^{(j)}. \tag{8.1.4}$$

We start by expanding the brackets within the summation in 8.1.1.

$$\mathcal{L}(\boldsymbol{\theta}, \theta_0, \alpha^{(i)}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + \sum_{i=1}^{N} \alpha^{(i)} - \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \boldsymbol{\theta}^\top \mathbf{x}^{(i)} - \theta_0 \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \tag{8.1.5}$$

By 8.1.2, the last summation is 0. Also, we can substitute in 8.1.2 to find:

$$\mathcal{L}(\boldsymbol{\theta}, \theta_0, \alpha^{(i)}) = \frac{1}{2} \left\| \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \right\|^2 + \sum_{i=1}^{N} \alpha^{(i)} - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^\top \mathbf{x}^{(j)}. \tag{8.1.6}$$

Finally, by using the definition of the two-norm, we find that the first term on the right of 8.1.6 is equivalent to half of the double summation in the third term. Hence, we find the dual objective

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^\top \mathbf{x}^{(j)}. \tag{8.1.7}$$

For the hard margin case, we start with the Lagrangian

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \theta_0, \xi^{(i)}, \alpha^{(i)}, \beta^{(i)}) = &\frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{i=1}^{N}\xi^{(i)} \\ &- \sum_{i=1}^{N}\alpha^{(i)}\left(y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0\right) + \xi^{(i)} - 1) \\ &- \sum_{i=1}^{N}\beta^{(i)}\xi^{(i)}, \end{aligned} \tag{8.1.8}$$

and the additional constraint

$$C - \alpha^{(i)} - \beta^{(i)} = 0. \tag{8.1.9}$$

From these and 8.1.2, 8.1.3, we aim to derive the same dual objective 8.1.4. This case reduces directly to the hard margin case, since

$$C\sum_{i=1}^{N}\xi^{(i)} - \sum_{i=1}^{N}\alpha^{(i)}\xi^{(i)} - \sum_{i=1}^{N}\beta^{(i)}\xi^{(i)} = \sum_{i=1}^{N}(C - \alpha^{(i)} - \beta^{(i)})\xi^{(i)} = 0, \tag{8.1.10}$$

where we have used constraint 8.1.9 to obtain the last equality. Hence, these terms drop directly out of 8.1.8, reducing this to 8.1.1 so it can be solved in the same way as the soft margin case.

8.2 SMO Algorithm

The Sequential Minimal Optimisation Algorithm is a method of solving the optimisation problem [**?**]:

$$\min_{\alpha^{(i)}} \mathcal{L}(\alpha^{(i)}) \tag{8.2.1}$$

$$\text{subject to: } 0 \leq \alpha^{(i)} \leq C$$

$$\sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0.$$

where

$$\mathcal{L}(\alpha) = \sum_{i=1}^{N}\alpha^{(i)} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}(\mathbf{x}^{(i)})^\top\mathbf{x}^{(j)}. \tag{8.2.2}$$

The algorithm works by iterating over and optimising pairs $\alpha^{(i)}, \alpha^{(j)}$, whilst assuming all other $\alpha^{(k)}$ are constant. We will denote the selected pair as $\alpha^{(1)}$ and $\alpha^{(2)}$. The second constraint allows us to build a linear formula for $\alpha^{(1)}$ and $\alpha^{(2)}$, since we have

$$\alpha^{(1)}y^{(1)} + \alpha^{(2)}y^{(2)} = -\sum_{i=3}^{N} \alpha^{(i)}y^{(i)} = k, \tag{8.2.3}$$

where k is the value of the sum in the middle of the equalities. As such, we have the following results:

$$\begin{gathered} y_1 \neq y_2 \text{ implies } \alpha^{(1)} - \alpha^{(2)} = \alpha^{*(1)} - \alpha^{*(2)} = k \\ y_1 = y_2 \text{ implies } \alpha^{(1)} + \alpha^{(2)} = \alpha^{*(1)} - \alpha^{*(2)} = k \\ \text{Or more compactly: } \alpha^{(1)} + s\alpha^{(2)} = \alpha^{*(1)} + s\alpha^{*(2)} = k \end{gathered} \tag{8.2.4}$$

