University of Bergen
Department of Informatics

Topological Regularization of Support Vector Machines

Author: Willem Schooltink
Supervisor: Nello Blaser

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

November, 2023

Abstract

The unique functioning of Support Vector Machines (SVMs) can create undesirable behaviour, however this unique functioning in turn also allow for exploration of novel topological regularization methods. SVMs are a common machine learning model used for classification tasks. Like most machine learning models, SVMs can be prone to overfitting to training data, which hurts its generalization. When optimizing the generalization of classifiers we can consider topological properties of the decision boundaries. Intuitively, a decision boundary slicing the input space into numerous disjoint components is less likely to generalize effectively. Building on this assumption, I propose two novel topological generalization techniques using properties specific to SVMs.

SVMs work by mapping data into a higher-dimensional feature space, with the goal of making data linear separable among classes. Analysing this feature space with methods from topology and Morse theory leads to new methods to change an SVM's decision boundary in a manner that reduces its topological complexity. In particular, I define a measure of topological complexity for SVMs and propose 2 methods intended to decrease this complexity. I show practical utility of these methods through illustrative examples, which indicate effectiveness in real-world scenarios. Moreover, the results of experiments challenge the idea that a maximal margin results in an optimal decision boundary.

This thesis's goal is to serve as a basic exploration into topological regularization for SVMs, providing ideas for further research. In the discussion I identify weaknesses and challenges such as reliance on approximations and computationally costly algorithms, which can be tackled through further future research.

Acknowledgements

First off, I want to extend my thanks to Nello Blaser, my supervisor for this thesis. Thank you helping me throughout this thesis, first by helping shape the topic of this thesis into an interesting and challenging problem I have gladly spent a good part of a year on working on. During this work you helped me by discussing possible approaches and providing invaluable feedback on my work and writing. Thank you for your help as supervisor.

Furthermore, I want to thank my fellow students who provided feedback on the thesis, while discussing the general challenges of working on a thesis. At the time of writing this some of you have already graduated, congratulations to you. To the others who are continuing to work on their own thesis, I want to wish you the best of luck.

`Willem Schooltink`
Friday 17$^{\text{th}}$ November, 2023

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Support Vector Machines (SVMs) are a specific kind of *machine learning classifiers* which through their unique workings can create undesirable behaviours, however these unique workings in turn also allow for exploration of novel *regularization* methods. SVMs, like other machine learning *models*, are designed such that they learn to solve pre-defined problems by finding patterns in data. The way an SVM classifies datapoints is by mapping them into a high dimensional *feature space*, after which the SVM finds an optimal *linear separation* of the different classes. The exact workings will be discussed in section 2.3, but in short SVMs transform data to a higher dimensional space with the goal of making it linearly separable. This process has the potential of generating undesirable behaviour for the classifier, but in turn also allows for unique topological analysis of the classifier. Figures 1.1 and 1.2 show 2 examples of Support Vector Machines classifiers with undesirable behaviour.

One common task tackled with machine learning is data *classification*, where a machine learning model is tasked to predict which class some *datapoint* belongs to. An example of a classification task is spam detection, where a model predicts whether some message is either in the class 'spam' or 'not spam'. Classifiers are trained by providing them a *dataset* of pre-classified data. Using this dataset the model attempts to optimize its prediction such that it most accurately can predict the class of any datapoint. The challenge in training classifiers is that the goal of such models is not necessarily to be very accurate at predicting the pre-classified data it uses to learn, but instead it should

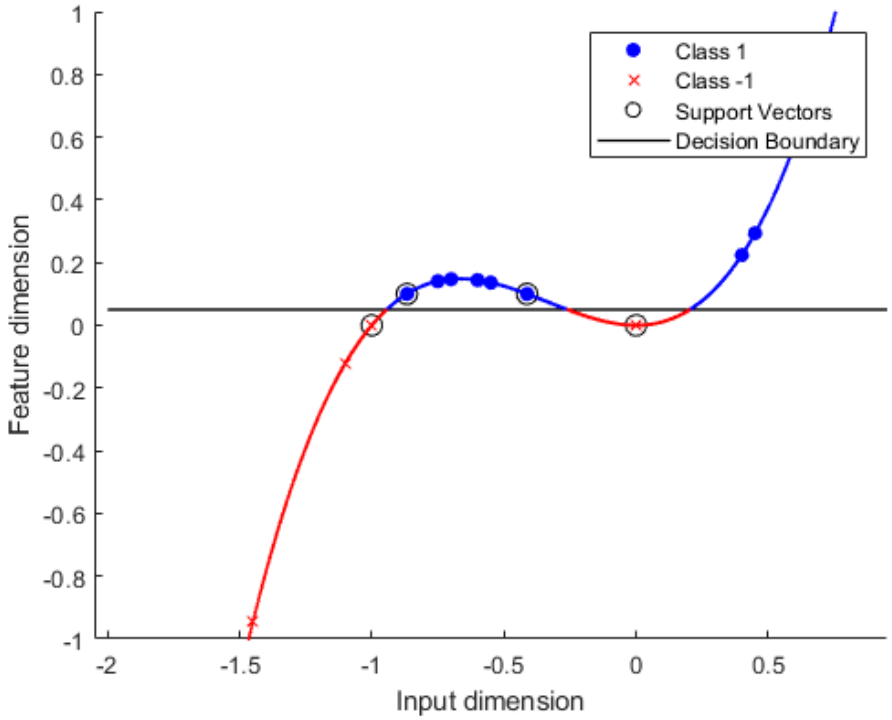

Figure 1.1: Let any datapoint $x \in \mathbb{R}$, the data is transformed into 2-dimensional space by mapping it to the function in red and blue. The function used here is $f(x) = x^3 + x^2$. The black line shows a learned linear separator. The colour of the function shows the prediction of the model for each point, as it is transformed to the high dimensional space. Assume the datapoint $x = 0$ is noise, then the model has an undesirable red section slicing the blue section.

be accurate in predicting unseen data. In other words, we are interested in how well a model's prediction *generalizes*.

Figure 1.1 can be interpreted as an example of *overfitting* (later explained in section 2.2): the classifier learns to predict the data it is trained with too specifically such that it degrades the classifiers ability to predict new data. Here the classifier learns an incorrect pattern from the noise in the training data. Furthermore, Figure 1.2 shows how undesirable classification for SVMs can happen without it being a result of overfitting. Here a mapping function that fits poorly to the data creates an undesirable classifier. In both examples we see that the undesirable behaviour of the classifiers is directly related to the number of components of the mapping function as sliced by the linear separator. In this thesis I will explore methods that attempt to mitigate such undesirable behaviour by reducing this number of components.

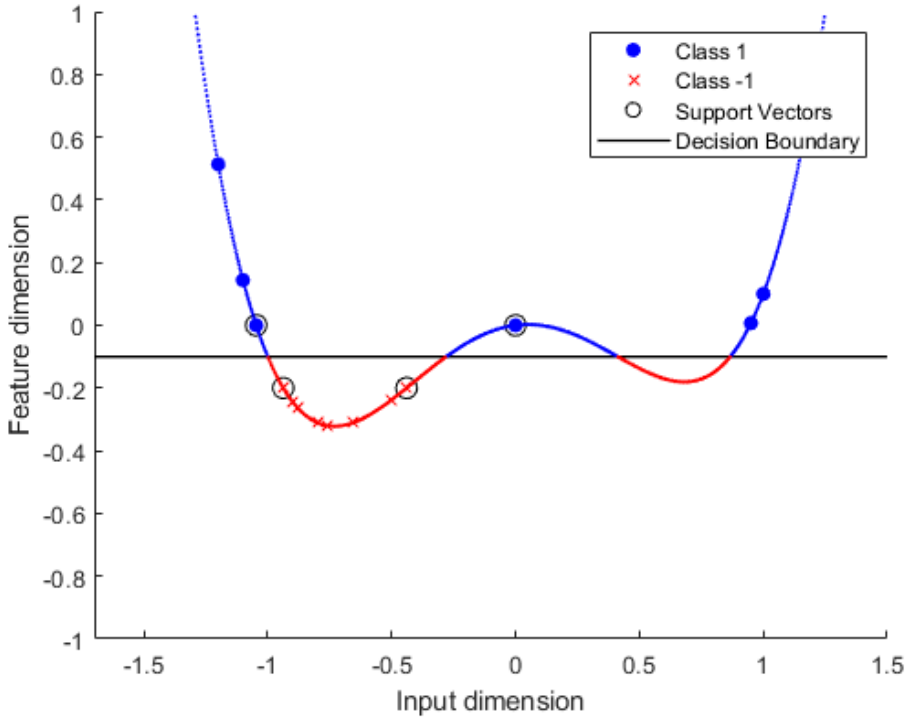

Figure 1.2: Let any datapoint $x \in \mathbb{R}$, the data is transformed into 2-dimensional space by mapping it to the function in red and blue. The function used here is $f(x) = x^4 - x^3 + 0.1x$. The black line shows a learned linear separator. The colour of the function shows the prediction of the model for each point, as it is transformed to the high dimensional space. We see that the function has a red section which has no red datapoints in it, and based on the datapoints next to it, should probably be blue.

1.2 Objectives

The ultimate objective of this thesis is to develop a method that reduces complexity using topological analysis of the high dimensional feature space, specifically addressing the undesirable complexity shown in Figure 1.1 and Figure 1.2. This objective can be split into smaller sub-objectives. The objectives of this thesis are:

- Measure the complexity of SVMs.
- Regularize SVMs through a novel method reducing the complexity.
- Show that the method has a positive effect in practice.

1.3 Thesis Outline

Following this introduction and objective statement I will introduce the requisite background knowledge in Section 2. The background can roughly be split into two parts: machine learning background, and mathematical background. The background introduces all concepts required to understand the working of SVMs, and mathematical notions used later for the novel regularization methods. In Section 3 I will describe two novel regularization methods for SVMs, and define some measures that give some insight into the performance of the methods. For both regularization methods I will first discuss the intuition behind them, before describing the algorithms. In Section 4 I will outline the setup of three experiments and show their results. Each of the exeriments are chosen to show a specific use case for the methods. The first 2 experiments show a situation designed to show the functioning of the methods in a simple environment. The third experiment shows the methods working in a more realistic scenario. I interpret and discuss the results of these experiments in Section 5. Here I will also identify the strengths and weaknesses of my proposed methods and provide a conclusion that relates back to the objectives of this thesis.

Chapter 2

Background

2.1 Classification Models

In this thesis I will be working with SVMs, which are a specific kind of *classification* model. The goal of classification models, also known as *classifiers*, is to predict the *class* a given input belongs to. An example of a classification task is determining whether a given message is spam or not. In the case of a spam detector a message is a *datapoint* which belongs to either the class 'spam' or the class 'not spam'. The example shows a *binary classifier*, that is: a classifier that predicts whether an input is either in class A or class B. Classification tasks can be extended to have any number of possible classes.

Often a classifier approximates the probability that a datapoint $\boldsymbol{x}$ belongs to any of the possible classes $c_i \in C$ of which the class with the highest probability is chosen:

$$\text{predict}(\boldsymbol{x}) = \underset{c \in C}{\arg\max}\ P(c \mid \boldsymbol{x}),$$

where $P(c \mid \boldsymbol{x})$ is the estimated probability of $\boldsymbol{x}$ belonging to class c. Relating back to the example of the spam filter $\boldsymbol{x}$ is a vector of numerical values representing an message, and $C = \{\text{'Spam'}, \text{'Not Spam'}\}$.

2.1.1 Performance Measures

In order to determine the 'goodness' of a classifier we define *performance measures*. Consider a binary classifier that predicts a given datapoint to be in either one of 2 classes,

lets call these classes the positive and negative class. Besides the prediction of the model, each point also has a *ground truth*: the actual correct class. If a model correctly predicts a datapoint to belong in the positive class we call that a *True Positive* (TP), conversely a correct negative prediction is called a *True Negative* (TN). When a model incorrectly predicts a class to be positive while its true class is negative it is called a *False Positive* (FP), and conversely an incorrect negative prediction is called a *False Negative* (FN), see table 2.1.

		Prediction	
		Positive	**Negative**
Truth	**Positive**	True Positive	False Negative
	Negative	False Positive	True Negative

Table 2.1: Definitions of True–, and False Positives and Negatives

Let TP, TN, FP, and FN be the number of predictions belonging to each respective case, such that TP equals the number of true positive predictions, and so forth. Using these definitions we can define performance measures. A common, and intuitive, performance measure is the *accuracy* score. Accuracy measures the percentage of correct predictions of the model, thus:

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

Since the accuracy score looks at the number correct predictions versus the number of incorrect predictions, for any imbalanced datasets other performance measures might be more fitting. For this thesis the datasets used will be balanced among classes, so I will use accuracy as the performance measure.

2.1.2 Decision Boundaries

Let's consider a dataset X which contains a set of datapoints $\boldsymbol{x}$, each of which is a vector of values, such that $\boldsymbol{x} = \langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \rangle, \forall \boldsymbol{x} \in X$. We call $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ the *features* of $\boldsymbol{x}$.

The task of a classification model is to take in a *feature vector* $\boldsymbol{x}$ and output a prediction as to what class it belongs to. Since $\boldsymbol{x}$ is a vector, we can view $\boldsymbol{x}$ as a point in an n-dimensional space, where n is the cardinality of $\boldsymbol{x}$. Therefore, the class predicted by a classification model depends on the location of a datapoint in this *feature space*.

Assuming a classifier predicts more than 1 class, then intuitively there must be a subset S of the feature space where all datapoints $\boldsymbol{x} \in S$ are classified as class 1, and another subset S' of the input space where all datapoints $\boldsymbol{x} \in S'$ are classified as class -1. When S and S' are adjacent in the input space the boundary between them is what we call the decision boundary. That is; the space where an infinitesimal change in the feature vector $\boldsymbol{x}$ changes the predicted class of $\boldsymbol{x}$.

2.2 Training and Generalization

In order to use machine learning classifiers, they have to be trained on a dataset first. Exactly how a classifier learns differs per choice of model and implementation, but there are practices that apply to all cases. First we must define what makes a machine learning model good. Since the goal of a model is to provide useful predictions on data we have not yet seen, we cannot rely on a models ability to accurately predict on training data. Instead we need a measure of how well a model predicts unseen data. For this purpose we use 3 datasets during the training and testing phase: a training set X_{train}, validation set X_{val}, and test set X_{test} such that

$$X_{train} \cap X_{val} = X_{train} \cap X_{test} = X_{val} \cap X_{test} = \emptyset.$$

X_{train} is intuitively used to train the model. During training we can check how well a model performs on other data by testing them using X_{val}. Based on the performance we can chose to continue training, tune hyper-parameters, or decide the model is finished. This means that we are optimizing a model to perform well on X_{val}, so to truly measure its performance on unseen data the final model can be tested using X_{test}.

A model is said to *generalize* well when it can predict unseen data well. In order to get a model that does generalize well it has to learn useful patterns from the X_{train}, without overly focusing on noise patterns that may exist in X_{train}. Models that learn the specifics of X_{train} too much, such that its ability to generalize is poor is said to *overfit*. Conversely, a model that does not learn enough patterns from the training dataset is said to be *underfitting*. Figure 2.1 gives a visual intuition on over– and underfitting. This shows that there is an important balance to be found when training a model, such that it finds the overarching patterns without learning patterns specific to X_{train} only.

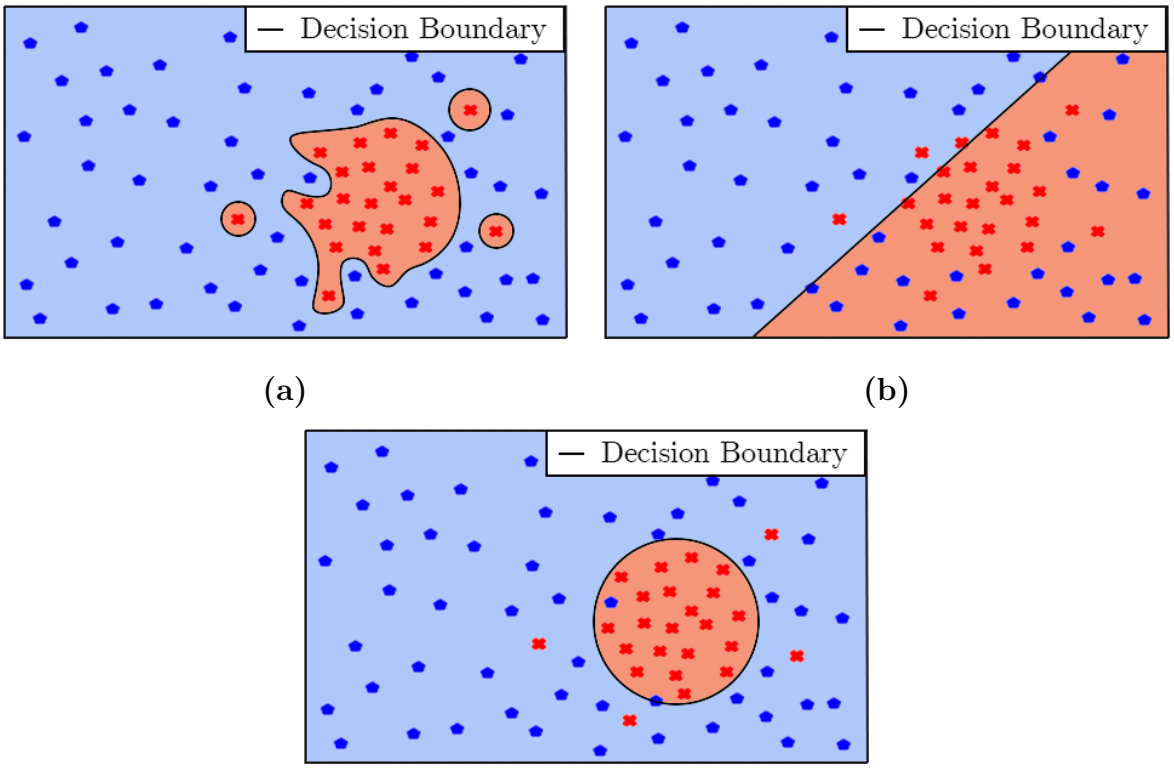

Figure 2.1: As data can be noisy machine learning models have to balance how precise they fit to patterns in training data. The colour and shape of the datapoints indicate what class they belong to according, and the red and blue areas of the space show the prediction of the model for data in those areas. Assume Figure **(c)** shows the true underlying pattern, then model **(a)** overfits to the training data by learning to include noise, and model **(b)** underfits by ignoring too much data such that it doesn't find the underlying pattern.

