NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PIECEWISE-AFFINE CLASSIFIERS IN SUPPORT VECTOR MACHINES

by

Matthew T. Miller

June 2019

Thesis Advisor:	Johannes O. Royset
Co-Advisor:	Arthur J. Krener
Second Reader:	Matthew Norton

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2019	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
PIECEWISE-AFFINE CLASSIFIERS IN SUPPORT VECTOR MACHINES

5. FUNDING NUMBERS

6. AUTHOR(S) Matthew T. Miller

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

The Support Vector Machine (SVM) model has been a topic of study for over twenty years, and novel approaches to the classification problem using SVM continue to be established. In this work, we develop a new, nonlinear version of SVM based on a piecewise-affine classifier. This class of classifiers constitutes a tractable class beyond the affine functions that enables approximation of nonparametric SVM in high dimensions. We solve the resulting Piecewise-Affine SVM (PA-SVM) model using the Difference-of-Convex Algorithm (DCA) and a stochastic gradient descent (SGD) algorithm. The PA-SVM model is nonconvex, and the algorithms generally only provide locally optimal solutions. Still, they provide for a robust, capable classifier. Results show that by using DCA, the PA-SVM model can significantly reduce training misclassifications relative to the common Affine SVM (A-SVM) model by as much as 92%. Additionally, we show that test set errors can be reduced by as much as 67% compared to A-SVM. We find that solutions are more affected by the number of pieces employed rather than by regularization penalties. These results come from applying the PA-SVM model to three real-world data sets whose total features range from 16 to 41 and whose total observations range from 194 to 1,553.

14. SUBJECT TERMS optimization, classification, piecewise-affine, difference of convex, difference of convex algorithm, support vector machine, stochastic gradient descent, nonconvex, nonlinear	**15. NUMBER OF PAGES** 73
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

PIECEWISE-AFFINE CLASSIFIERS IN SUPPORT VECTOR MACHINES

Matthew T. Miller
Major, United States Marine Corps
BS, Georgia Institute of Technology, 2009

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

and

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2019

Approved by: Johannes O. Royset
Advisor

Arthur J. Krener
Co-Advisor

Matthew Norton
Second Reader

W. Matthew Carlyle
Chair, Department of Operations Research

Wei Kang
Chair, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Support Vector Machine (SVM) model has been a topic of study for over twenty years, and novel approaches to the classification problem using SVM continue to be established. In this work, we develop a new, nonlinear version of SVM based on a piecewise-affine classifier. This class of classifiers constitutes a tractable class beyond the affine functions that enables approximation of nonparametric SVM in high dimensions. We solve the resulting Piecewise-Affine SVM (PA-SVM) model using the Difference-of-Convex Algorithm (DCA) and a stochastic gradient descent (SGD) algorithm. The PA-SVM model is nonconvex, and the algorithms generally only provide locally optimal solutions. Still, they provide for a robust, capable classifier. Results show that by using DCA, the PA-SVM model can significantly reduce training misclassifications relative to the common Affine SVM (A-SVM) model by as much as 92%. Additionally, we show that test set errors can be reduced by as much as 67% compared to A-SVM. We find that solutions are more affected by the number of pieces employed rather than by regularization penalties. These results come from applying the PA-SVM model to three real-world data sets whose total features range from 16 to 41 and whose total observations range from 194 to 1,553.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

A-SVM	Affine SVM
DC	Difference-of-Convex
DCA	Difference-of-Convex Algorithm
MTVR	Medium Tactical Vehicle Replacement
PA-SVM	Piecewise-Affine SVM
PA-SVM LP	PA-SVM Linear Program
QSAR	Quantitative Structure Activity Relationship
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
WPBC	Wisconsin Prognostic Breast Cancer

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

The Support Vector Machine (SVM) model is well documented for its successful application to numerous existing classification problems. Current nonlinear versions, however, rely on inefficient approaches such as using nonlinear mathematical programs, transforming data into higher dimensions, and requiring the user to select an appropriate kernel. In this work, we present a new nonlinear version of the SVM model based on a piecewise-affine classifier. This class of classifiers constitutes a tractable class beyond the affine functions that enables approximation of nonparametric SVM in high dimensions.