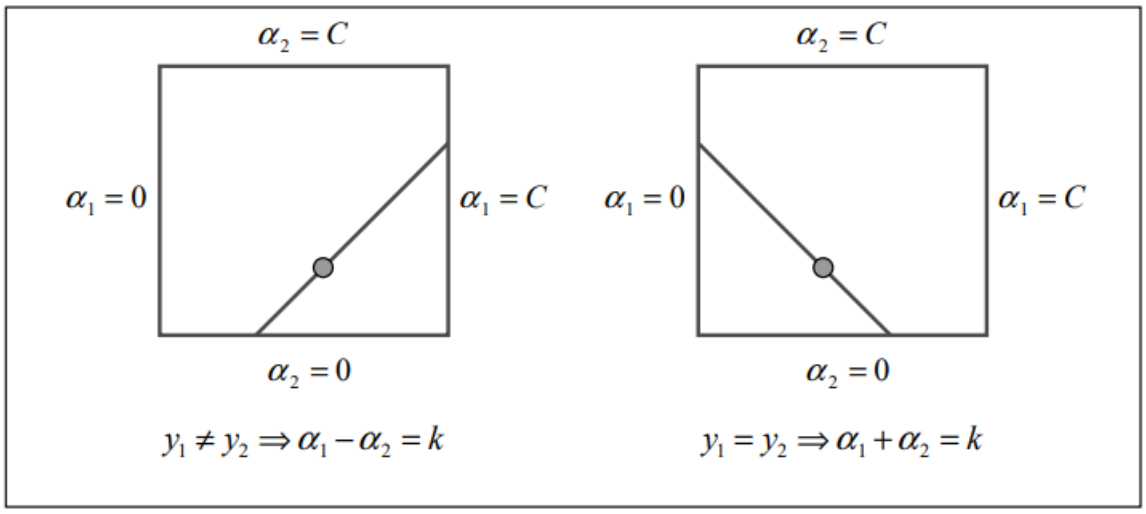

Figure 14: The values of $\alpha^{(1)}, \alpha^{(2)}$ are constrained by $0 \leq \alpha^{(1)}, \alpha^{(2)} \leq C$, creating the box. Also, they must satisfy the linear equality constraint, so they must stay on the line.

where $s = y^{(1)}y^{(2)}$ and the values with a * denote their values during the previous iteration. This, along with the first constraint $0 \leq \alpha^{(1)}, \alpha^{(2)} \leq C$, are shown in Figure 14. The values $\alpha^{(1)}, \alpha^{(2)}$ must be within the box due to the inequality constraint, and must also be within the line due to the linear equality constraint. Thus, we solve for the values

of $\alpha^{(1)}, \alpha^{(2)}$ that minimise 8.2.1 along this section of the line, whilst keeping all other $\alpha^{(k)}$ constant. We can find $\mathcal{L}$ exclusively in terms of $\alpha^{(2)}$ by substituting in $\alpha^{(1)} = k - s\alpha^{(2)}$ into 8.2.2. Since this is a quadratic function, this can be solved in the usual way, by taking the derivative with respect to $\alpha^{(2)}$ and setting it equal to 0. Then, by substituting in $k = \alpha^{*(1)} + s\alpha^{*(2)} = k$, we can rearrange for a formula for $\alpha^{(2)}$ in terms of $\alpha^{*(2)}$. This gives the formula

$$\alpha^{(2)} = \alpha^{*(2)} + \frac{y_2(E^{(1)} - E^{(2)})}{\eta}, \tag{8.2.5}$$

where $E^{(i)} = \hat{y}^{(i)} - y^{(i)}$ is the error in the prediction for data point i using the previous model, and $\eta = \mathbf{x}^{(1)\top}\mathbf{x}^{(1)} + \mathbf{x}^{(2)\top}\mathbf{x}^{(2)} + 2\mathbf{x}^{(1)\top}\mathbf{x}^{(2)}$ is the second derivative of $\mathcal{L}$ along the line in 8.2.4. Note that in the linear case, η will always be negative since $\mathcal{L}$ is always positive definite. If a Kernel is applied, η could be negative, in which case this method must be slightly modified.