2.3 Support Vector Machines

Support Vector Machines (SVMs) are supervised learning models that can be used for binary or multi-class classification problems [3, 7]. From here on we will consider binary classification SVMs, without explicitly specifying further on.

SVMs work by finding a way to separate data into classes as best as possible using a linear decision boundary (a line, plane, or other n-dimensional hyperplane). Let $\boldsymbol{w}^T X + b = 0$ describe the hyperplane defining the decision boundary, where $\boldsymbol{w}$ is the normal vector of the hyperplane and b its bias. The classification of any datapoint $\boldsymbol{x}$ is then determined by the sign of $\boldsymbol{w}^T \boldsymbol{x} + b$. We can see that this is similar to calculating the

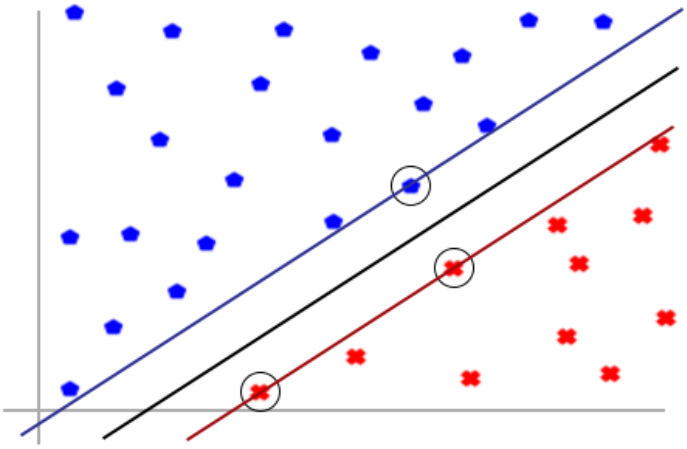

Figure 2.2: Consider the following example of a SVM. The black line indicates the optimal linear separator that maximizes the margin, as visualized by the blue and red lines. The points encircled in black are the support vectors. Note how these support vectors define the linear separator.

distance of a point $\boldsymbol{x}$ to a hyperplane, which is given by

$$\frac{\boldsymbol{w}^T\boldsymbol{x} + b}{||\boldsymbol{w}||}.$$

The only difference is the scaling factor $\frac{1}{||\boldsymbol{w}||}$ which normalizes the length of $\boldsymbol{w}$. We can see that this scaling is irrelevant to the problem, as we only care about which side of the hyperplane the datapoint $\boldsymbol{x}$ is, not the exact distance.

Generally we consider a decision boundary of SVMs optimal if the margin between the closest datapoints and the linear decision boundary is the largest [3]. This margin is the minimum distance between any of the datapoints and the hyperplane. These closest points that determine the margin are known as support vectors, as visualized in figure 2.2.

In the case of *linear SVM*s we consider the input space to be the feature space, and therefore find the hyperplane optimally separating the data in this input space.

2.3.1 SVM Kernels

Often data is not linearly separable. In these cases SVMs make use of *mapping functions* ϕ. A mapping function ϕ maps a datapoint $\boldsymbol{x}$ from the original space $\mathbb{R}^n$ to a higher dimensional space $\mathbb{R}^m$ such that $\phi(\boldsymbol{x}) = \mathbb{R}^n \rightarrow \mathbb{R}^m$, where n is the amount of input

space dimensions and m the amount of dimensions of the feature space. Let $\phi(\boldsymbol{x}) = \langle f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}) \rangle$ then the functions $f_1, f_2, \ldots, f_n$ are called basis functions of ϕ. A common type of basis function are polynomial functions. Using polynomial basis functions we map data to a new space generating some polynomial hypersurface. The goal of this mapping is to transform data in such a way that it can be linearly separated in this new higher dimensional feature space. Figure 2.3 shows an example of an SVM classifier mapping 1-dimensional data to a 2-dimensional space.

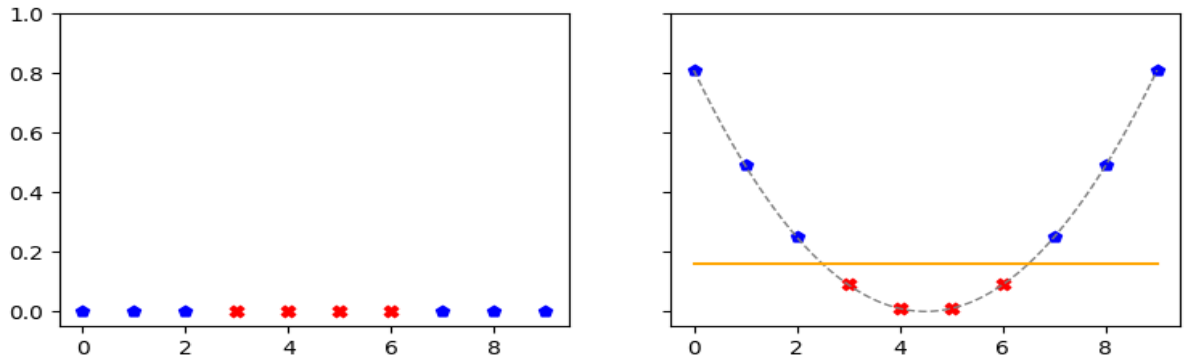

Figure 2.3: Consider the 1-dimensional dataset as shown in the first plot, where all points either belong to the red or blue class as indicated by their colour in the plot. Let $\phi(x) = \langle x, 0.05(x - 4.5)^2 \rangle$. In the right plot we see the data mapped into this feature space, with the grey dashed line visualizing the curve created by $\phi(x) \mid x \in \mathbb{R}$. Now the data is linearly separable in the feature space, for example by the yellow line which is the separator which maximizes the margin.

We can compute the high dimensional relations of transformed feature vectors with *kernel functions* without having to explicitly calculate the transformation $\phi(\boldsymbol{x})$ by using the *Kernel Trick*, as first shown by M.A. Aizerman, E.A. Braveman, and L. Rozonoer [1]. A kernel function $K(\boldsymbol{x}, \boldsymbol{y})$ calculates the dot product of $\boldsymbol{x}$ and $\boldsymbol{y}$ in the high dimensional feature space such that $K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$.

There are several types of kernel functions that are often used with SVMs. Some of the commonly used ones are:

- *Linear Kernel* ($K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \cdot \boldsymbol{y}$):
  The linear kernel is used in linear SVMs. It represents the standard dot product of $\boldsymbol{x}$ and $\boldsymbol{y}$ in the input space. So when using the linear kernel the input space is the feature space.
- *Polynomial Kernel* ($K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} \cdot \boldsymbol{y} + c)^d$):
  The polynomial kernel calculates a relation between $\boldsymbol{x}$ and $\boldsymbol{y}$ in a higher dimensional space. The amount of dimensions is determined by the amount of dimensions of

the input space and the degree set by the parameter d. The parameter c is tunable setting the relative influence of higher-order terms versus lower-order terms of the polynomial.

We can find the ϕ that is generated by a polynomial kernel by simply expanding the kernel function. Consider the following example: let $K(\boldsymbol{x}, \boldsymbol{y})$ be a polynomial kernel of degree $d = 3$ with $c = 1$, and $\boldsymbol{x} = \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle$, $\boldsymbol{y} = \langle \boldsymbol{y}_1, \boldsymbol{y}_2 \rangle$ such that $\boldsymbol{x} \in \mathbb{R}^2$ and $\boldsymbol{y} \in \mathbb{R}^2$. In order to find the mapping generated by K we can work out $K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} \cdot \boldsymbol{y} + c)^d$ for $\boldsymbol{x}$ and $\boldsymbol{y}$.

$$
\begin{aligned}
K(\boldsymbol{x}, \boldsymbol{y}) =& (\boldsymbol{x} \cdot \boldsymbol{y} + 1)^3 \\
=& \left(\sum_{i=1}^{2} \boldsymbol{x}_i \boldsymbol{y}_i + 1 \right)^3 \\
=& (\boldsymbol{x}_1 \boldsymbol{y}_1 + \boldsymbol{x}_2 \boldsymbol{y}_2 + 1)^3 \\
& \Downarrow \text{Apply multinomial theorem} \\
=& (\boldsymbol{x}_1 \boldsymbol{y}_1)^3 + (\boldsymbol{x}_2 \boldsymbol{y}_2)^3 + (1)^3 + 3 (\boldsymbol{x}_1 \boldsymbol{y}_1)^2 (\boldsymbol{x}_2 \boldsymbol{y}_2) \\
& + 3 (\boldsymbol{x}_1 \boldsymbol{y}_1)^2 (1) + 3 (\boldsymbol{x}_2 \boldsymbol{y}_2)^2 (\boldsymbol{x}_1 \boldsymbol{y}_1) + 3 (\boldsymbol{x}_2 \boldsymbol{y}_2)^2 (1) \\
& + 3(1)^2 (\boldsymbol{x}_1 \boldsymbol{y}_1) + 3(1)^2 (\boldsymbol{x}_2 \boldsymbol{y}_2) + 6 (\boldsymbol{x}_1 \boldsymbol{y}_2) (\boldsymbol{x}_2 \boldsymbol{y}_2) (1) \\
=& 1 + (\boldsymbol{x}_1^3)(\boldsymbol{y}_1^3) + (\boldsymbol{x}_2^3)(\boldsymbol{y}_2^3) + (\sqrt{3}\boldsymbol{x}_1^2 \boldsymbol{x}_2)(\sqrt{3}\boldsymbol{y}_1^2 \boldsymbol{y}_2) \\
& + (\sqrt{3}\boldsymbol{x}_1^2)(\sqrt{3}\boldsymbol{y}_1^2) + (\sqrt{3}\boldsymbol{x}_1 \boldsymbol{x}_2^2)(\sqrt{3}\boldsymbol{y}_1 \boldsymbol{y}_2^2) + (\sqrt{3}\boldsymbol{x}_2^2)(\sqrt{3}\boldsymbol{y}_2^2) \\
& + (\sqrt{3}\boldsymbol{x}_1)(\sqrt{3}\boldsymbol{y}_1) + (\sqrt{3}\boldsymbol{x}_2)(\sqrt{3}\boldsymbol{y}_2) + (\sqrt{6}\boldsymbol{x}_1 \boldsymbol{x}_2)(\sqrt{6}\boldsymbol{y}_1 \boldsymbol{y}_2).
\end{aligned}
$$

We know that $K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$, and note how the expansion of $K(\boldsymbol{x}, \boldsymbol{y})$ looks like the result of a dot product of 2 vectors. So it follows that

$$
\phi(\boldsymbol{x}) = \langle 1, \boldsymbol{x}_1^3, \boldsymbol{x}_2^3, \sqrt{3}\boldsymbol{x}_1^2 \boldsymbol{x}_2, \sqrt{3}\boldsymbol{x}_1 \boldsymbol{x}_2^2, \sqrt{3}\boldsymbol{x}_1^2, \sqrt{3}\boldsymbol{x}_2^2, \sqrt{3}\boldsymbol{x}_1, \sqrt{3}\boldsymbol{x}_2, \sqrt{6}\boldsymbol{x}_1 \boldsymbol{x}_2 \rangle,
$$

for the polynomial kernel with $c = 0$, $d = 3$, and $n = 2$ such that $\boldsymbol{x} \in \mathbb{R}^n$. While it is possible to define kernels that only add a single dimension to the input data, often many new dimensions are created like in this example. The ϕ we just found maps data from $\mathbb{R}^2$ to $\mathbb{R}^{10}$. Note however how the polynomial kernel creates one dimension in which all points are mapped to some constant value. Consequently this dimension does not help in creating linear separability and can be ignored.

- *The Radial Basis Function (RBF) Kernel* ($K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma ||\boldsymbol{x} - \boldsymbol{y}||^2}$):
  The RBF kernel takes the idea of mapping input data to higher dimensions to its logical extreme: by definition of e the RBF kernel calculates the relation between $\boldsymbol{x}$ and $\boldsymbol{y}$ in infinite dimensions. γ is a tunable parameter that controls the spread

of the kernel. A kernel with a high γ value is more likely to overfit and vice versa. To see how the RBF kernel generates a ϕ of infinite dimensions consider the following example: let $K(\boldsymbol{x}, \boldsymbol{y})$ be an RBF kernel, with $\gamma = 1$, $\boldsymbol{x} \in \mathbb{R}$, and $\boldsymbol{y} \in \mathbb{R}$ for simplicity. Expanding this kernel gives us:

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{y}) &= e^{-||\boldsymbol{x}-\boldsymbol{y}||^2} \\ &= e^{-||\boldsymbol{x}||^2} \cdot e^{-||\boldsymbol{y}||^2} \cdot e^{2\boldsymbol{x}\cdot\boldsymbol{y}} \\ &\qquad \Downarrow \text{ Taylor expansion of the last } e \\ &= e^{-||\boldsymbol{x}||^2} \cdot e^{-||\boldsymbol{y}||^2} \sum_{n=0}^{\infty} \frac{(2\boldsymbol{x}\cdot\boldsymbol{y})^n}{n!}. \end{aligned}$$

Note how $e^{-||\boldsymbol{x}||^2}$ and $e^{-||\boldsymbol{y}||^2}$ simply act as some scalar value. This creates a $\phi = e^{-||\boldsymbol{x}||^2}\langle e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \ldots, e_\infty(\boldsymbol{x})\rangle$, with $e_n(x) = \sqrt{\frac{2^n}{n!}}x^n$ [18]. Following this definition we can see that this ϕ maps data from $\mathbb{R}$ to $\mathbb{R}^\infty$. I will show a more general form of this feature mapping in section 2.4.1

- *The Sigmoid Kernel* ($K(\boldsymbol{x}, \boldsymbol{y}) = tanh(\boldsymbol{x}\cdot\boldsymbol{y} + c)$):
  The sigmoid kernel, using a trick similar to the RBF kernel, also computes a relation between $\boldsymbol{x}$ and $\boldsymbol{y}$ in an infinite dimensional feature space.

2.3.2 Optimization

If the data in the feature space is linearly separable we can optimize an SVM with a *hard margin*, finding a linear separator such that no datapoint is misclassified. Let $\boldsymbol{w}$ be the normal vector of the decision boundary hyperplane and b its bias, such that $\boldsymbol{w}^T X + b = 0$ describes the hyperplane. Let the possible classes $C = \{-1, 1\}$. Let $y_i \in C$ be the class label of a datapoint $\boldsymbol{x}_i$, then the hard margin constraint can be defined as $\boldsymbol{w}^T\boldsymbol{x}_i + b \geq 1$ for any datapoint $\boldsymbol{x}_i \in \{\boldsymbol{x}_i |\ \forall\,(\boldsymbol{x}_i, y_i) \in X, y_i = 1\}$, and $\boldsymbol{w}^T\boldsymbol{x}_i + b \leq -1$ for any datapoint $\boldsymbol{x}_i \in \{\boldsymbol{x}_i |\ \forall\,(\boldsymbol{x}_i, y_i) \in X, y_i = -1\}$. We can further generalize this to $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1$, where $y_i \in \{-1, 1\}$ is the class label for $\boldsymbol{x}_i$.

The goal of the optimization is to find a hyperplane such that the margin is maximized. We can define the margin with 2 hyperplanes parallel to the separator hyperplane as shown in Figure 2.4.