We solve the resulting nonconvex Piecewise-Affine SVM (PA-SVM) model using the Difference-of-Convex Algorithm (DCA) and a stochastic gradient descent (SGD) algorithm, and we apply the model to three real-world data sets whose total features range from 16 to 41 and whose total observations range from 194 to 1,553. As the model is nonconvex, the algorithms generally only provide locally optimal solutions. However, the PA-SVM model still provides a robust and capable classifier. On one data set in particular, while the Affine SVM (A-SVM) model results in a 16.2% misclassification rate, the PA-SVM model, using DCA, results in a 1.3% misclassification rate, a 92% improvement. On another set of data, the A-SVM model results in a 5.8% misclassification rate on the test set, but the PA-SVM model reduces that by 67% to only 1.9%, again using DCA.

While not the end goal, classification results on training sets using DCA suggests that the PA-SVM model is capable of classifying known data with almost 100% accuracy. This supports the theory behind piecewise-affine functions and their ability to approximate upper semi-continuous functions arbitrarily well as the number of pieces increase. SGD, while proven to be highly successful elsewhere, did not achieve similar results in this work. Results on training data were on par or worse than A-SVM. However, SGD did outperform DCA on one test set, reducing the A-SVM error rate by 36% (compared to only 27% using DCA). With so many variations of the SGD algorithm in existence though, it may be capable of producing more competitive results on our model than what we find in our work.

We discuss in detail and, through small experiments, explore the many parameters associated with the PA-SVM model and the two algorithms we use to solve it. On larger experiments,

we find that the two most significant parameters are the number of pieces one uses and the number of trials dedicated to any one experiment. We find that better solutions are found over time, so we recommend efficient modifications to the PA-SVM as future work. A surprising result from this work suggests that the best solutions do not always require the most pieces. Indeed, using fewer pieces appears to negate the common task of finding the optimal penalty values in order to ensure sufficient model generalization. Specifically, the best test result we find by using a total of nine pieces and thousands of penalty optimization trials can be found by using only five pieces without having to incorporate the penalty optimization process.

Due to its ability to reduce training and test set error, the PA-SVM model is a strong contender in addressing emerging classification problems. A successful application of nonconvex optimization techniques and algorithms, the PA-SVM model reduces classification errors compared to its affine version by only incorporating a few more affine pieces. Since the domain of these pieces does not have to be predetermined based on each unique set of data, we are optimistic in the model's application to data in higher dimensions than those we experiment on in this study. Although better solutions may still be out there, and despite room for future work, the PA-SVM model has proven its success in that it exhibits the ability to be a robust classifier.

CHAPTER 1:
Background

The concept of machine learning has been around for several decades; however, advances in computer technology and optimization theory have recently enabled the phrase to join the mainstream lexicon. This is because machine learning techniques, like classification, are now able to be applied to numerous practical problems. As we find new ways of storing information in quantitative form, and thus, new applications of machine learning, the need for these techniques is sure to increase. In this work, we focus on a subset of machine learning classification models, called Support Vector Machines (SVM), and we provide an alternative nonlinear version as a possible means of answering this need. We begin by introducing concepts on which the model is founded.

1.1 Supervised Learning

Suppose one has a set of data that takes the form of (x_i, y_i) pairs, where x_i is an n-dimensional data vector, y_i is a 1-dimensional vector of a certain value (either numeric or categorical) that describes the x_i data in some way, and $i = 1, \ldots, m$. Now suppose that one seeks to create some function, $f(x)$, whereby each x_i vector could be used as input, and the function's output would be the input vector's corresponding y_i value. This theoretical function, therefore, would create a perfect mathematical relationship between the data's x_i independent variables and the y_i dependent variable. Depending on the variability and nature of the data, however, creating a single function that produces the exact y_i value for each x_i input vector may be impossible. At this point, it may be sufficient for the function to produce output in the form of some predicted value, $\hat{f}(x_i)$, that may not correspond to the correct y_i associated with *every* x_i but would hopefully correspond to the correct y_i value for as many x_i as possible. While creating such a function would be valuable, it would *really* be valuable if the function were capable of producing accurate predictions for new, unknown data.

The scenario described above summarizes the objective of supervised learning (Hastie et al. 2017). It is called *supervised* because of the process by which the function is tuned to provide a solution. An algorithm, typically in the form of a computer program (Hastie et al.

2017), *learns* from a set of *training* data where all of the (x_i, y_i) information is known, and through some optimal process, explores the solution space and provides the function that gives the best prediction capability.