This value of $\alpha^{(2)}$ will be along the line 8.2.4, but may not necessarily be within the box given by the $0 \leq \alpha^{(1)}, \alpha^{(2)} \leq C$ constraint. There are two cases for the bounds of $\alpha^{(2)}$:

- If $y^{(1)} = y^{(2)} : L = \max(0, \alpha^{(2)} - \alpha^{(1)})$ and $H = \min(C, C + \alpha^{(2} - \alpha^{(1)})$,
- If $y^{(1)} \neq y^{(2)} : L = \max(0, \alpha^{(2)} + \alpha^{(1)} - C)$ and $H = \min(C, \alpha^{(2} + \alpha^{(1)})$.

Hence, we can define the 'clipped' value of $\alpha^{(2)}$ as

$$\alpha^{(2)}_{\text{clipped}} = \begin{cases} H & \text{if } \alpha^{(2)} \geq H, \\ \alpha^{(2)} & \text{if } L < \alpha^{(2)} < H, \\ L & \text{if } \alpha^{(2)} \leq L. \end{cases} \tag{8.2.6}$$

Finally, $\alpha^{(1)}$ can be computed from this clipped value of $\alpha^{(2)}$ using 8.2.4:

$$\alpha^{(1)} = \alpha_1 + s(\alpha^{*(2)} - \alpha^{(2)}_{\text{clipped}}). \tag{8.2.7}$$

Note that $\alpha^{(1)}$ will already be clipped.

This method can then be used to iterate over pairs of αs until convergence. Heuristics can be used to select the $\alpha^{(1)}, \alpha^{(2)}$ pairs to improve the speed of convergence. However, as long as one of the $\alpha^{(i)}$ chosen violates the KKT conditions, it is guaranteed that this method will decrease the objective function by Osuna's Theorem [**?**].

8.3 Kernels

8.3.1 Necessary and sufficient condition for kernels

The following is a necessary and sufficient condition in order for a function to be a valid kernel. Let G be the Kernel matrix, or Gram matrix, which is of size $m \times m$, and where each i, j entry corresponds to $G_{i,j} = K\left(x^{(i)}, x^{(j)}\right)$ of the dataset $\mathbf{X} = \left\{x^{(1)}, \ldots, x^{(m)}\right\}$

The function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a valid kernel if and only if the matrix G is symmetric, positive semi - definite. This is a variation of Mercer's Theorem.

8.3.2 Justification of polynomial kernel example

For example, we may have

$$\boldsymbol{\phi}(\mathbf{x}) = \left(5.9^2, 3^2, \sqrt{2} \cdot 5.9 \cdot 3\right)^T = (34.81, 9, 25.03)^T$$
$$\boldsymbol{\phi}(\mathbf{y}) = \left(6.9^2, 3.1^2, \sqrt{2} \cdot 6.9 \cdot 3.1\right)^T = (47.61, 9.61, 30.25)^T$$
$$\boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{y}) = 34.81 \times 47.61 + 9 \times 9.61 + 25.03 \times 30.25 = 2501$$

And can see that the quadratic kernel gives us the exact same value:

$$K(\mathbf{x}, \mathbf{y}) = \left(\mathbf{x}^T \mathbf{y}\right)^2 = (50.01)^2 = 2501$$

8.3.3 Justification of kernel width property

The Hilbert space distance in a high dimensional feature space of any two samples, is represented as

$$\|\phi(\mathbf{x}) - \phi(\mathbf{y})\|^2 = k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{y}) + k(\mathbf{y}, \mathbf{y}) \tag{8.3.1}$$