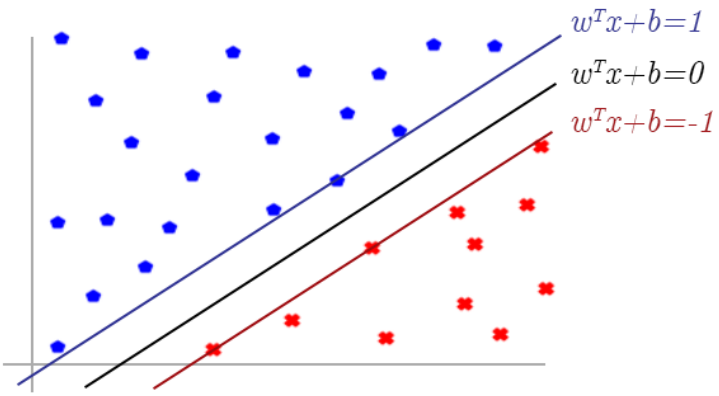

Figure 2.4: The margin of a linear separator is defined by the closest datapoints. Here the black line shows the linear separator, and the blue – and red line visualize the margin of the separator.

Using the formula for calculating the distance between a point and a line we see that the distance between any point on $\boldsymbol{w}^T\boldsymbol{x} + b = 1$ to the line $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ is $\frac{1}{||\boldsymbol{w}||}$. This means the total distance between the 2 lines of the margin is $\frac{2}{||\boldsymbol{w}||}$. With this we can define the optimization goal as $\max_{\boldsymbol{w},b} \frac{2}{||\boldsymbol{w}||}$ or alternatively $\min_{\boldsymbol{w},b} \frac{||\boldsymbol{w}||}{2}$. In order to simplify for derivation later we square the $||\boldsymbol{w}||$ term. Which results in the final optimization goal being:

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2,$$
$$\text{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 \text{ for } i \in \{1, 2, \ldots, n\}.$$

We can solve this optimization problem using Lagrange multipliers α as follows:

$$\mathcal{L}(\boldsymbol{w}, b, \alpha) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\boldsymbol{w}^T\boldsymbol{x}_i + b \right) - 1 \right].$$

Using the derivatives of the Lagrange function w.r.t. $\boldsymbol{w}$ and b we can find conditions for the global minimum.

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \alpha) = 0$$
$$\nabla_b \mathcal{L}(\boldsymbol{w}, b, \alpha) = 0$$

Solving these equations gives us the following conditions:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i,$$
$$\sum_{i=1}^{n} \alpha_i y_i = 0.$$

Substituting in these conditions we can then finally define the dual of the SVM optimization problem as

$$\max_{\alpha} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i=1}^{n} \alpha_i,$$
$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

This dual is relatively easy to solve, as it only requires optimization of the Lagrange multipliers α instead of both $\boldsymbol{w}$ and b.

In cases where the data can not be linearly separated in the feature space, or if we want to allow for some misclassification with the goal of improving generalization of the model, we use the *soft margin* optimization goal. The idea is that the soft margin allows for some misclassification when optimizing its linear separator and margin. To achieve this the hard margin goal is extended as:

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}||\boldsymbol{w}||^2 + \mathrm{C} \sum_{i=1}^{n} \zeta_i,$$
$$\text{s.t.} \ \ y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \zeta_i \text{ for } i \in \{1, 2, \ldots, n\}.$$

Here ζ_i is a measure of how far the i-th datapoint is on the wrong side of its corresponding margin and C is a parameter that determines the importance of ζ_i. A C with a high value puts more importance on correct classification, and might push a model towards overfitting, and vice versa. Figure 2.5 shows a visual intuition for the ζ_i term.

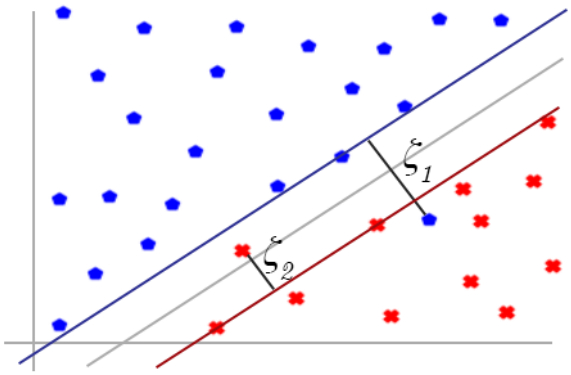

Figure 2.5: ζ_i describes the distance from a datapoint on the wrong side of its margin to the margin. Here the grey line indicates the linear separator and the blue – and red line visualize the margin of the separator.

Tackling this optimization problem as shown with the hard margin constraint before gives us a slightly different dual problem, adding an upper bound to the Lagrange multipliers.

$$\max_{\alpha} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i,$$
$$\text{s.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0,\ 0 \leq \alpha_i \leq \mathrm{C}.$$

The Lagrange multipliers α_i define how much importance a datapoint $\boldsymbol{x}_i$ has on the definition of the linear separator.

2.4 RBF kernel approximation

In section 2.3.1 I show how the RBF kernel function generates a ϕ that maps $\mathbb{R} \to \mathbb{R}^\infty$ for a 1-dimensional input space. I will show an expansion of the definition of ϕ such that it works for any n-dimensional input space, and a method to approximate such mappings with a finite dimensional feature space. As this infinite dimensional space can be difficult or expensive to work with, some might want to approximate the feature space of the RBF kernel with some finite dimensional space. While these approximate feature spaces are mostly used for efficient predictions, they are necessary later in this thesis as some methods rely on finite dimensional feature spaces.

In literature there are several approaches to tackle this problem. In their paper Hui Cao, Takashi Naito, and Yoshiki Ninomiya [4] show and use a polynomial approximation

of the RBF kernel function, and show similar performance of these approximate models as compared to exact RBF models for their specifc usecase.

Alternatively, M. Ring and B.M. Eskofier [17] propose an approximation of the RBF kernel which truncates the number of feature space dimensions. The paper details the relative error of the truncated ϕ_Q against the original mapping ϕ, and show that decision boundaries can be very accurately estimated following this approach. In this section I will outline this method, as this is the approximation method I will be using later.

2.4.1 RBF feature space

We know that the ϕ generated by the RBF kernel function can be written as $\phi = e^{-||\boldsymbol{x}||^2}\langle e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \dots, e_\infty(\boldsymbol{x})\rangle$ for 1-dimensional data and $\gamma = 1$. In the case of higher dimensional input spaces we first need to introduce the multi-index $\mathbf{n}_i = (n_1, n_2, \dots, n_D)$ with $D = |\boldsymbol{x}|$ such that $\mathbf{n}_i \in \mathbb{N}_0^D$. Note that a bold $\mathbf{n}_i$ refers to a multi-index, and a regular n_i refers to an element of a multi-index. We then define $e_{\mathbf{n}_i}(\boldsymbol{x})$ as the tensor product $e_{\mathbf{n}_i}(\boldsymbol{x}) = e_{n_1}(\boldsymbol{x}_1) \otimes e_{n_2}(\boldsymbol{x}_2) \otimes \cdots \otimes e_{n_D}(\boldsymbol{x}_D)$ with $D = |\boldsymbol{x}|$. As all $e_{n_i}(\boldsymbol{x}) \in \mathbb{R}$ the tensor product is equivalent to a simple scalar product of the functions.

We can rewrite the RBF kernel as follows:

$$K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma||\boldsymbol{x}-\boldsymbol{y}||^2} = e^{-||\boldsymbol{x}||^2} \cdot e^{-||\boldsymbol{y}||^2} \sum_{\mathbf{n}_i \in \mathbb{N}_0^D} e_{\mathbf{n}_i}(\boldsymbol{x}) e_{\mathbf{n}_i}(\boldsymbol{y}),$$

where $\mathbf{n} \in \mathbb{N}_0^D = (n_1, n_2, \dots, n_D)$ with D equal to the number of input space dimensions. From the definition of $e_{n_i}(\boldsymbol{x}_i) = \sqrt{\frac{(2\gamma)^n}{n_i!}}\boldsymbol{x}_i^{n_i}$, we write the general form for the multi-index as

$$e_{\mathbf{n}_i}(\boldsymbol{x}) = \sqrt{\frac{(2\gamma)^{n_1+n_2+\cdots+n_D}}{n_1!n_2!\dots n_D!}}\boldsymbol{x}_1^{n_1} x_1^{n_2} \dots x_D^{n_D}.$$

We know that a kernel function can be written as an inner product of mapping functions: $K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$. Given the general $e_{n_i}(\boldsymbol{x}_i)$ we can define a ϕ for the RBF kernel, generalized for any number of input dimensions, as $\phi(\boldsymbol{x}) = e^{-\gamma||\boldsymbol{x}||^2}\langle e_{\mathbf{n}_1}(\boldsymbol{x}), e_{\mathbf{n}_2}(\boldsymbol{x}), \dots\rangle$ for all $\mathbf{n} \in \mathbb{N}_0^D$.

From this definition we can define the approximate finite dimensional feature space using a mapping function $\phi_Q(\boldsymbol{x}) = e^{-\gamma||\boldsymbol{x}||^2}\langle e_{\mathbf{n}_1}(\boldsymbol{x}), e_{\mathbf{n}_2}(\boldsymbol{x}), \dots\rangle$ for all $\mathbf{n}_i \in \mathbb{N}_0^D$ where

$n_i \leq Q \ \forall \ n_i \in \mathbf{n}_i$. Note how this feature map is a truncated version of the infinite dimensional feature map ϕ. The hyper-parameter Q decides the number of dimensions of ϕ, and is called the *Quality* hyper-parameter. More precisely the number of dimensions of ϕ_Q grows exponentially with respect to D such that the number of dimensions equals $(Q+1)^D$ [1].

Figure 2.6 shows how Q determines how well the approximation matches the exact RBF kernel. We see that a higher Q allows for a more accurate classifier. A formal prove of this claim can be found in the original paper [17].

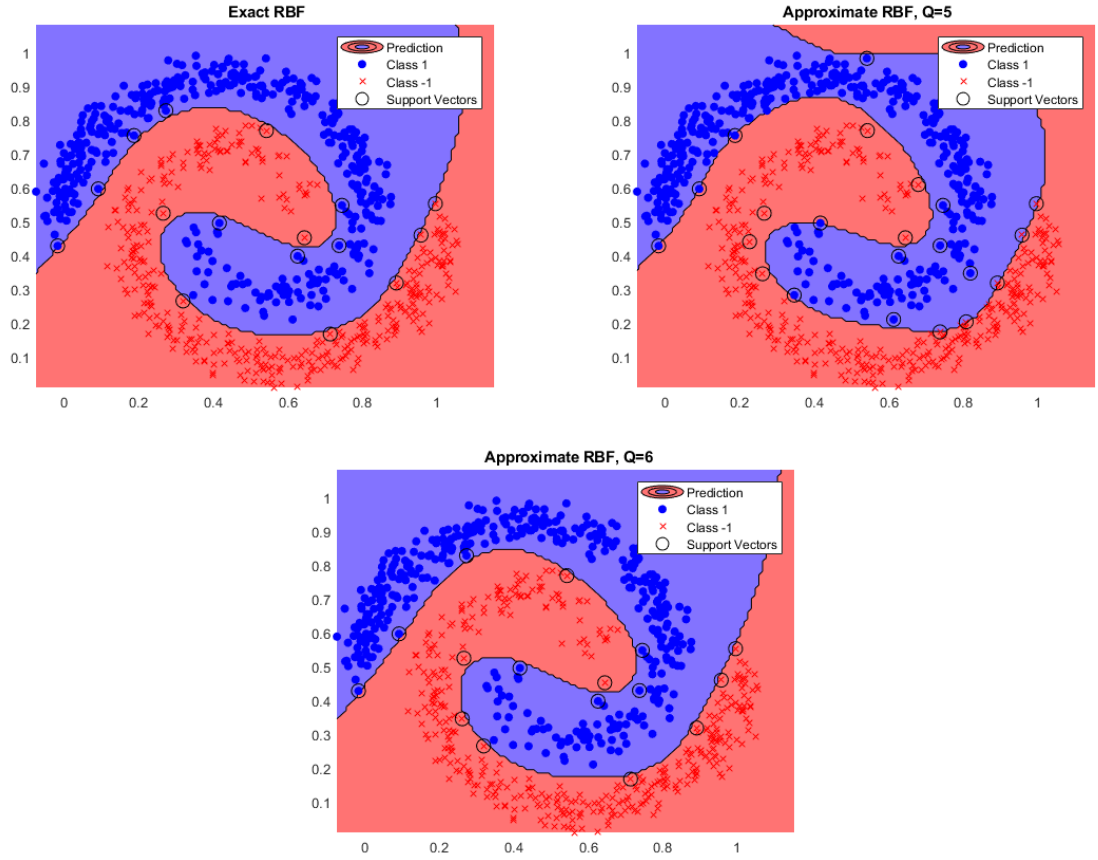

Figure 2.6: Plots of 3 SVMs, one using the RBF kernel, two using the approximate RBF kernel with Q=5 and Q=6 respectively. We see that the approximate decision boundary gets closer to the RBF decision boundary as the quality parameter Q increases.

[1] Note that the original paper [17] states a dimensionality of Q^D, this seems to be a mistake as there are $Q+1$ numbers which satisfy $n \leq Q \mid n \in \mathbb{N}_0$.

2.5 Morse Theory

Morse theory, referring to M. Morse's work [14], provides tools that will prove useful when optimizing topological complexity of SVMs. Morse theory provides theorems that show that a levelset homology of a *Morse function* only changes at specific point of a function, called *critical points*.

Critical points of a function f are defined as all points of f for which all partial derivatives of f are 0, meaning the slope at a critical point is flat. There are 3 types of critical points: *maxima*, *minima*, and *saddle points*. In order to determine the type of a critical point we use the *Hessian matrix* of f. A Hessian matrix is a matrix of all second order partial derivatives of f:

$$H_f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial^2 x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \frac{\partial^2 f}{\partial x_n \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial^2 x_n}(x) \end{bmatrix}.$$

The Hessian matrix of f gives us the change in slope of f. Using this we can determine for each critical point whether it is a minimum, maximum, or saddle. Let's consider a critical point x:

- The eigenvalues of $H_f(x)$ are all positive, then x is a minimum of f.
- The eigenvalues of $H_f(x)$ are all negative, then x is a maximum of f.
- The eigenvalues of $H_f(x)$ contains a combination of positive and negative values, then x is a saddle point of f.

We can further use the Hessian Matrix to determine whether some function f is a Morse function: f is a Morse function if and only if all critical points of f are *non-degenerate*. A critical point x is degenerate if the determinant of $H_f(x)$ equals 0 [14].

For Morse functions every critical point x comes with an index i which indicates the number of linearly independent directions the gradient is decreasing in x. Note that since we work with Morse functions there are no degenerate critical points. Formally, we define the index of a critical point x to be the number of negative eigenvalues of $H_f(x)$. So it follows that:

- A minimum is a point where the gradient increases in all directions, so the index of a minimum is 0.
- A maximum is a point where the gradient decreases in all directions, so the index of a maximum is n where n is the number of variables in f.
- A saddle point is a point where the gradient increases in at least one direction and decreases in at least one other direction, so the index of a saddle is between 1 and $n-1$. It follows that for functions with > 3 dimensions there can be saddle points with different indices.

An important notion in topology are *Betti numbers*: β, as introduced by H. Poincaré [15] (work translated into English by John Stillwell [19]). Specifically, the n-th Betti number β_n of a topological space represents the amount of unique n-dimensional holes in that space. Foregoing a more rigorous formal definition of Betti numbers, for the purposes of this thesis it is important to know that β_0 indicates the number of connected components of a topological space. The topological spaces we consider in this thesis are the n-dimensional $\mathbb{R}$ spaces $\mathbb{R}^n$, and undirected graphs.

With these definitions we can connect Morse Theory to topology by introducing Morse inequalities [14]: Consider a Morse function f, and let m_i denote the count of critical points with index i. Furthermore, let β_i represent the i-th Betti number. The Morse inequalities are then defined as follows:

$$\begin{aligned} m_0 &\geq \beta_0 \\ m_1 - m_0 &\geq \beta_1 - \beta_0 \\ &\vdots \\ m_d - m_{d-1} + \cdots \pm m_0 &\geq \beta_d - \beta_{d-1} + \cdots \pm \beta_0 \end{aligned}$$

These critical points and their indices relate to topology in a specific way. Imagine a flat plane starting at ∞ moving down, and we look at β_0 of the part of a Morse function f above the plane. We call this the superlevelset homology of f. Morse theory state that this β_0 can only change whenever the plane passes a critical point [12]. Figure 2.7 provides some visual intuition for this claim. For a more formal definition let $a, b \in \mathbb{R}$ such that $a > b$ and let $M^a = \{x | f(x) \geq a\}$ be a superlevel set of f, then $M^a_b = M^a - M^b$ such that it is the level set in the range $[b, a)$. It follows that if the Betti numbers of M^a differ from those of M^b then we know that M^a_b contains a critical point. And conversely, the Betti numbers of M^a will only be different to those of M^b if and only if M^a_b contains a critical point.