1.2 Classification

Classification is one of the two basic prediction tasks within supervised learning (Hastie et al. 2017), and it involves taking quantitative input data (the x_i) and producing a qualitative response ($\hat{f}(x_i)$), usually in the numeric form of some discrete or categorical variable (regression, the other prediction task, instead produces a quantitative response on a numeric scale) (Hastie et al. 2013). A classification model, therefore, seeks to create a function that, through the methods of supervised learning described above, produces as output a prediction of the class, or category, that each data entry is assigned. Knowing that there will be some error between the predicted value, $\hat{f}(x_i)$, and the actual value, the objective would be for the function to minimize the number of errors that it produces over the entire set of training data. Solving for this optimal function, however, could prove challenging depending on the form that the function takes, namely, whether or not the function is convex.

1.3 Convexity

In general, a function, $f(x)$, defined on the appropriate domain, S, in $\mathbb{R}^n$, is said to be convex on S if

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2) \tag{1.1}$$

holds for every $x_1, x_2 \in S$ and for every λ such that $0 \leq \lambda \leq 1$ (Rockafeller 1970). The geometric consequence is shown in Figure 1.1. Since these functions are convex, one can choose any x_1, x_2 pair and any λ value, and Equation 1.1 will be true. While a thorough discussion on the theory of convexity is beyond the scope of this work, and rules beyond Equation 1.1 exist for determining whether a function is convex or not, we briefly expand on the terms *convex* and *nonconvex* in order to understand their impact on minimization problems.

The functions in Figure 1.1 satisfy the condition of Equation 1.1 and are therefore convex. A minimization problem involving a convex objective function exhibits the important property that every locally optimal solution as well as every stationary point is actually a globally

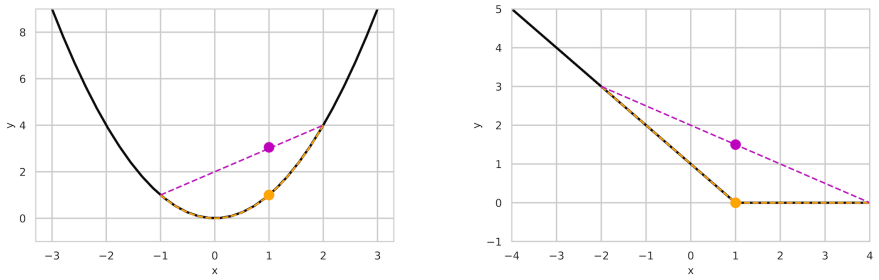

Figure 1.1. Examples of Convex Functions

These functions provide a graphical representation of Equation 1.1 being true. On the left is a graph of the function $f(x) = x^2$, where $x_1 = -1$, $x_2 = 2$, and $\lambda = \frac{1}{3}$. The figure on the right is the hinge-loss function (described later), where $x_1 = -2$, $x_2 = 4$, and $\lambda = \frac{1}{2}$. The dashed orange lines represent all possible values that the function can take at any linear combination of x_1 and x_2, which graphically depicts the left-hand side of Equation 1.1. The dashed purple lines show all possible values between $f(x_1)$ and $f(x_2)$, the right-hand side of the equation.

optimal solution (Beck 2014; Schölkopf and Smola 2002). Since standard optimization algorithms can often obtain a stationary point relatively easily, a globally optimal solution of a convex function is computationally tractable.

Functions that do not satisfy Equation 1.1 are nonconvex, and thus standard algorithms may fail to provide a globally optimal solution. This is caused by the large number of locally optimal solutions and stationary points that might not be globally optimal for a minimization problem involving a nonconvex function (see Figure 1.2). To overcome this difficulty, advanced algorithms for nonconvex problems need to introduce either some element of exploration through randomization and/or partition of the space of possible solutions.