. When $\sigma \to 0$, it is easy to find from equation 3.1.4 that

$$k(\mathbf{x}, \mathbf{x}) = k(\mathbf{y}, \mathbf{y}) = 1 \tag{8.3.2}$$

And equation 8.3.1 becomes

$$k(\mathbf{x}, \mathbf{y}) = 0$$

Then we have

$$\|\phi(\mathbf{x}) - \phi(\mathbf{y})\|^2 = 2 \tag{8.3.3}$$

Which indicates that as $\sigma \to 0$, the distance between any two data points in the feature space is $\sqrt{2}$. Samples from the same class will not gather, and each sample will then be classified as one single class, so then all the training data is correctly classified. However, this SVM is over fitting, and will fail to generalise.

When $\sigma \to \infty$, equation 3.1.4 becomes

$$k(\mathbf{x}, \mathbf{x}) = k(\mathbf{y}, \mathbf{y}) = 1 \tag{8.3.4}$$

And thus, equation 8.3.1 can be simplified, and then we obtain

$$\|\phi(\mathbf{x}) - \phi(\mathbf{y})\|^2 = 0 \tag{8.3.5}$$

Then, equation 8.3.5 implies that, when $\sigma \to \infty$, samples that have been mapped to the feature space become the same point, and then the distance of any two samples is 0. This means all samples will be classified as one class, and the SVM cannot distinguish between the training samples.

8.3.4 Figure 6 Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_circles
from sklearn import svm
from mpl_toolkits.mplot3d import Axes3D

def feature_map(X):
    return np.asarray((X[:, 0], X[:, 1],
    X[:, 0] ** 2 + X[:, 1] ** 2)).T

X, y = make_circles(300, factor=.1, noise=.1, random_state=0)
Z = feature_map(X)

Z[y == 0, 1] += 1.5

fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(1, 2, 1)
ax.scatter(X[:, 0], X[:, 1],
c=['blue' if label == 0 else 'orange' for label in y])
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_title('Original data')

ax = fig.add_subplot(1, 2, 2, projection='3d')
ax.scatter3D(Z[:, 0], Z[:, 1],
```

```
Z[:, 2], c=['blue' if label == 0 else 'orange' for label in y])
ax.set_xlabel('$z_1$')
ax.set_ylabel('$z_2$')
ax.set_zlabel('$z_3$')
ax.set_title('Transformed data')

clf = svm.SVC(C=1, kernel='linear')
clf.fit(Z, y)

xx, yy = np.meshgrid(np.linspace(-1.5, 1.5, 100),
np.linspace(-1.5, 1.5, 100))
zz = (-clf.intercept_ -
clf.coef_[0, 0] * xx - clf.coef_[0, 1] * yy) / clf.coef_[0, 2]

trisurf = ax.plot_trisurf
(xx.flatten(), yy.flatten(), zz.flatten(),
color='lightgray', alpha=0.3,
linewidth=0)
trisurf._facecolors2d = trisurf._facecolors3d
trisurf._edgecolors2d = trisurf._edgecolors3d

plt.show()
```

8.4 Model Evaluation and Comparison

8.4.1 Breast Cancer Dataset feature names

mean radius	mean texture	mean perimeter	mean area
mean smoothness	mean compactness	mean concavity	mean concave points
mean symmetry	mean fractal dimension	radius error	texture error
perimeter error	area error	smoothness error	compactness error
concavity error	concave points error	symmetry error	fractal dimension error
worst radius	worst texture	worst perimeter	worst area
worst smoothness	worst compactness	worst concavity	worst concave points
	worst symmetry	worst fractal dimension	

Table 3: Name of 30 features in Breast cancer dataset

8.4.2 Figure 8 Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.discriminant_analysis import (
    LinearDiscriminantAnalysis,
    QuadraticDiscriminantAnalysis,)
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

breast_cancer_df = load_breast_cancer()
breast_cancer_X = pd.DataFrame
(data=breast_cancer_df.data,
columns=breast_cancer_df.feature_names)
breast_cancer_X.head()

breast_cancer_Y = pd.DataFrame
(data=breast_cancer_df.target,
columns=['Diagnosis'])
breast_cancer_Y.value_counts()

fig, ax = plt.subplots()

ax.bar(['Benign', 'Malignant'],
list(breast_cancer_Y.value_counts()), color='blue')
ax.set_xlabel('Labels')
ax.set_ylabel('Count of Tumors')
ax.set_title('Distribution of Labels')
fig.tight_layout()
```