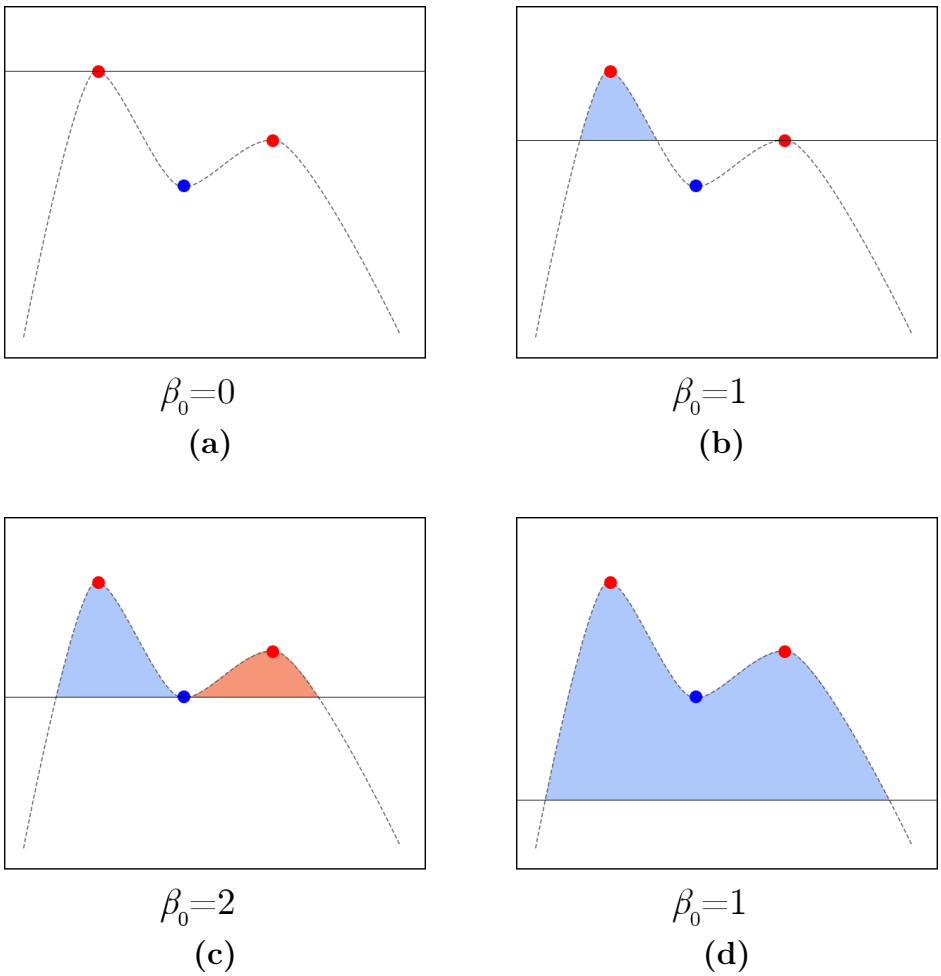

Figure 2.7: Maxima indicated by red dots, and minima by blue dots. We can see how superlevelset homology only changes once the lower bound passes a critical point. **(a)** shows a β_0 of 0 before passing through the highest maximum, and a $\beta_0 = 1$ after passing the first maximum up until reaching the second maximum **(b)**. With the lower bound between the second maximum and minimum we observe $\beta_0 = 2$ **(c)**, and finally with a lower bound below the minimum we see a final change in topology such that $\beta_0 = 1$ **(d)**.

2.6 Delaunay Graphs

Finally I will introduce *Delaunay graphs*, named after the B. Delaunay's paper [8], as they will help approximate topological complexity later in the methods section. Delaunay graphs, or Delaunay triangulations, are a specific type of graphs that can be generated for any set of points for which a distance between them can be determined, creating a triangulation with the given points as vertices. While there are other definitions, arguably the most useful way for this thesis is to define a Delaunay graph as the dual graph of a Voronoi diagram.

A Voronoi diagram, given some set of points $X = \{\boldsymbol{x}_1, \boldsymbol{x_2}, \ldots, \boldsymbol{x}_n\}$ and a space S,

slices S into regions such that each $\boldsymbol{x}_i \in X$ has a corresponding region R_i where for any $\boldsymbol{s} \in S$ $\boldsymbol{x}_i$ is the closest point in X. So formally we can describe R_i, given some distance function $d()$, as:

$$R_i = \{\boldsymbol{s} \in S \mid d(\boldsymbol{s}_i, \boldsymbol{x}_j) \leq d(\boldsymbol{s}_i, \boldsymbol{x}_k) \; \forall \; \boldsymbol{x}_j, \boldsymbol{x}_k \in X\}.$$

We call R_i a Voronoi cell. Figure 2.8 shows an example of such a Voronoi diagram. Note that, although figure shows a 2 dimensional space, this definition generalizes to any n-dimensional space.

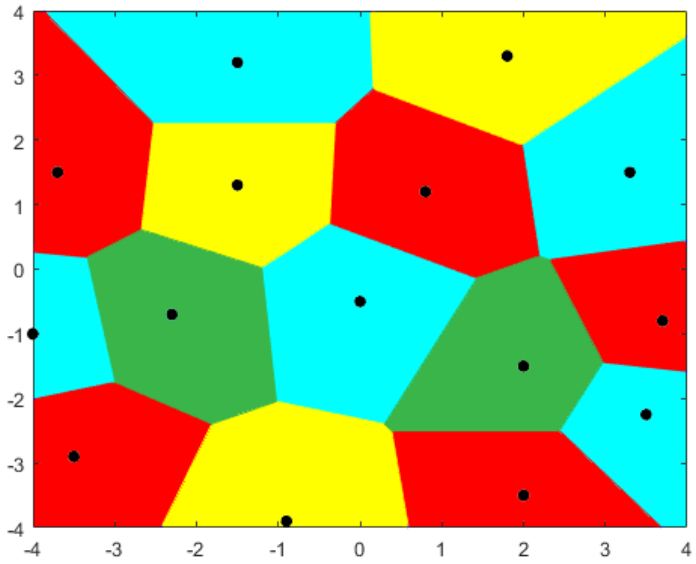

Figure 2.8: For a set of points X, illustrated with black dots, in a 2 dimensional space S considering the Euclidian distance, we can create a Voronoi diagram. Each Voronoi Cell R_i, indicated by the differently coloured areas, is the region of S where all points in R_i are closest to $\boldsymbol{x}_i \in X$.

If we now consider all points in X to be vertices of a graph, and create an edge between all $\boldsymbol{x}_i, \boldsymbol{x}_j \in X$ where their respective Voronoi cells touch we get the Delaunay graph. Figure 2.9 shows how this is equivalent to the dual graph of the Voronoi diagram.

As discussed these definitions generalize to any n-dimensional space. While the number of regions of a Delaunay graph is known to scale linearly with the number of points in 2-dimensional spaces, for higher dimensional spaces this scaling does not necessarily hold anymore. C. Duménil shows that an approximation of this complexity in 3-dimensional spaces is $O(n \ln n)$ [9], but this remains an approximation.

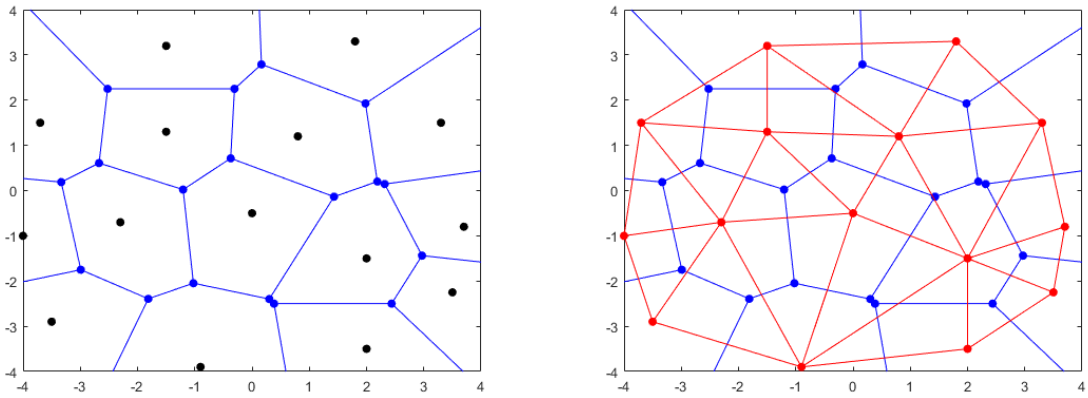

Figure 2.9: Given a Voronoi diagram connecting the points in X when their cells touch creates the dual graph of the diagram, the Delaunay graph of X. The points of X are shown in black. The boundaries of the Voronoi cells are indicated by blue edges, places where more than 2 cells meet are indicated with a blue vertex. The Delaunay graph is shown with red edges and vertices in the right plot.

2.7 Related work

2.7.1 Topological Regularization

While it seems there are no works yet into the exact topic of topological regularization of SVMs, there have been works that incorporate topology in the optimization of other machine learning models.

A paper by C.Chen, X. Ni, Q. Bai, and Y. Wang [5] proposes a novel topological regularization method that aims to punish the topological complexity of the decision boundary, similar to the goal of this paper with the caveat being their focus on neural networks. Their paper not only finds a way to measure topological complexity of the decision boundary for a neural network, but also shows they can find a gradient for this complexity measure. The significance of this is that the existence of this gradient means that the the topological complexity can be added to the loss function for neural networks, such that it topology will be optimized during the learning phase. Unfortunately the paper does not compare results of their method against a baseline neural network, but they do show increased performance over other machine learning methods.

A regularization method using topology for Convolutional Neural Networks (CNN) is proposed by J.R. Clough, N. Byrne, I. Oksuz, V.A. Zimmer, J.A. Schnabel, and A.P. King [6]. This method does not consider the topology of decision boundaries, but rather

segmentation topology of images. First the paper outlines how persistent homology of an image can be determined in a meaningful way using a filtration method based on pixel values. Then the paper continues to describe a loss function which penalizes differences in the persistent homology between the input data and encoded data. This function is derivable, meaning the neural network can be optimized following the standard gradient descent algorithm.

2.7.2 Topology of Decision Boundaries

A paper by M. Masden et al. [13] shows how exact decision boundaries can be constructed for fully connected neural networks using the ReLU activation function. Consequently some topological analysis is possible on these decision boundaries such as identifying its Betti numbers, as shown in their paper. While their approach is specific to RelU Neural networks, it does illustrate how decision boundaries can sometimes be exactly constructed using algebraic analysis.

Work by K.R. Varsney and K.N. Ramamurthy [21], in a paper proceeding their work on the labeled Čech complex [16] (discussed later), introduces a novel way to find persistent homology of a decision boundary. The proposed method tackles a similar problem to this thesis, in a reverse manner. Whereas I aim to improve topology of decision boundaries post hoc, their paper determines topology of decision boundaries beforehand and use that information to help in Kernel function selection and tuning.

2.7.3 Finding Decision Boundaries

Often decision boundaries are not easily extracted from machine learning models as they emerge from complex functions instead of being strictly defined. There are previous works that tackle the problem of finding the decision boundaries in various ways. Research in this area seems to be mainly focused on neural networks, with little works found attempting to construct the decision boundary exploiting the properties of SVMs.

In many cases decision boundaries are not exactly constructed, but rather estimated through sampling the input space. H.Kamari and J. Tang [11] propose a method that aims to optimize and refine this sampling of the decision boundary using a method dubbed DeepDIG: **Deep D**ecision boundary **I**nstance **G**eneration. As the name implies DeepDIG generates samples that help estimate the shape of decision boundaries for deep neural

networks. Samples are generated through autoencoder based methods such that the samples are misclassified by the model, these are called adversarial samples. DeepDig then refines these adversarial samples to move them closer to the decision boundary, while still being misclassified. Now these samples can be used to approximate the decision boundary, as they are known to be close to it. Their presented results show high accuracy in the estimation of the decision boundaries over a number of different models and datasets (`MNIST`, `FashionMNIST`, and `CIFAR10`).

Through an added restriction on the Čech complex [10, p. 72–75] K.N. Ramamurthy, K.R. Varshney, and K. Mody [16] introduce the labeled Čech complex. This labeled Čech complex specifies a class $c_a \in \{-1, 1\}$ for any 0-simplex a in the complex. Then 1-simplices are only generated between 2 0-simplices a, b if $c_a \neq c_b$. The 0-simplices are the datapoints used to train a machine learning model, and the class of a 0-simplex is simply the class of its corresponding datapoint. The paper illustrates how this labeled Čech complex can be used to approximate the topological complexity of a decision boundary. This method is purely dependent on datapoints and their predicted labels. Consequently, this sampling method can be used for any binary classification model.

Chapter 3

Methods

In this section I will show the necessary steps and algorithms to both quantify the complexity of the decision boundary, and attempt to optimize it. It is important to note that the optimization relies on kernels have a mapping ϕ that has a feature space of finite dimensions. From this point on ϕ is assumed to meet this criterion, unless stated otherwise. This means that the following approach will not work for the exact RBF – and Sigmoid kernel function (2.3.1). The methods are implemented in Matlab, the source code can be found on GitHub at `https://github.com/WillemSch/Topological-Regularization-SVM`.

3.1 Quantify Complexity of Decision Boundary

Since the objective is to decrease complexity of the decision boundary, first I define a measure of this complexity. Let the complexity of a decision boundary be the 0th Betti number β_0 of the input space when cut by the decision boundary (see Figure 3.1). For SVMs this is equivalent to the number of components of the hypersurface generated by ϕ in the feature space when cut by the linear separator, assuming the mapping function ϕ is continuous.

One logical direction to explore is that of topology of algebraic varieties, the intersection of multiple polynomial functions. This is because ϕ can consist of polynomial functions, and a hyperplane can be described using a degree 0 polynomial this could prove to be a valuable tool. Unfortunately methods in this area seem to provide only approximations or bounds on the Betti numbers of such varieties [2, 20]. Many of these methods

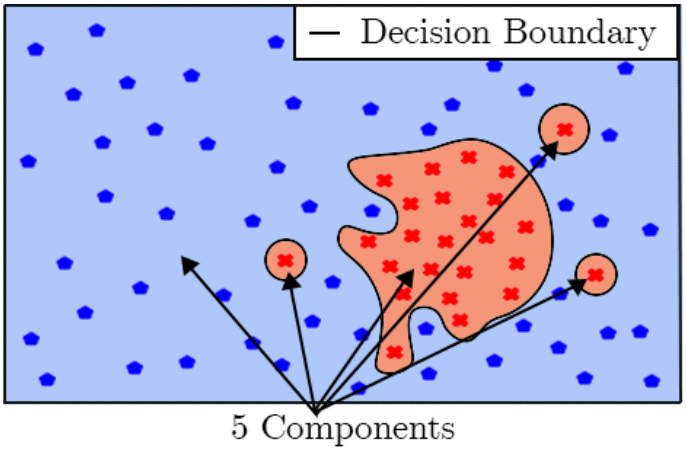

Figure 3.1: A decision boundary cuts the input space into separate components. In this example we see how the decision boundary cuts the input space into 5 components: 1 blue component, and 4 red components.

only consider the degrees of the polynomials when determining the approximations. Such approximations would be pointless for my purposes as I aim to analyse changes in β_0 by moving the linear separator, thus not changing degrees of the polynomials generating the intersection.

So, alternatively, in this thesis I determine β_0 through an approximation by sampling the input space. First I create a n-dimensional grid $G = (V, E)$ in the input space, where n equals the number of input space dimensions, V are the vertices of the grid, and E the edges connecting adjacent vertices. This gives us a discretized approximation of the input space. To create such a graph I create an empty graph and add vertices one at a time in order of their indices of each dimension, going through all dimensions in an arbitrary order. When a vertex is added to the graph I create edges between the new vertex and vertices whose indices are 1 lower in exactly 1 dimension. Repeat until all vertices are added to the graph. Second I remove all $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in E$ where $predict(\boldsymbol{v}_1) \neq predict(\boldsymbol{v}_2)$ where $predict(\boldsymbol{v}_i)$ is the predicted class of $\boldsymbol{v}_i$. This is equivalent to removing edges in between vertices which are on different sides of the linear separator in the feature space. Finally we count the amount of connected components of G which gives us an approximation of β_0. Note that the accuracy of the approximation is dependant on the resolution of the grid, which in turn determines computation time.

As the number of vertices of the grid graph grows exponentially with respect to the number of dimensions, it can quickly become impracticable to use. An alternative to grid sampling in higher dimensional spaces is random sampling. Random sampling lets the user decide exactly how many samples will be considered when approximating the input

space, and with that control the computation time. As we are interested in connected components in the sliced input space we need to connect the random samples into a graph, similar to the grid graph. For this I create a Delaunay graph (2.6) from the random samples. This Delaunay graph is then a randomly sampled discrete approximation of the input space.

To count the number of connected components in G we perform a Depth First Search (DFS) on a vertex in the graph. Each vertex we visit during DFS will be labeled as such. If after the DFS there still are unvisited vertices then we start a new DFS on one of them, repeating until all vertices are found. For each DFS we increment the counter of connected components by one.