1.4 Support Vector Machines

It is clear from the above discussion that when minimizing over an objective function, form matters. Luckily, classification models with convex objective functions, like SVM, exist. The SVM model is one of the many, well documented (Hastie et al. 2017, 2013; Campbell and Ying 2011), classification models in use today. Capable of being tuned for sensitivity and feature selection, SVM is a robust model that produces both an intuitive and highly interpretable classifier. Before describing how the SVM model works, however, we must

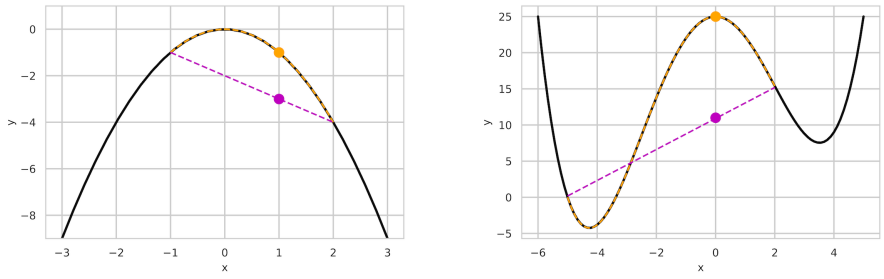

Figure 1.2. Two Nonconvex Functions

These functions provide a graphical representation of Equation 1.1 being false. The two lines represent the same information as in Figure 1.1. The left-hand graph is $f(x) = -x^2$, and while only a small subset of the function's range is shown, there is no minimum to the function. The right-hand graph consists of two local minima across the function's domain, only one of which is the global minimum.

first briefly describe a hyperplane, the geometric subspace that SVM leverages to produce its classifier.

1.4.1 Defining a Hyperplane

According to Hastie et al. (2013), a hyperplane in an n-dimensional space is a flat affine subspace that lies in dimension $n - 1$. Affine simply refers to the fact that the subspace does not necessarily pass through the origin. A hyperplane in a two-dimensional space, therefore, is a line, and a hyperplane in three dimensions is a plane. Most would agree that trying to visualize a hyperplane in dimensions greater than three is a very challenging mental exercise. We can define a hyperplane in an n-dimensional space with the following equation:

$$f(x) = a_1x_1 + a_2x_2 + \ldots + a_nx_n + \alpha = 0. \tag{1.2}$$

The equation above, despite its simplicity, is indeed the function that the SVM model uses to produce classification predictions, where the learning algorithm's decision variables are the hyerplane's coefficients $a_1, a_2, ..., a_n$ and the origin offset, α. While this looks like the same function used in linear regression, the methodology between that technique and SVM are quite different. Linear regression (which can easily be transformed into logistic regression for classification purposes) seeks to find a hyperplane that minimizes the distance from each observed data point to that hyperplane, whereas the SVM model seeks to find a hyperplane

that actually separates all of the observations in one y_i class from all of the observations in another. While SVM models exist that can accommodate more than two classes, we focus here on the binary case, where $y_i \in \{-1, 1\}$.

1.4.2 The Separating Hyperplane

If the SVM learning algorithm finds a hyperplane whereby every training observation, x_i, can be perfectly separated according to its y_i class label, then such a hyperplane is called a separating hyperplane (Campbell and Ying 2011), and the model's prediction error is minimized to zero. By observing the set of possible y_i values and Equation 1.2, we see a natural classification rule emerge, where

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + \alpha < 0 \quad \text{if } y_i = -1 \tag{1.3}$$

and

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + \alpha > 0 \quad \text{if } y_i = 1 \tag{1.4}$$

Therefore, we can classify a new test observation, x^*, based on the sign from $f(x^*) = a_1 x_1^* + a_2 x_2^* + \cdots + a_n x_n^* + \alpha$. If $f(x^*) < 0$, then we assign x^* to the class -1 (which is equivalent to setting $y^* = -1$), and if $f(x^*) > 0$, to the class 1, since the sign of $f(x^*)$ corresponds to which side of the hyperplane the observation lies.

When a separating hyperplane can be found, then an infinite number of separating hyperplanes exist (Hastie et al. 2013). This is because the separating hyperplane can be shifted up or down by a small (possibly infinitely small) amount without changing which side any observation resides. In this scenario, the goal of the SVM model shifts from simply separating the data to choosing the separating hyperplane that provides the largest margin on either side before coming in contact with any observation, thus reducing bias and allowing for better test classification. Figure 1.3 shows an example of such a case.