8.4.3 Figure 9 Code

```
breast_cancer_dataset = pd.concat
([breast_cancer_X, breast_cancer_Y], axis=1)

breast_cancer_datasetM = breast_cancer_dataset
[breast_cancer_dataset['Diagnosis']==0]
breast_cancer_datasetB = breast_cancer_dataset
[breast_cancer_dataset['Diagnosis']==1]

fig, ax = plt.subplots( figsize=(10,8))
scatter = ax.scatter(breast_cancer_dataset
[breast_cancer_dataset['Diagnosis']==1]
['mean-compactness'], breast_cancer_dataset[breast_cancer_dataset
['Diagnosis']==1]['mean-radius'],
                     c='b',
                     label="Benign",
                     s=50, )
scatter = ax.scatter(breast_cancer_dataset
[breast_cancer_dataset['Diagnosis']==0]['mean-compactness'],
                     breast_cancer_dataset
                     [breast_cancer_dataset
                     ['Diagnosis']==0]['mean-radius'],
                     c='orange',
                     label="Malignant",
                     s=50,)
```

```
ax.set_xlabel("Mean Compactness")
ax.set_ylabel("Mean Radius")
ax.set_title('Plot of 569 tumor Mean Compactness vs Mean Radius')
ax.legend()
fig.tight_layout()
plt.savefig('breast_cancer_plot.png')
```

8.4.4 Figure 10 Code

```
fig, axs = plt.subplots(nrows=5, ncols=2, figsize = (8,11))
axs = axs.ravel()

for i, ax in enumerate(axs):
    ax.figure
    ax.hist([breast_cancer_datasetM.
    iloc[:,i], breast_cancer_datasetB.iloc[:,i]], label =
    ['Malignant', 'Benign'],
             alpha=0.9, bins =40, stacked=True,
             color= ['orange','b'], density=True)
    ax.legend()
    ax.set_title(breast_cancer_dataset.columns[i])

plt.tight_layout()
plt.savefig('feature_distribution.png')
```