3.2 Finding the Feature Space Separator

As a result of how SVMs function I cannot immediately extract the learned linear separator of the feature space. It is, however, possible to reconstruct the hyperplane given that the ϕ maps to a finite dimensional feature space. In order to construct this hyperplane I need both the normal vector $\boldsymbol{w}$ and the bias b term such that the hyperplane can be described as $\boldsymbol{w}^T\boldsymbol{x} + b = 0$. The bias b is already computed for the SVM and can be easily extracted. The normal vector $\boldsymbol{w}$ needs to be constructed from the support vectors and corresponding α-values obtained during the Lagrange optimization [7] (2.3.2). The weights $\boldsymbol{w}$ can then be found using:

$$\begin{gathered}\boldsymbol{w} = X\alpha Y, \\ \text{st. } X = \{\phi(\boldsymbol{x}) \; \forall \; \boldsymbol{x} \in \text{Support Vectors}\}.\end{gathered}$$

Here α and Y are the alpha values and labels respectively corresponding to the support vectors $\in$ Support Vectors, and ϕ is the mapping function generated by the kernel. As here we rely on a finite dimensional feature space, in the case of RBF kernels we substitute in the approximate ϕ_Q (2.4.1).

3.3 Locating Critical Points

Recall that a point is a critical point if and only if all partial derivatives are zero (2.5) for that point, in other words: the tangent hyperplane is parallel to the coordinate hyperplane. Consequently, a point on a vector valued function ϕ is a critical point if the

derivative vector is perpendicular to the normal vector of the coordinate hyperplane. Consider the special case where the linear separator is parallel to the coordinate hyperplane. Then moving it along its normal vector $\boldsymbol{w}$ will provide the useful constraint that the superlevelset homology only changes at the critical points of ϕ (2.5).

If the linear separator is not parallel to the coordinate hyperplane we can use almost the same approach to determine the superlevelset homology. To do this we consider the normal vector of the linear separator $\boldsymbol{w}$ as the 'up' direction, instead of the normal vector to the coordinate hyperplane. In this case we will consider a point a critical point if and only if the derivative vector is perpendicular to $\boldsymbol{w}$. This is analogous to rotating both ϕ and the linear separator among the origin such that the direction of $\boldsymbol{w}$ is equal to the direction of the normal vector of the coordinate hyperplane. This means that we don't have to perform these rotations to find the distance from the linear separator to any critical points of ϕ with respect to the separator.

So in order to find the critical points of ϕ with respect to some normal vector I first compute all partial derivatives $\frac{\partial \phi}{\partial x}$ of ϕ for all $x \in$ input space dimensions. When working with an RBF kernel the approximate ϕ_Q is used here (2.4.1). Then I create the equations $\boldsymbol{w} \cdot \frac{\partial \phi}{\partial x} = 0$ for all partial derivatives. I find the solutions that hold for all the equations, which give us the critical points in the input space. Finally I obtain the feature space coordinates by mapping the critical points to the feature space using the ϕ mapping.

3.4 Method 1: Parallel Transformation of the Linear Separator (PTLS)

3.4.1 Algorithm

One way to change the complexity of the decision boundary is to move the linear separator such that it intersects critical points of the mapping function ϕ. Recall that the distance from a point $\boldsymbol{x}$ to a hyperplane $\boldsymbol{w}^T X + b = 0$ is calculated by $\frac{\boldsymbol{w}^T \boldsymbol{x} + b}{||\boldsymbol{w}||}$. Using this formula I determine the required bias term $b_{\boldsymbol{x}}$ such that the hyperplane intersects a given point $\boldsymbol{x}$ by solving the equation

$$\frac{\boldsymbol{w}^T \boldsymbol{x} + b_{\boldsymbol{x}}}{||\boldsymbol{w}||} = 0.$$

Solving the equation for $b_{\boldsymbol{x}}$, gives:

$$b_{\boldsymbol{x}} = -\boldsymbol{w}^T \boldsymbol{x}.$$

Substituting in the new bias term $b_{\boldsymbol{x}}$ in the equation for the linear separator we have effectively moved it to intersect the given point $\boldsymbol{x}$, without rotating the hyperplane. In order to also remove the critical point from the decision boundary, and to allow for floating point precision errors I move the bias by an extra epsilon value in the distance it is moved, such that $b_{\boldsymbol{x}} = b_{\boldsymbol{x}} + \epsilon \text{ sign}(b_{\boldsymbol{x}} - B)$ where B is the original bias of the SVM and ϵ is an arbitrarily chosen small value. I find the bias $b_{\boldsymbol{x}}$ for each critical point $\boldsymbol{x}$ using the method described in section 3.3.

Note how the methods do not compute or check the Hessian matrices for the critical points and therefore cannot determine whether they are degenerate or not. Section 2.5 shows how the Morse inequalities only apply to Morse functions, which require that all critical points are non-degenerate. This is however not an issue for my methods. Degenerate critical points with respect to some hyperplane, by their nature, only exist when the hyperplane is at some exact angle. Any infinitesimal deviation from this angle degenerates the critical point into 0, or into several other critical points. For this reason, encountering a degenerate critical point is rare. Furthermore, even if there is a degenerate critical point, the claim that superlevelset homology only changes at critical points still holds. That is; the presence of a degenerate critical point does not make it possible for superlevelset homology to change at any levels that do not intersect a critical point.

3.4.2 Performance Measures

Finally I substitute in the biases as found through the method described before into the SVM. We can then measure the accuracy of the SVM again to check for change. Furthermore we can extract some more measures which can provide insight such as the β_0 of the decision boundary and the distance measure which tells us the distance the linear separator is moved in terms of the size of the margin D_m. Let m be the margin and d be the distance we moved the linear separator among its normal vector then the distance measure $D_m = \frac{|d|}{m}$. The distance measure D_m provides some intuition of how extreme the movement of the linear separator is without relying on the scale of the data.

3.5 Method 2: Artificial Addition of Critical Points (AACP)

The PTLS method described in the section before moves the linear separator along its normal vector, such that some critical points are in– or excluded in the superlevelset generated by ϕ and the hyperplane of the separator. Here I will describe a second method, the *AACP* method, that instead of parallel transformations of the hyperplane can also introduce rotational transformations. We achieve this by adding a datapoint at a critical point of ϕ with respect to the linear separator. Then I fit a new SVM on this extended dataset. Through the Lagrange optimization we find a new optimal separator, which is not necessarily equivalent to a parallel transformation of the hyperplane.

Recall that SVMs have a parameter C that defines the importance of datapoints being inside their respective margin during fitting (2.3.2). In order for this optimization to work as you might intuit the SVM must have a sufficient high C, such that it does not allow the newly introduced datapoint to be misclassified. More specifically, the SVM must not be allowed to misclassify the newly added datapoint. With this in mind we can rewrite the SVM optimization goal as

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{i=1}^{n} \mathrm{C}_i \zeta_i.$$

Here the general scalar C is changed to a collection of specific C values specific to each datapoint, such that C_i is the specific C value corresponding to i-th datapoint. As a result, this approach is limited to hard margin models, or models with a high general – or specific C value. For later experiments I will use the scalar C version, because SVM implementations commonly do not allow for a collection of specific C values.

3.5.1 Algorithm

First I find the critical points with respect to the linear separator in the feature space. I do this using the methods as described in sections 3.2 and 3.3. Then I add one of the critical points to the dataset with the opposite label than the originally predicted label. This is because I want the critical point to now be moved to the other side of the hyperplane slicing the feature space, similar to the PTLS method. After I added the critical point I fit a new SVM, which will calculate a new optimal hyperplane to separate the feature space, considering this newly created datapoint.

3.5.2 Performance Measures

To measure the results of the AACP method we again approximate β_0 of the newly fitted SVM. Furthermore, I measure the accuracy of the SVM in the same way as in method 1. The distance measure D_m as used for method 1 does not work for this method. This is because the linear separator is no longer simply moved, but found again through the Lagrange optimization of the SVM problem. Therefore I do not consider any distance measure like D_m.

3.6 Model Selection

Both the PTLS– and AACP method generate a new model for each critical point found in the feature space and the baseline unaltered model, making it so we have to decide which model is best. As the goal is topological regularization one method that at a glance makes sense is defining some loss function that weighs accuracy and topological loss. Such a loss function would look like

$$\mathcal{L} = \mathcal{L}_{\text{accuracy}} + \alpha\, \mathcal{L}_{\text{topology}},$$

With $\mathcal{L}_{\text{accuracy}} = 1 - \text{accuracy}$, $\mathcal{L}_{\text{topology}} = \beta_0$, and α some weighting hyperparameter specifying the importance of penalizing topological complexity versus optimizing accuracy. The problem with this approach is easiest shown with an example: Let M_a, M_b, M_c be models created during the regularization, M_a, M_b, M_c all have some $\mathcal{L}_{\text{accuracy}}$ and $\mathcal{L}_{\text{topology}}$. The plot in figure 3.2 shows that $\mathcal{L}$, given $\mathcal{L}_{\text{accuracy}}$ and $\mathcal{L}_{\text{topology}}$ is a linear function with respect to α. Moreover we see that the best model changes at discrete values of α, so if you were to sample α values between these discrete points nothing will change in your final model. This makes α quite a poor hyperparameter to be optimized.

This leaves us with a simple alternative approach for selecting the best model by picking the model with best accuracy on a validation dataset. Formally this means that $\mathcal{L} = \mathcal{L}_{\text{accuracy}}$. This approach does in practice deviate from convention. Following this approach you obtain a set of models after training of topologically regularized SVMs, which have to go through this model selection step before the ultimate model is found. This is in contrast to conventional methods, where you obtain one single model after training.

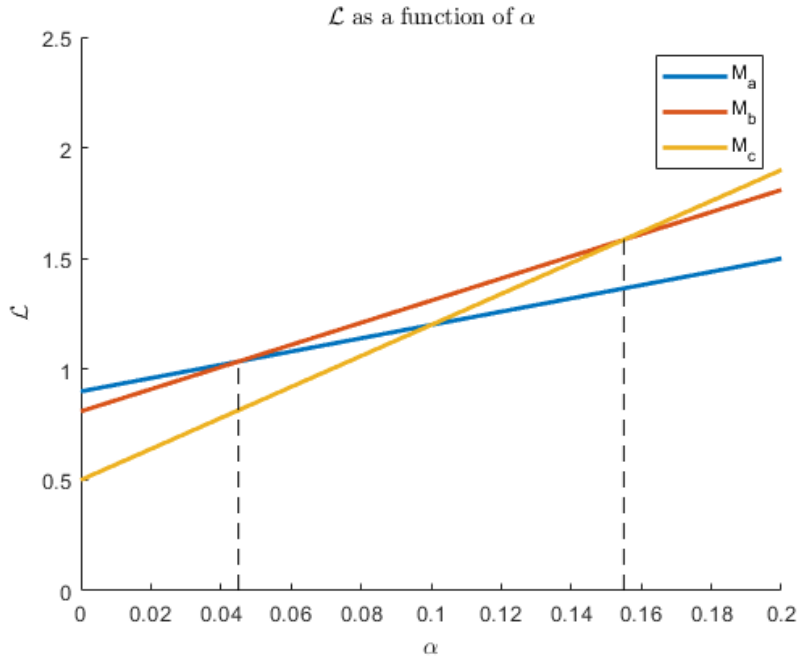

Figure 3.2: With the total loss $\mathcal{L}$ plotted as a function of the scaling factor α we see that the best model is constant throughout certain intervals of α, and each model will only be the best during at most one of these intervals. Here the ($\mathcal{L}_{\text{accuracy}}$, $\mathcal{L}_{\text{topology}}$) are (0.9, 3) for M_a, (0.81, 5) for M_b, and (0.5, 7) for M_c.

Chapter 4

Experiments

I will show the setup and results of several experiments using my method on SVM classifiers. Note that the first 2 experiments are examples specifically made to show easy to understand cases where my method can help. Source code for the experiments can be found in the GitHub repository: `https://github.com/WillemSch/Topological-Regularization-SVM`.

For all experiments 2 separate datasets are created: a training dataset which is used to train the SVMs, and a testing dataset which is used to determine the performance of the models.

4.1 Experiment 1: Data Noise, Example Case

This first experiment intents to show how my proposed method can help remove topological complexity of a decision boundary caused by noise in data. This experiment is specifically designed to show this problem in a simple manner.

4.1.1 Method 1: PTLS

Setup

For this first experiment I created a kernel function which results in the mapping function $\phi(x) = \langle x, x^3 + x^2 \rangle$. For this example this training dataset is designed to result in a linear

separator parallel to the coordinate hyperplane cutting the function ϕ into 4 components. Figure 4.1 visualizes the decision boundary of the SVM by plotting the basis function, training data, and decision boundary. Note the datapoint at $x = 0$, this point is the noisy data which creates complexity in the decision boundary. The SVM is trained with C=∞, essentially enforcing a hard margin for the classifier. Note that the SVM implementation used does still allow misclassification with C=∞, presumably due to digital precision errors. I set the hyper-parameter $\epsilon = 2^{-10}$ when offsetting the bias following: $b_{\boldsymbol{x}} = b_{\boldsymbol{x}} + \epsilon \text{ sign}(b_{\boldsymbol{x}} - B)$ (3.4).

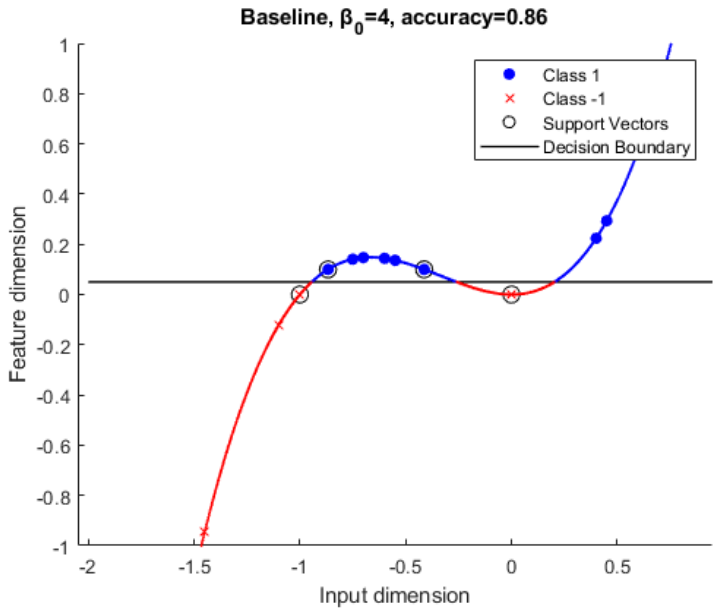

Figure 4.1: Noise in data can generate undesirable complexity in decision boundaries. Using a basis function $f(x) = x^3 + x^2$, the addition of the noisy datapoint $x = 0$ generates complexity in the decision boundary.

Results

After training the SVM on the training dataset grid and Delaunay sampling approximate β_0 of the model. The model's accuracy is determined for the test dataset. After obtaining the initial results, the bias of the original model is changed such that the linear separator intersects the critical points. For each critical point the distance measure D_m and the approximated β_0 as described in subsection 3.4.2 are noted, as well as the accuracy of the altered model on the test dataset. The results for each critical point can be found in table 4.1 with the results of the baseline SVM at the top. Figure 4.2 visualizes how the decision boundary changes when using the new found bias values.

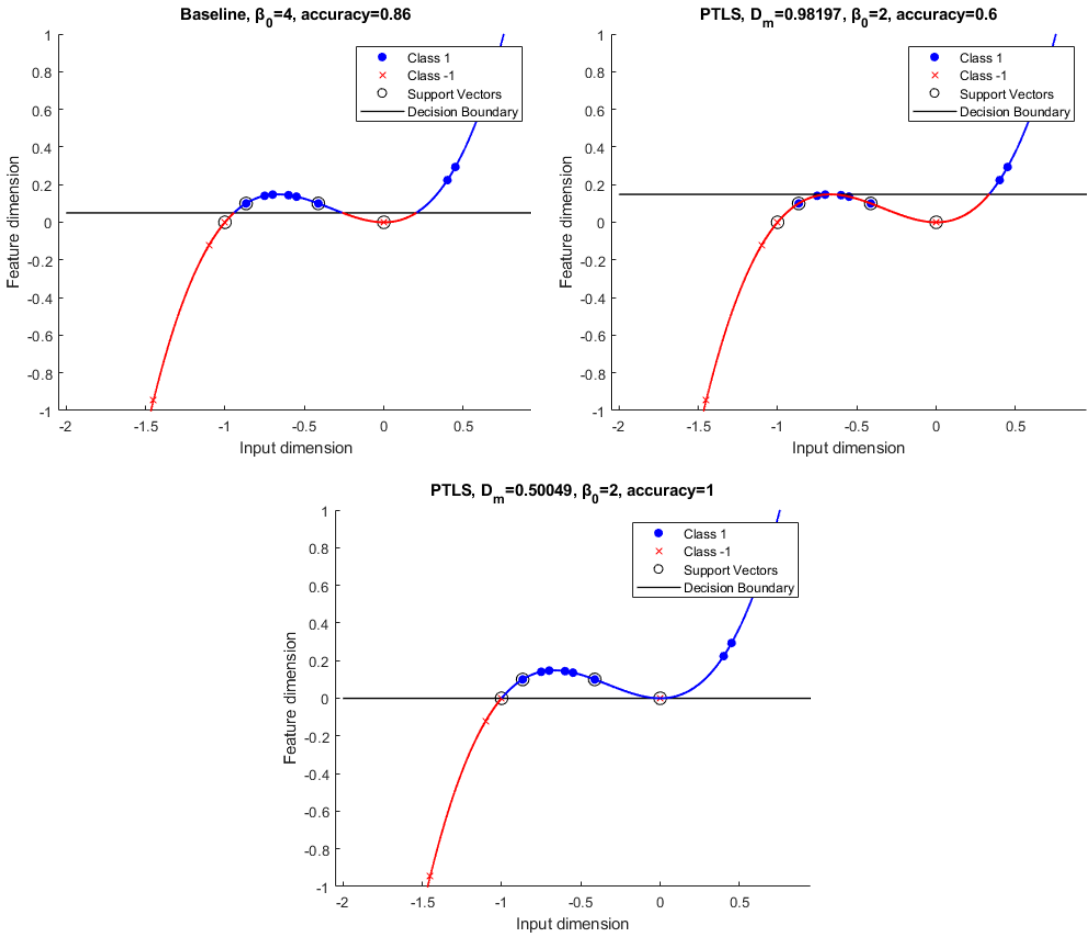

Figure 4.2: When changing the bias such that the linear separator intersects the basis function at critical points we see topological changes in the decision boundary. The basis function $f(x) = x^3 + x^2$ is shown to visualize the reason of the classification of any input data. The function is coloured to show the classification of parts of the function, the colours match with the colours for points in class 1 and -1.

Critical Point				
Input space	**Feature Space**	D_m	β_0	**Accuracy**
Baseline		0	4	78%
(-0.667)	$(-0.667, 0.148)$	0.98246	2	57%
$(\mathbf{0})$	$(\mathbf{0}, \mathbf{0})$	**0.50049**	**2**	**100%**

Table 4.1: Results of experiment 1, using the PTLS method

4.1.2 Method 2: AACP

For the second part of the experiment the AACP method (3.5) is used. Using the critical points found in the first part of the experiment a new dataset X_c is created per critical point $\boldsymbol{c}$ where $X_c = \{X,\ \boldsymbol{c}\}$, and $y_c = \{y,\ -\text{predict}(\boldsymbol{c})\}$. The notation $\{X,\ x\}$ defines a collection containing X appended by x. New SVMs are fitted on these datasets. The SVM implementation used uses the distribution of classes in the training data to prefer labels that are more common when predicting, unless specified otherwise. So, as the addition of 1 new datapoint creates a slight imbalance in the dataset, I explicitly set the prior to be that of a uniform distribution, removing the preference of predicting the most common training label.

For each SVM similar performance measures are calculated as for the PTLS method with the exception of D_m, as explained in section 3.5.2. These results for each critical point found can be found in table 4.2. Visualizations of the SVMs can be found in figure 4.3.

Critical Point				
Input space	**Feature Space**	D_m	β_0	**Accuracy**
Baseline		0	4	78%
(-0.667)	$(-0.667, 0.148)$	0.98246	4	57%
$(\mathbf{0})$	$(\mathbf{0}, \mathbf{0})$	**0.50049**	**4**	**100%**

Table 4.2: Results of experiment 1, using the AACP method

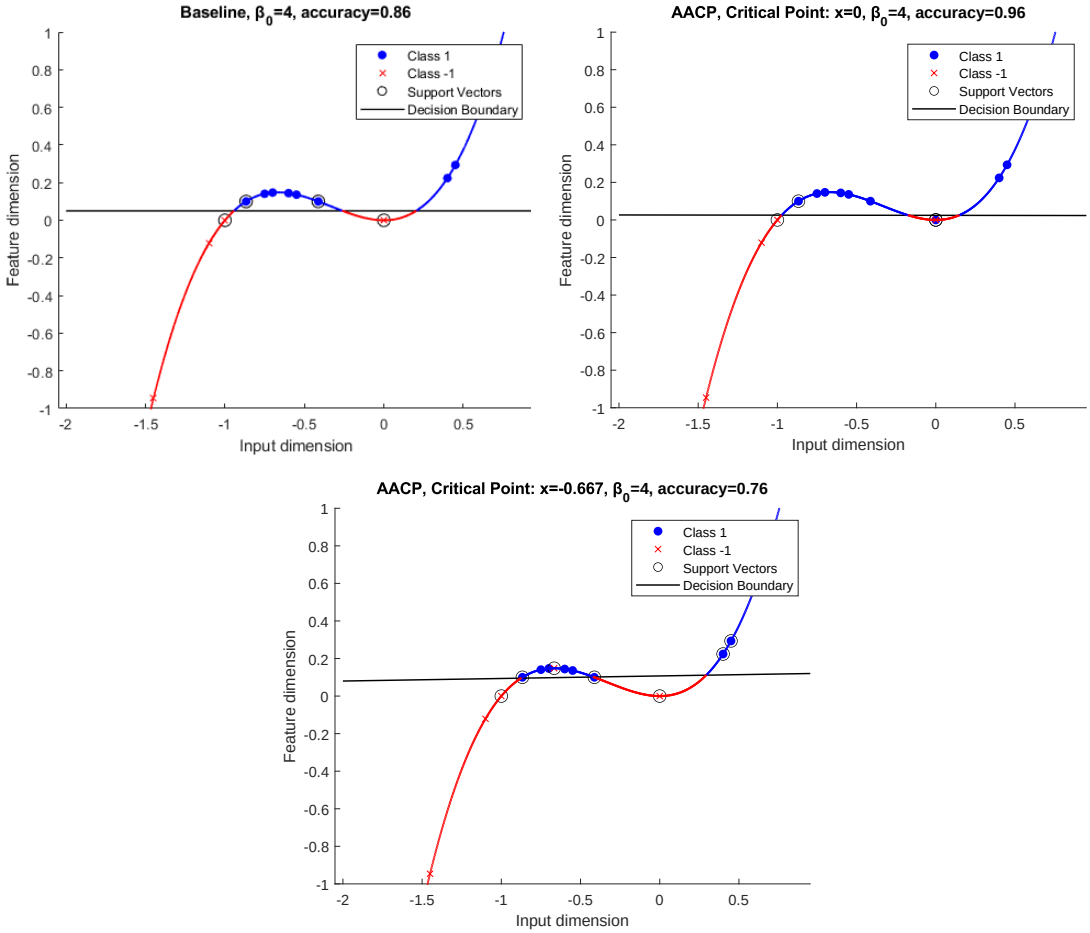

Figure 4.3: The AACP Method adds a critical point to the training dataset such that it is misclassified in the baseline SVM, after which an SVM is trained on this altered dataset. The basis function $f(x) = x^3 + x^2$ is shown to visualize the reason of the classification of any input data. The function is coloured to show the classification of parts of the function, the colours match with the colours for points in class 1 and -1. Note that in the last plot the added critical point overlaps an already existing point, making it so there are 2 support vectors of different classes at the same point.

4.2 Experiment 2: Poor Kernel Function, Example Case

This second experiment will show how the proposed methods can be used to reduce topological complexity for SVMs when a basis function is chosen which poorly fits to the data. Similar to experiment 1 the intention of this experiment is to show how undesirable complexity can emerge, and test my proposed methods' abilities to combat it. For this purpose this experiment is again designed such that this specific problem occurs, and not necessarily to be a realistic example.

4.2.1 Method 1: PTLS

Setup

First I create a kernel such that $\phi(x) = \langle x, x^4 - x^2 + 0.1x \rangle$. The SVM is trained with a hard margin constraint on the training dataset (C= ∞). Figure 4.4 shows how the choice of the basis function with this specific data generates a decision boundary with unnecessary complexity, hurting the model's ability to generalize. I set $\epsilon = 2^{-10}$ when offsetting the bias following: $b_{\boldsymbol{x}} = b_{\boldsymbol{x}} + \epsilon \text{ sign}(b_{\boldsymbol{x}} - B)$.

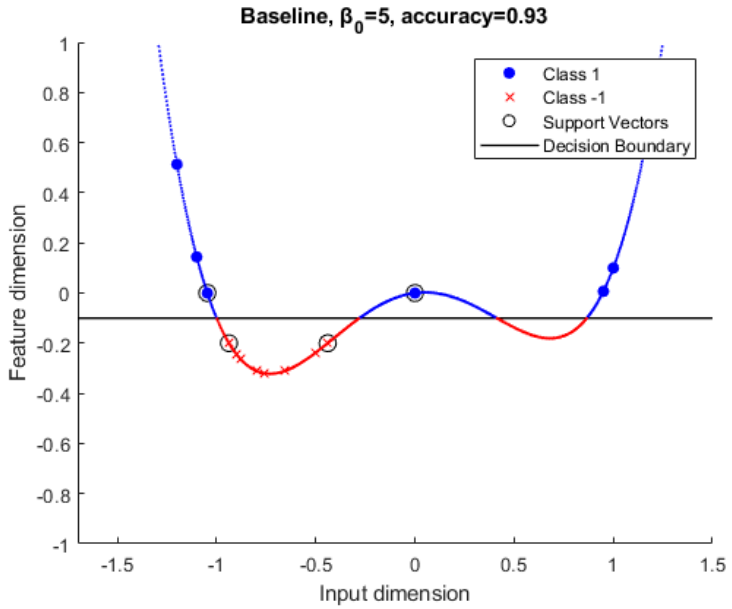

Figure 4.4: A basis function that poorly fits the pattern of the data can generate undesirable complexity in decision boundaries. In this case the basis function $f(x) = x^4 - x^2 + 0.1x$ generates complexity without noise in the data.

Results

Similar to the PTLS experiment the results are shown in table 4.4, as well as visualizations in Figure 4.6. The first row states the scores of the baseline SVM, and further rows state the scores of the model when the separator is made to intersect the stated critical point.

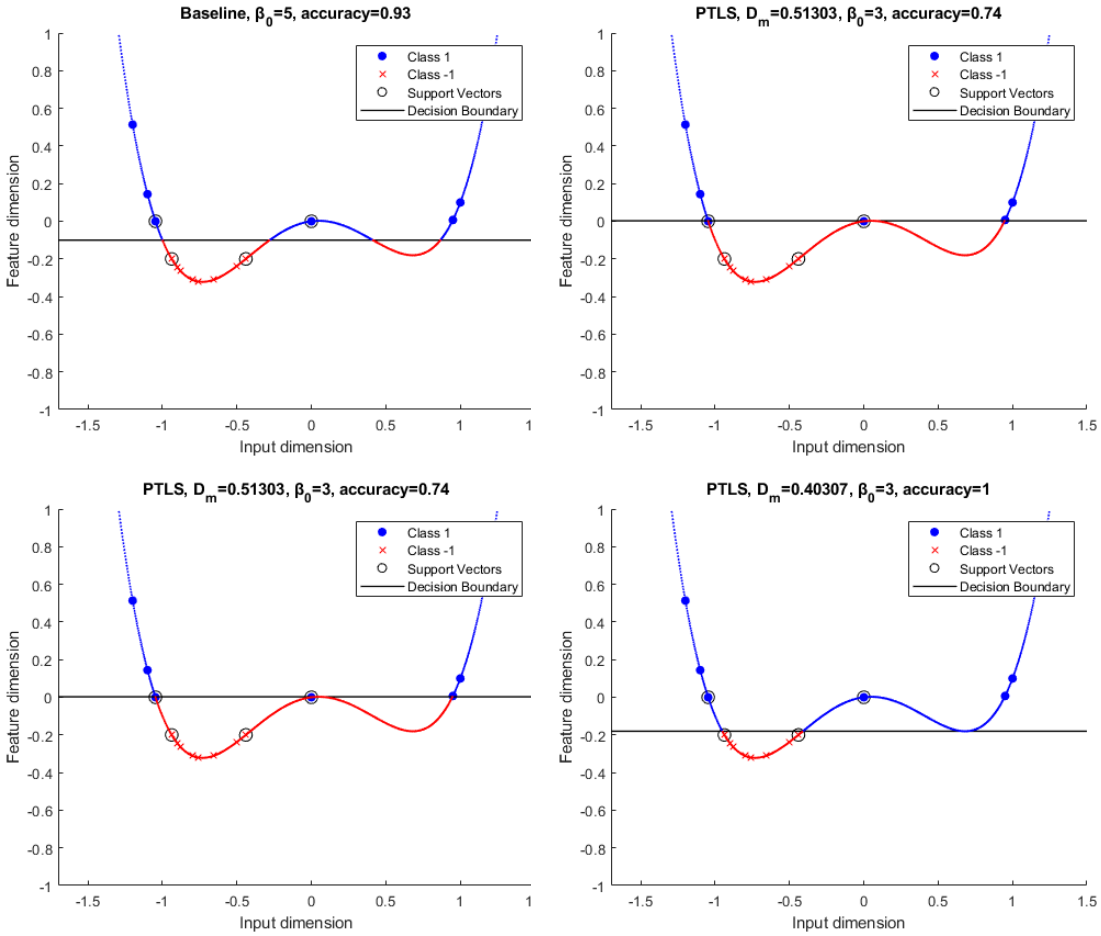

Figure 4.5: When changing the bias such that the linear separator intersects the basis function at critical points we see topological changes in the decision boundary. The basis function $f(x) = x^4 - x^2 + 0.1x$ is shown to visualize the reason of the classification of any input data. The function is coloured to show the classification of parts of the function, the colours match with the colours for points in class 1 and -1.

Critical Point				
Input space	**Feature Space**	D_m	β_0	**Accuracy**
Baseline		0	5	93%
(0.05)	$(0.05, 0.003)$	0.51245	3	49%
(0.681)	$(0.681, -0.181)$	0.40351	3	74%
$(\mathbf{-0.731})$	$(\mathbf{-0.731}, \mathbf{-0.322})$	**1.1084**	**1**	**100%**

Table 4.3: Results of experiment 2, using the PTLS method

4.2.2 Method 2: AACP

Here the AACP method is used with the same original SVM and orginal dataset as used for the PTLS method. The procedure here is the same as the second part of experiment 1, using the AACP method. Plots of the baseline, and all newly fitted models are shown in figure 4.6, with results shown in table 4.4.

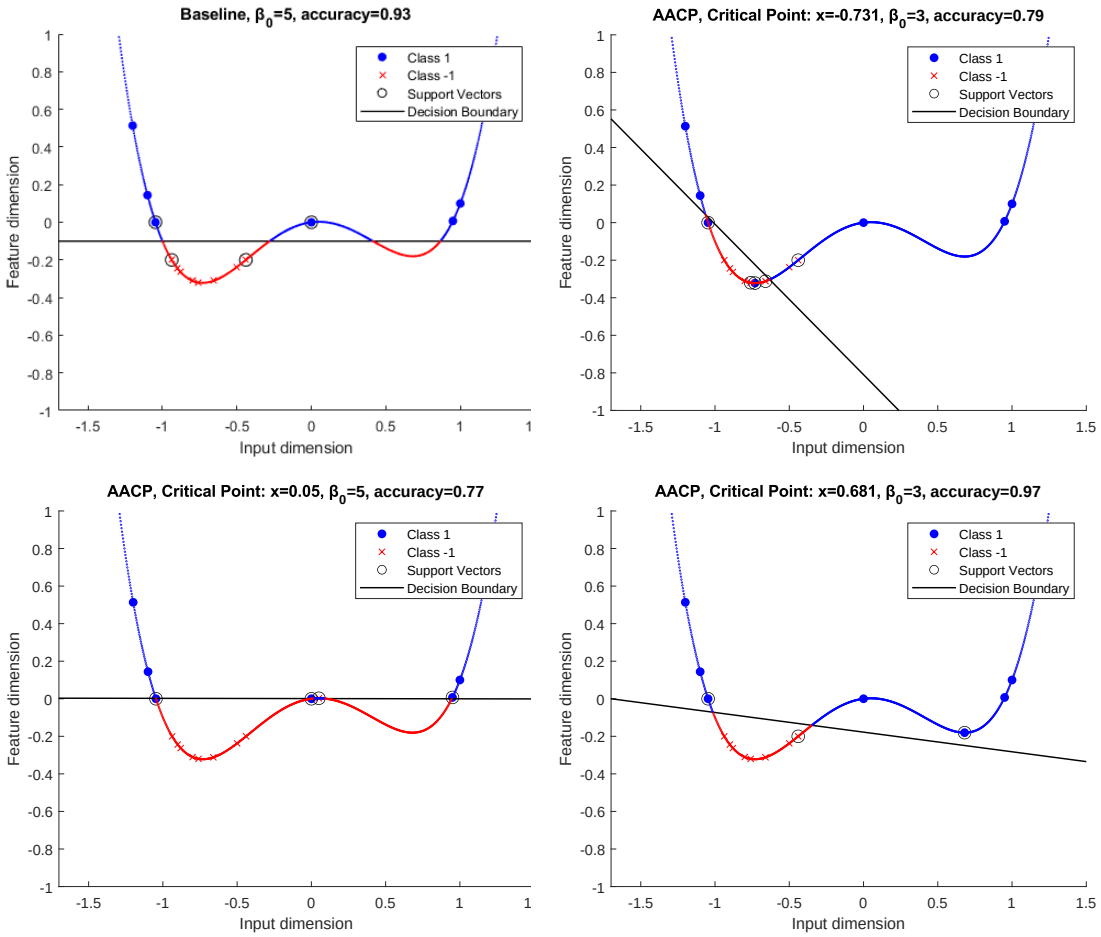

Figure 4.6: The AACP method adds a critical point to the training dataset such that it is misclassified in the baseline SVM, after which an SVM is trained on this altered dataset. The basis function $f(x) = x^4 - x^2 + 0.1x$ is shown to visualize the reason of the classification of any input data. The function is coloured to show the classification of parts of the function, the colours match with the colours for points in class 1 and -1.