8.4.5 Model Training and Cross Validation Code

```
scaler = StandardScaler()
```

```
breast_cancer_X = scaler.fit_transform
(breast_cancer_X)

cv = KFold(n_splits=5, random_state=1,
shuffle=True)

lda = LinearDiscriminantAnalysis()

#Evaluate model
scores = cross_val_score
(lda, breast_cancer_X,
np.array
(breast_cancer_Y).ravel(),
scoring='accuracy', cv=cv,
n_jobs=-1)

print(f"For LDA the score is {scores.mean()}")

lda.fit(breast_cancer_X,
np.array(breast_cancer_Y).ravel())

# try QDA with different regularization parameter
for reg in [0, 0.1, 1, 2, 10, 100]:
    cv = KFold(n_splits=5, random_state=1, shuffle=True)

    qda = QuadraticDiscriminantAnalysis(reg_param=reg)
```

```
    # evaluate model
    scores = cross_val_score(qda,
    breast_cancer_X, np.array(breast_cancer_Y).ravel(),
    scoring='accuracy', cv=cv, n_jobs=-1)

    print(f"For reg {reg} the score is {scores.mean()}")

#SVC Linear Kernel
for reg in [0, 0.1, 1, 2, 10, 100]:
    cv = KFold(n_splits=5, random_state=1, shuffle=True)

    SVM_linear = SVC(kernel='linear', C=reg)

    # evaluate model
    scores = cross_val_score(SVM_linear,
    breast_cancer_X,
    np.array(breast_cancer_Y).ravel(),
    scoring='accuracy', cv=cv, n_jobs=-1)

    print(f"For C {reg} the score is {scores.mean()}")

#SVC Polynomial Kernel
parameter = {'C':[0.001, 0.01, 0.1, 1, 2, 10, 100],
'degree': [2,3,4] ,
'gamma':[0.01,0.02,0.03,0.04,0.05, 0.06,
0.1, 0.2, 0.5, 0.7, 1, 10, 100]}
```

```
grid_search = GridSearchCV(estimator = SVC(kernel='poly'),
                           param_grid = parameter,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)

grid_search.fit(breast_cancer_X,
np.array(breast_cancer_Y).ravel())


# examine the best model
# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'
.format(grid_search.best_score_))

# print parameters that give the best results
print('Parameters that give the best results :','\n\n',
(grid_search.best_params_))

# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n',
(grid_search.best_estimator_))


#SVC - RBF
parameter = {'C':[0.001, 0.01, 0.1, 1, 2,
10, 100],'gamma':[0.01,0.02,0.03,0.04,0.05,
```

```
0.06,0.1, 0.2, 0.5, 0.7, 1, 10,
100]}

grid_search = GridSearchCV(estimator = SVC(kernel='rbf'),
                           param_grid = parameter,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)

grid_search.fit(breast_cancer_X, np.array(breast_cancer_Y).ravel())
# examine the best model
# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format
(grid_search.best_score_))

# print parameters that give the
# best results
print('Parameters that give the
best results :','\n\n', (grid_search.best_params_))

# print estimator that was chosen by the
GridSearch
print('\n\nEstimator that was chosen
by the search :','\n\n',
(grid_search.best_estimator_))

#Logistic
```

```
cv = KFold(n_splits=5, random_state=1,
shuffle=True)
logit = LogisticRegression
(random_state=0)

#evaluate model
scores = cross_val_score(logit,
breast_cancer_X, np.array(breast_cancer_Y).ravel(),
scoring='accuracy', cv=cv, n_jobs=-1)

print(f"For Logistic the score is {scores.mean()}")

#Naive Bayes
from sklearn.naive_bayes import GaussianNB
cv = KFold(n_splits=5, random_state=1, shuffle=True)
NB = GaussianNB()
# evaluate model
scores = cross_val_score
(NB, breast_cancer_X, np.array
(breast_cancer_Y).ravel(),
scoring='accuracy', cv=cv, n_jobs=-1)

print(f"For Logistic the score is {scores.mean()}")
```

References

[1] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, (Proceedings of the fifth annual workshop on Computational learning theory (pp. 144-152) 1992, July).

[2] V. N. Vapnik, and A. Y. Lerner, *Recognition of patterns with help of generalized portraits*, (Avtomat. i Telemekh, 24(6), 774-780, 1963)

[3] C. Cortes and V. N. Vapnik, *Support vector networks. Machine Learning*, (Machine learning 20: 273-297, 1995).

[4] N. Aronszajn, *Theory of reproducing kernels* , (Transactions of the American mathematical society, 68(3), 337-404, 1950)

[5] A. Hertzmann, D. J. Fleet and M. Brubaker, *CSC C11/D11 Machine Learning and Data Mining Lecture Notes* (University of Toronto, 2015).

[6] T. Hastie, R. Tibshirani, J. Friedman *The Elements Of Statistical Learning* (Springer, 2017)

[7] J. C. Platt, *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines* (Microsoft Research, 1998).

[8] D. Knowles, *Lagrangian Duality for Dummies* (Stanford University, 2010)

[9] Alpaydin, E., 2014, *Introduction to Machine Learning.* , The MIT Press.

[10] Jiawei Y., Zeping W., Ke P, Patrick N. O., Weihua Z., Hailong Z., and Jingbo S., *Parameter selection of Gaussian kernel SVM based on local density of training set*, *Inverse Problems in Science and Engineering*, **29**(4), 536-548, 2021, Taylor & Francis

[11] Allerbo, O., Jörnsten, R., 2023, *Bandwidth Selection for Gaussian Kernel Ridge Regression via Jacobian Control*, arXiv: 2205.11956.