Added Critical Point				
Input space	**Feature Space**	D_m	β_0	**Accuracy**
Baseline		0	5	93%
(-0.731)	$(-0.731, -0.322)$	1.1084	3	79%
(0.05)	$(0.05, 0.003)$	0.51245	5	77%
$(\mathbf{0.681})$	$(\mathbf{0.681}, \mathbf{-0.181})$	**0.40351**	**3**	**97%**

Table 4.4: Results of experiment 2, using the AACP method

4.3 Experiment 3: Approximate RBF kernel

This third experiment is meant to show the methods in a less contrived setting to gauge real world effects. As the goal of the methods is to help regularization of SVM models, in this experiment we start with a sub-optimal SVM. To make this experiment more relevant to real world use cases the SVM is trained following the approximate RBF method (2.4.1), instead of a contrived custom kernel.

Both the training and testing datasets are 2 dimensional datasets generated using a function. This function creates a balanced dataset between the 2 classes. Class 1 is generated inside a circle with diameter of 1, centered at (0.5, 0.5). Points in class -1 are generated in a ring with inner diameter of 0.25 and an outer diameter of 1.25, also centered at (0.5, 0.5). This pattern is visualized in figure 4.7. For these experiments the training and testing dataset contain 100 and 200 datapoints respectively.

4.3.1 Method 1: PTLS

Setup

Following the approximate RBF method I create a ϕ_Q, I choose a Q of 15 resulting in a $(15+1)^2 = 256$ dimensional feature space. I set $\gamma = 16$ in order to try to generate a complex decision boundary through overfitting.

First ϕ_Q maps all data to the high dimensional feature space. Note that this step is rather slow in my implementation, I assume this is due to the inefficiency of using the `subs` function. Then an SVM is trained on this data with using the linear kernel. Matlab's symbolic (exact) solver used to find the critical points in the previous 2 experiments works well on polynomial functions, but not necessarily well on more complex functions. The

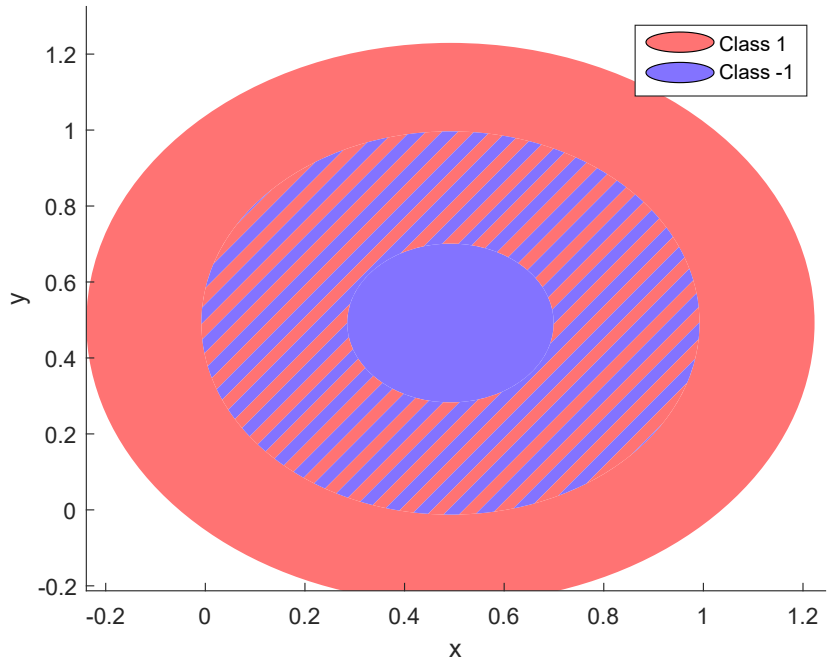

Figure 4.7: This figure shows the pattern for the classes of the generated datapoints for experiment 3. The red area indicates the space where datapoints for class 1 can be generated, and the blue area indicates the space where datapoints for class -1 can be generated. The striped area indicates the overlap in the blue and red area.

large number of feature space dimensions combined with the long running time of the symbolic solver made it impracticable to use in this experiment. Alternatively Matlab provides a numerical (approximate) solver which finds 0 or 1 approximate solutions each time it is run. Through random initialization the numerical solver can find different solutions each time it is run. This solver can be further optimized by specifying limits in which it looks for solutions. So I find critical points in this experiment by using the numerical solver with random initialization over 100 iterations and keep unique solutions, considering the limits $-0.2 < x < 1$ for both input space dimensions. I find the new biases and test them through substitution similarly to the previous 2 experiments.

Results

For each model I measured the accuracy on a separate test dataset, D_M, and β_0 as approximated through grid sampling and Delaunay sampling separately. These results can be found in table 4.5, with visualizations of the models in 4.8.

Critical Point	D_m	β_0 (Grid)	β_0 (Delaunay)	**Accuracy**
Baseline	0	3	3	63%
$(0.97558, 0.038774)$	0.1278	3	3	69.5%
$(0.64014, 0.16134)$	0.1233	3	2	70.5%
$(0.70768, 0.47673)$	0.1338	3	4	71%
$(0.29223, 0.069893)$	0.6008	3	4	56.5%
$(0.76266, 0.34814)$	0.1296	3	3	70%
$(-0.018606, -0.17567)$	140.1004	1	1	50%
$(-0.12281, 0.50861)$	58.0546	2	2	50.5%
$(0.82055, 0.68305)$	0.1287	4	4	70%
$(0.25193, 0.51062)$	9.4939	2	2	49.5%
$(0.11447, 0.32943)$	32.2307	1	1	50%
$(\mathbf{0.88205}, \mathbf{0.46421})$	**0.1271**	**2**	**3**	**73.5**%
$(0.80837, 0.63895)$	0.1287	5	5	69.5%
$(0.77168, 0.28105)$	0.1298	3	4	70%
$(0.20472, 0.64426)$	14.8038	2	2	48%

Table 4.5: Results of experiment 3, using the PTLS method

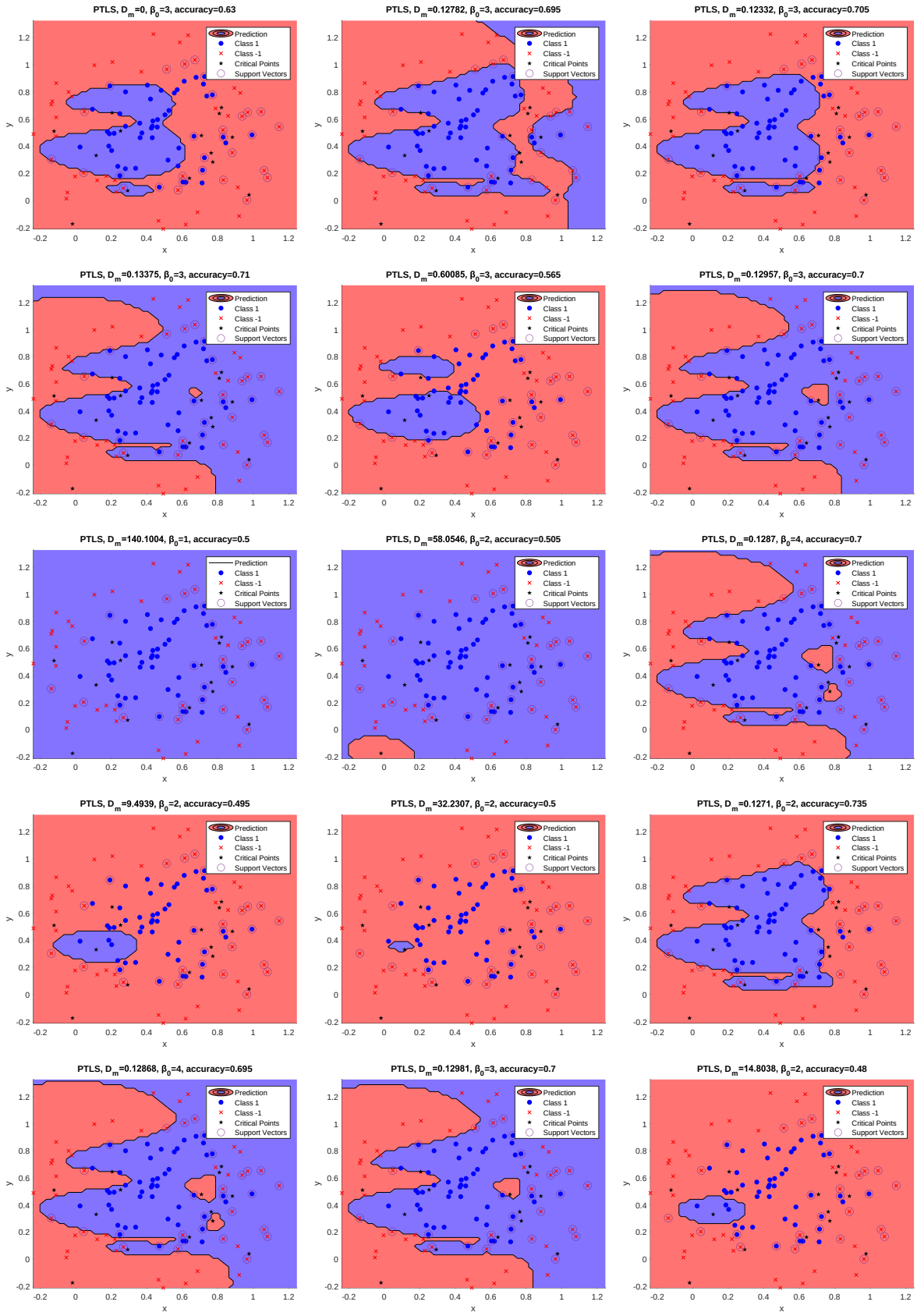

Figure 4.8: The models shown are all instances of an SVM using the approximate RBF kernel with $Q = 15$ and $\gamma = 16$. Each plot shows the input space decision boundary of a model where the bias is substituted for some new bias $b_{\boldsymbol{x}}$ such that the feature space separator is moved past some critical point $\boldsymbol{x}$. x and y indicate the first – and second input space dimension respectively. The first plot shows the unaltered model.

4.3.2 Method 2: AACP

The AACP method is performed similarly to the AACP part of the previous experiments. The approximated critical points found for the PTLS method are used as the critical points for this method as well. All new models are similar approximate RBF models with $Q = 15$, $\gamma = 16$, $C = \infty$, and prior to be that of a uniform distribution. The results of these models are shown in table 4.6, with visualizations of the model shown in figure 4.9.

Critical Point	β_0 (Grid)	β_0 (Delaunay)	**Accuracy**
Baseline	3	3	66.5%
$(0.97558, 0.038774)$	2	2	61.5%
$(0.64014, 0.16134)$	3	3	67%
$(0.70768, 0.47673)$	3	3	66.5%
$(0.29223, 0.069893)$	3	3	66.5%
$(0.76266, 0.34814)$	2	2	65.5%
$(-0.018606, -0.17567)$	4	4	66.5%
$(-0.12281, 0.50861)$	2	2	62.5%
$(0.82055, 0.68305)$	3	3	63.5%
$(0.25193, 0.51062)$	3	4	62.5%
$(\mathbf{0.11447}, \mathbf{0.32943})$	**3**	**2**	**75.5%**
$(0.88205, 0.46421)$	3	3	67%
$(0.80837, 0.63895)$	3	4	60.5%
$(0.77168, 0.28105)$	2	2	65.5%
$(0.20472, 0.64426)$	3	3	63%

Table 4.6: Results of experiment 3, using the AACP method

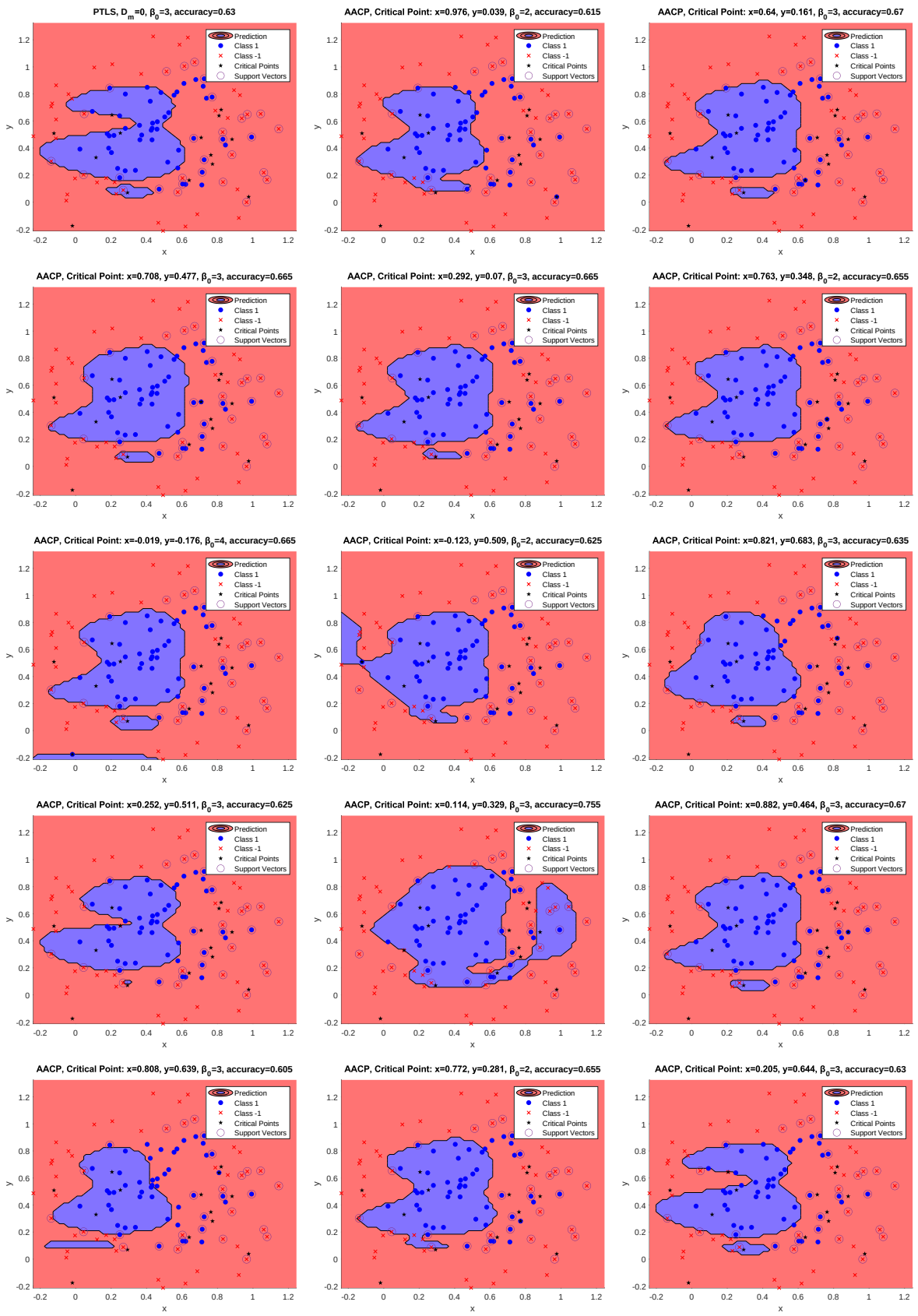

Figure 4.9: As the dataset used to fit the SVM is altered such that a single critical point $\boldsymbol{x}$ is added with a label $y_{\boldsymbol{x}} = -\text{predict}(\boldsymbol{x})$ a new model is created. The plots visualize the decision boundary in the input space. x and y indicate the first – and second input space dimension respectively. Here all models are approximate RBF models with $Q = 15$, $\gamma = 16$, and $C = \infty$.

Chapter 5

Discussion, Conclusion, and Future Work

5.1 Thesis Objectives

In section 1.2 I outlined the objectives set for my thesis. As to reflect on the results of this thesis, first let's state the objectives again:

- Measure the complexity of SVMs.
- Regularize SVMs through novel methods reducing the complexity.
- Show that the methods have a positive effect in practice.

In the following subsections I will outline which parts of the thesis relate to which objectives and discuss to what degree I was successful at tackling them.