[12] Xavier Bourret Sicotte, *Kernel Feature Map*, 2018, `https://xavierbourretsicotte.github.io/Kernel_feature_map.html`.

[13] W. Wolberg, O. Mangasarian, N. Street, and W. Street, *Breast Cancer Wisconsin (Diagnostic)* , (UCI Machine Learning Repository, 1995).

[14] M. Hossin, and M. N. Sulaiman, *A review on evaluation metrics for data classification evaluations.*, (International journal of data mining & knowledge management process, 5(2), 1. 2015).

[15] M. Grandini, E. Bagli, and G. Visani, *Metrics for multi-class classification: an overview.*, (arXiv preprint arXiv:2008.05756., 2020).

[16] S. M. McElwee, *Probabilistic clustering ensemble evaluation for intrusion detection*, (Figure 3, p29, Doctoral dissertation, Nova Southeastern University, 2018).

[17] S. S. Keerthi, O. Chapelle,D. DeCoste,K. P. Bennett, and E. Parrado-Hernández, 2006. *Building support vector machines with reduced classifier complexity.*, Journal of Machine Learning Research, 7(7).

[18] R. Akbani, S. Kwek and N. Japkowicz, 2004. *Applying support vector machines to imbalanced datasets*, In Machine Learning: ECML 2004: 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004. Proceedings 15 (pp. 39–50). Springer Berlin Heidelberg.

[19] S. Huang, N. Cai, P. P. Pacheco, S. Narrandes, Y. Wang, and W. Xu, 2018, *Applications of support vector machine (SVM) learning in cancer genomics*, Cancer genomics & proteomics, 15(1), pp. 41–51.

[20] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, and C. D. Bloomfield, 1999.

Molecular classification of cancer: class discovery and class prediction by gene expression monitoring , Science, 286(5439), pp.531-537.

[21] V. Vapnik, and S. Mukherjee, 1999, *Support vector method for multivariate density estimation* , Advances in neural information processing systems.

[22] E. J. Moler, M. L. Chow, and I. S. Mian, 2000, *Analysis of molecular profile data using generative and discriminative methods* , Physiological genomics, 4(2), pp.109-126.

[23] T. B. Trafalis, and H. Ince, 2000, July, *Support vector machine for regression and applications to financial forecasting* , In Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium (Vol. 6, pp. 348-353). IEEE.

[24] W. Huang, Y. Nakamori, and S.Y. Wang, 2005, *Forecasting stock market movement direction with support vector machine.* ,Computers & Operations Research, 32(10), pp. 2513–2522.

[25] J.H. Trustorff, P. M. Konrad and J. Leker, 2011, *Credit risk prediction using support vector machines* , Review of Quantitative Finance and Accounting, 36, pp.565-581.

[26] H. S. Kim, and S.Y. Sohn, 2010, *Support vector machines for default prediction of SMEs based on technology credit.* , European Journal of Operational Research, 201(3), pp. 838–846.

[27] A. Pradhan, 2012, *Support vector machine-a survey*, International Journal of Emerging Technology and Advanced Engineering, 2(8), pp.82-85.

[28] Y. Ma, and G. Guo eds., 2014, *Support vector machines applications (Vol. 649)* , Springer- Verlag, New York.

[29] I. Scholl, t. Aach, T. M. Deserno, and T. Kuhlen, 2011, *Challenges of medical image processing* , Computer science-Research and development, 26, pp.5-13.

[30] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, 1998., *Support vector machines* , IEEE Intelligent Systems and Their Applications, 13(4), pp. 18–28.

[31] Q. Zhao, and L. Zhang, 2005, October, *ECG feature extraction and classification using wavelet transform and support vector machines* , 2005 IEEE International Conference on Neural Networks and Brain (Vol. 2, pp. 1089–1092).

[32] X. Y. Wang, T. Wang, and J. Bu, 2011. *Color image segmentation using pixel-wise support vector machine classification* , Pattern Recognition, 44(4), pp. 777–787.