5.1.1 Measure the Complexity of SVMs

The first objective set was to define some measure of complexity for SVMs and find a method to measure it. I have chosen the number of connected components of the input space as it is sliced by the decision boundary to act as this measure of complexity. For the sake of notational simplicity components of the input space as sliced by the decision boundary will be referred to simply as components. The intuition here is that many

small components can be an indication of overfitting, and can be replaced by fewer larger components. In the motivation (1.1) I show an example of how overfitting to noise can create an undesirable component in the input space. The second example shows how undesirable complexity can emerge from a poorly fitting mapping function. This shows that overfitting is not necessarily the only reason for undesirable complexity

Through literature review I have not found a way to exactly determine this complexity of an SVM, so I have decided to use an approximation method. The method I have chosen is the simple grid sampling method, where I discretely approximate the input space by graph approximation of the input space. For low dimensional input spaces grid graphs can be used for the approximation, but due to their exponential growth as the number of input space dimensions grow Delaunay graphs are an alternative for high dimensional spaces. By classifying each vertex, and removing edges between vertices with different classes, I can determine the number of connected components of the graph. This approximation serves as the measure of complexity for SVMs in this thesis.

Relating back to the objective: I have successfully defined a measure of complexity for SVMs and found a way to approximate it. So the objective has been mostly achieved, although an exact measure of complexity would be better.

5.1.2 Regularize SVMs by Reducing Complexity

I have proposed 2 methods which can reduce complexity of the decision boundaries of SVMs. Both methods are designed following the same general idea: changing the number of intersections of the linear separator with the surface generated by the mapping function ϕ changes the complexity of the SVM.

The PTLS method build on theorems from Morse Theory (2.5) which leads us to the conclusion that the complexity of the SVM only changes when the linear separator is moved past critical points of ϕ. This claim relies on the transformation of our linear separator to not alter the normal vector, but only its bias. So for the PTLS method I move the linear separator along its normal vector, such that it moves past critical points of ϕ. After this transformation I evaluate the SVM, to determine whether the SVM improved or not.

With the AACP method I attempt to address the restriction of the PTLS method int that PTLS does not allow for any rotations of the linear separator. I keep the intuition

provided by Morse theory that we want the linear separator to in– or exclude critical points. For the AACP method I again find the critical points of ϕ with respect to the linear separator. For each critical point I add it to the original training data, such that its label is opposite of the prediction of the original SVM. Then I fit a new SVM on this dataset and evaluate the model. A downside of this method is that there is no guarantee that we see changes in the complexity of the SVM, as rotational transformations of the linear separator can be introduced. Moreover the claim that complexity only changes by in– or excluding these critical points does not necessarily hold anymore either. This becomes clear from the definition of critical points: $\boldsymbol{x}$ is a critical point with respect to some hyperplane $\boldsymbol{w}X + b$ if the dot product $\boldsymbol{x} \cdot \boldsymbol{w}$ equals 0. We can see that when we rotate our reference hyperplane the critical points change as well, so they are no longer the same as the points added to the dataset.

Both these methods come with the restriction that they do not work for specific types of kernels, namely kernels for which the feature space cannot be defined in finite dimensions. I go over this and other weaknesses of my methods in a later section. As for reaching the objective, the experiments show that both methods have the ability to improve SVMs through topological regularization. This success is partially limited by the restriction mentioned before.

5.1.3 Show the Method's Positive Effects in Practice

The results of the experiments are shown in the experiments Section 4. The first two experiments were designed to show the problem that my methods aim to solve. We should not put too much weight on the measures from the experiments. What the experiments do show is how the two methods work, and their potential effectiveness. We can see that the PTLS method can successfully reduce complexity in a model, and given the right circumstances improve a model's ability to generalize because of it. We can also see that almost every alteration of the model, following the PTLS method, results in some change to the SVM's complexity. This is where the AACP method differs, as discussed before. The AACP method does not guarantee any change in complexity, and we can see this in the results of the experiments. However, from experiment 2 we can see the AACP method generating less complex decision boundaries, while improving accuracy. This shows a potential for the method to produce models with more desirable decision boundaries.

The third experiment shows how the methods work for a less contrived case, specifically for a poorly fitting approximate RBF SVM. Again we can see improvement in

performance when the linear separator is moved to different spots. However, the experiment does not necessarily show that the PTLS method improves an SVM by reducing topological complexity, as results also show more accurate classifiers where the topological complexity is similar or increased. The results do suggest that choosing the linear separator such that the margin is maximal is not necessarily the optimal solution.

5.2 Main Contribution

The methods I describe in this thesis provide novel approaches to optimize SVMs using the topology of their decision boundaries. As described, the PTLS method (3.4) will find a few alternative values for the bias of an SVM and with it some measures of its performance and complexity. In section 3.6 I show how we can choose the best model from the collection of models found. As the topological complexity plays no role in model selection, it only serves as an indication to show whether the original intuition of the thesis holds. That means that in practice this measure does not have to be computed.

The AACP method provides us with SVM models with altered linear separators through fitting a new SVM on an altered dataset. While experiments show the AACP method can produce more optimal decision boundaries, its behaviour is less predictable than the PTLS method and does not necessarily alter a model's complexity.

By the nature of this optimization there is no expectation of large changes in a model's performance, but it can provide means to refine models to increase their performance at least slightly. The large improvements in accuracy for the designed examples should not be taken as more than illustrative of potential. These examples exist to show what situations the proposed methods can be helpful in, in quite a contrived manner. As discussed above, looking at the results of the third experiment challenges the idea that the optimal linear separator is the one with the maximal margin.

Perhaps most interestingly, the results show that maximal margin is not always the optimal optimization criteria, and altering the bias can positively change the performance of an SVM. The PTLS method proposes an automated search for bias values that might improve the performance. So, in a way, we can view the bias of an SVM as a hyper-parameter that can be optimized after the learning phase. In this light, the PTLS method acts as an automated hyper-parameter search.

I have implemented the methods and experiments discussed in this thesis in Matlab. The code for this can be found on GitHub: `https://github.com/WillemSch/Topological-Regularization-SVM`. This code can be rerun, verified, and build upon by anyone.

5.2.1 Strengths

The main strengths of my proposed methods are shown in the previous sections, where I show that the methods have the potential to reduce topological complexity and the prediction accuracy of SVMs. Here I will outline a less obvious strength of this thesis.

Challenging the Maximal Margin Optimization Goal

One of the main takeaways from the results and intuitions is a challenge to the idea that the optimal linear separator for an SVM is the one that maximizes the margin. The results of the PTLS method specifically show that this is not necessarily the case, as some parallel transformation of the separator can improve an SVM. Moreover, the intuition that less topological complexity can indicate a better decision boundary suggests that some other optimization goal might result in a better classifier, if it considers such topological complexity.

5.2.2 Weaknesses

The topological regularization methods I propose in this thesis does, of course, also come with its weaknesses. In this section I attempt to outline all the weaknesses I found, knowing that this is likely not an exhaustive list. Where possible I add a potential way to alter or improve the methods, with the intention to inspire future work in this field.

Exploring Feature Space

The main weakness of my proposed method is that it requires us to explore the feature space. This is in contrast to the usual workings, and perhaps elegance, of the SVM, which through mathematical tricks does not require exploration of the feature space. Besides the potential computational and memory issues this may bring, this characteristic

of my method also makes it unsuitable to be used with kernels that map to infinite dimensions. The popular RBF kernel can therefore not be optimized using my approach. The requirement to explore the feature space comes forth from the use of Morse theory. If future works can find ways to determine the new bias values through other means, the reliance on the feature space might be removed, and therefore might benefit the most from the new regularization methods.

One way to get around this limitation is through finite dimensional approximations of ϕ for these infinite dimensional kernels. One such method is used in experiment 3. The approximation for the RBF kernel proves to be especially important for the methods I propose in this thesis, as RBF kernels have a tendency to generate topologically complex decision boundaries.

Finding Critical Points

In the proposed method for finding critical points we consider all critical points of the mapping function ϕ when attempting to optimize the classifier. For mapping functions with infinite critical points, such as sine functions, this methods will not work. A simple solution to this problem is to define a range where critical points are considered relevant, and ignoring all critical points outside this range. This is potentially even desirable to do even when working with finite critical points, as one might not care about complexity in decision boundaries outside of the range that any datapoint will be found. For experiment 3 such limits are used already to optimize performance.

Another downside of the reliance on critical points is computational complexity. I find critical points by solving a set of equations for all input features. Computation time grows significantly for each extra input feature. This is especially costly for symbolic (exact) solving of these equations. Alternatively one can use numerical solvers, which greatly improve computation time, at the cost of potentially missing critical points, and finding wrong critical points through approximation. This method is used for experiment 3, as the equations become highly complex when using the RBF approximate kernels.

Sampling Decision Boundary

The grid sampling method is used as it can find complexity in the decision boundary in spots where there is no training data, whereas sampling using the datapoints of the

training dataset, such as **DeepDIG**[11], cannot find these complexities. One downside of this grid sampling is however the exponential growth of the grid with respect to the input space. The grid will have m^n vertices where m is the resolution of the grid and n the number of input dimensions. Future work in defining the complexity of the decision boundary can experiment with different (sampling-) methods.

Furthermore, the proposed grid sampling provides an approximation of β_0, and not necessary the exact value. While I have not found a method which can provide any exact Betti numbers, I will not exclude that this is possible at all. Since we are looking at the topology of the intersection of a hyperplane and a function one approach could be tackling the problem using Algebraic varieties or Semi-Algebraic sets. In my review of literature it seems work on homology of algebraic varieties and semi-algebraic sets only provide estimates on Betti numbers. However, it seems possible such algorithms are found in the future.

As an alternative to grid sampling, I also show the performance of the approximation of β_0 through Delaunay graphs. These provide a more scalable and flexible solution as one can choose the exact number of vertices that generates the Delaunay graph. If one is interested in β_0 for any SVMs with a high-dimensional input space, this method would likely be the only viable option of the two used in this thesis.

Finally, one can just not do any sampling of the complexity, as it is not required for any of the functionality. However, it does provide insights in what the methods' impact are on decision boundaries.

5.3 Future Work

5.3.1 Penalizing Topological Complexity

While my proposed method allows for optimization of topological complexity of SVM decision boundaries, this optimization is performed after a linear separator has been found. A different approach can look into redefining the optimization goal of the SVM such that the optimal separator is not just the one with the greatest margin, but also punish topological complexity, similar to how topological losses have been introduced in loss function for neural networks [5]. By the nature of the optimization process of SVMs one will need to find a measure of topological complexity which is derivable, and fits with the Lagrange optimization.

5.3.2 Exploring the Feature Space

An idea unexplored in this thesis is using the approximate RBF to finding new biases and critical points for the PTLS – and AACP methods respectively. Then with these found values performing the methods on the exact RBF model, that is: substituting the bias $b_{\boldsymbol{x}}$ into the exact RBF model in the case of the PTLS method. If this works the methods proposed in my thesis can be made applicable to kernels with infinite dimensional feature spaces as well. The main reason no experiments in this thesis do this is due to implementational difficulties: the SVM implementation used does not allow for the bias variable to be changed, and time constraints did not permit implementing a work-around.

5.3.3 Iterative AACP

The second method I propose in this thesis is the AACP method, where a new SVM is trained on a dataset where a critical point is artificially added. In the experiments the best model is chosen once this step is done. This leaves an interesting alternative to look into: iteratively performing AACP. Instead of stopping after choosing the best model, one can repeatedly keep performing the AACP method in an attempt to further optimize the SVM. Whether such an iterative approach improves the methods, and if so what stopping criterion is useful, might prove an interesting topic of research.

5.4 Conclusion

The experiments show there are cases where my proposed regularization methods do improve the performance of an SVM classifier. This shows there is merit to exploring such optimization techniques further. The proposed methods do come with their weaknesses and points of improvement, which can be addressed through further research. Moreover, the results of the experiments show that the idea that the maximal margin goal generates the optimal decision boundary is not necessarily true. Following this we can, in the future, perhaps start to consider the bias of an SVM as a hyper-parameter to be optimized after training.

Bibliography

[1] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25 in Automation and Remote Control,, pages 821–837, 1964.

URL: https://cs.uwaterloo.ca/~y328yu/classics/kernel.pdf.

[2] Saugata Basu, Dmitrii V. Pasechnik, and Marie-Françoise Roy. Computing the betti numbers of semi-algebraic sets defined by partly quadratic systems of polynomials. *Journal of Algebra*, 321(8):2206–2229, apr 2009. doi: 10.1016/j.jalgebra.2008.09.043.

URL: https://doi.org/10.1016%2Fj.jalgebra.2008.09.043.

[3] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130401.

URL: https://doi.org/10.1145/130385.130401.

[4] Hui Cao, Takashi Naito, and Yoshiki Ninomiya. Approximate rbf kernel svm and its applications in pedestrian classification. In *The 1st International Workshop on Machine Learning for Vision-based Motion Analysis-MLVMA'08*, 2008.

URL: https://inria.hal.science/inria-00325810/document.

[5] Chao Chen, Xiuyan Ni, Qinxun Bai, and Yusu Wang. A topological regularizer for classifiers via persistent homology. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 2573–2582. PMLR, 16–18 Apr 2019.

URL: https://proceedings.mlr.press/v89/chen19g.html.

[6] James R. Clough, Nicholas Byrne, Ilkay Oksuz, Veronika A. Zimmer, Julia A. Schnabel, and Andrew P. King. A topological loss function for deep-learning based image segmentation using persistent homology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):8766–8778, 2022. doi: 10.1109/TPAMI.2020.3013679.
URL: https://arxiv.org/abs/1910.01877.

[7] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995. doi: 10.1007/bf00994018.
URL: https://doi.org/10.1007/bf00994018.

[8] B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoï. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, 6: 793–800, 2006.
URL: https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=im&paperid=4937&option_lang=eng.

[9] Charles Duménil. *Expected Size of the 3-Dimensional Delaunay Triangulation of Random Points on a Surface*. Theses, Université de Lorraine, May 2022.
URL: https://theses.hal.science/tel-03695908.

[10] Herbert Edelsbrunner and John L Harer. *Computational topology: an introduction*. American Mathematical Society, 2022.
URL: https://www.maths.ed.ac.uk/~v1ranick/papers/edelcomp.pdf.

[11] Hamid Karimi, Tyler Derr, and Jiliang Tang. Characterizing the decision boundary of deep neural networks. *arXiv preprint arXiv:1912.11460*, 2019. doi: 10.48550/ARXIV.1912.11460.
URL: https://arxiv.org/abs/1912.11460.

[12] Andreas Knauf and Nikolay Martynchuk. Topology change of levels sets in morse theory, 2019.
URL: https://arxiv.org/abs/1910.05294.

[13] Marissa Masden. Algorithmic determination of the combinatorial structure of the linear regions of relu neural networks. *arXiv preprint arXiv:2207.07696*, 2022. doi: 10.48550/ARXIV.2207.07696.
URL: https://arxiv.org/abs/2207.07696.

[14] Marston Morse. Relations between the critical points of a real function of independent variables. *Transactions of the American Mathematical Society*, 27(3):345–396, 1925.

URL: https://www.ams.org/journals/tran/1925-027-03/S0002-9947-1925-1501318-X/S0002-9947-1925-1501318-X.pdf.

[15] Henri Poincare. gallica:12148/bpt6k4337198/f7. *Journal de l'École Polytechnique.*, 1895.
URL: https://gallica.bnf.fr/ark:/12148/bpt6k4337198/f7.

[16] Karthikeyan Natesan Ramamurthy, Kush Varshney, and Krishnan Mody. Topological data analysis of decision boundaries with application to model selection. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5351–5360. PMLR, 09–15 Jun 2019.
URL: https://proceedings.mlr.press/v97/ramamurthy19a.html.

[17] Matthias Ring and Bjoern M. Eskofier. An approximation of the gaussian rbf kernel for efficient classification with svms. *Pattern Recognition Letters*, 84:107–113, 2016. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2016.08.013.
URL: https://www.sciencedirect.com/science/article/pii/S016786551630215X.

[18] I. Steinwart, D. Hush, and C. Scovel. An explicit description of the reproducing kernel hilbert spaces of gaussian rbf kernels. *IEEE Transactions on Information Theory*, 52(10):4635–4643, 2006. doi: 10.1109/TIT.2006.881713.
URL: https://ieeexplore.ieee.org/abstract/document/1705021?casa_token=47-6OCp_XXYAAAAA:2eiUFRffIuQkL-xOTrLMWRGhsm-L6_bZTaPMMAwDs9MLAkFOyKzOq7MFEz4qwRBaa4DB2Dd8qQ.

[19] John Stillwell. *Papers on Topology*. 2009.
URL: https://www.maths.ed.ac.uk/~v1ranick/papers/poincare2009.pdf.

[20] Burt Totaro. Topology of singular algebraic varieties, 2003.
URL: https://arxiv.org/abs/math/0304296.

[21] Kush R. Varshney and Karthikeyan Natesan Ramamurthy. Persistent topology of decision boundaries. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3931–3935, 2015. doi: 10.1109/ICASSP.2015.7178708.
URL: https://krvarshney.github.io/pubs/VarshneyR_icassp2015.pdf.