1.4.3 Support Vector Classifiers

Completely separable data, as in the above case, is unfortunately not representative of reality. Real-world data is inherently noisy and littered with variability and therefore not capable of being perfectly separated into its distinct labels. This leads to the nuance of the SVM model, as well as every other predictive model, which is to develop the most

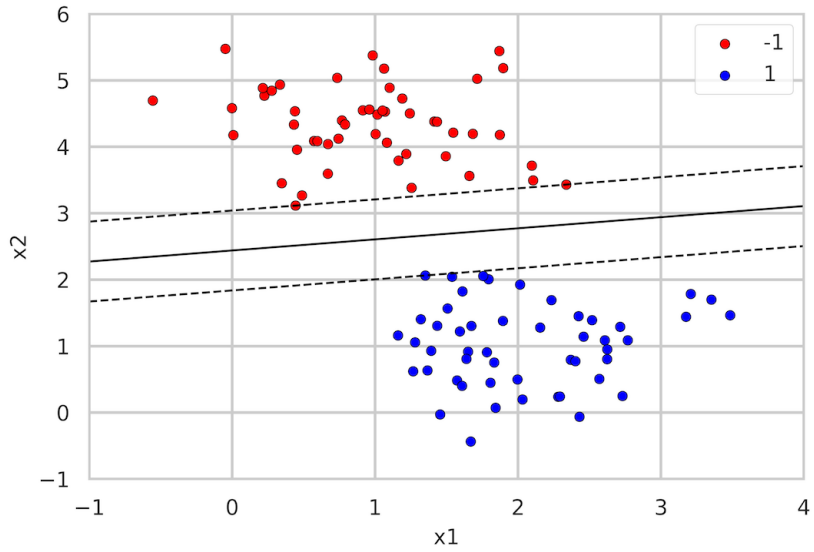

Figure 1.3. Maximum Margin Classifier

Synthetic data from the makers of scikit-learn (Pedregosa et al. 2011). The solid line represents the separating hyperplane, and the dashed lines represent the maximum margins. Line code comes from (VanderPlas 2016).

accurate classifiers as possible using real world data. The SVM model can still be used, and its objective stays the same (to classify data using a hyperplane), but the interpretation of the margins changes. Now, instead of maximizing the distance from the hyperplane to the nearest data point in each of the two classes, and as a result, forcing every observation not only to be on the correct side of the hyperplane, but also the correct side of the margin, the SVM model allows observations to be inside the margin and even on the wrong side of the hyperplane (which is inevitable in many cases). This model, when data cannot be completely separated using a single hyperplane, is specifically called the support vector classifier. Figure 1.4 shows an example.

In support vector classification, the margins are referred to as "soft" because observations are now allowed to enter between them, and the size of the margin translates into how sensitive the classifier will be to observations close to the hyperplane. A larger margin will result in a more generalized solution that is not highly influenced by outliers, but, depending on how large the margin is, the model will likely have a greater number of training observations on the wrong side of the margin, or even on the wrong side of the

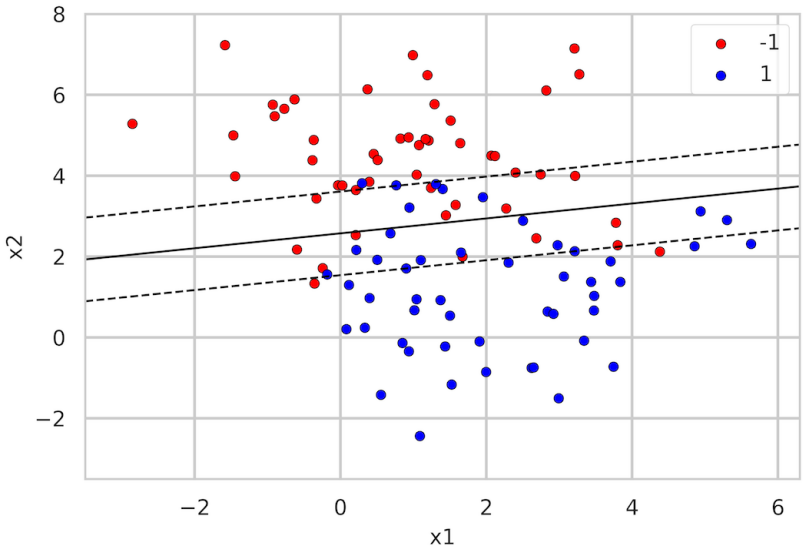

Figure 1.4. Support Vector Classifier

The solid line represents the classifier's optimal hyperplane and the dashed lines represent the margins. The synthetic data and visualization techniques come from the same sources as in Figure 1.3.

hyperplane. In fact, all observations could be on the wrong side of the hyperplane if the margin is large enough. A smaller margin, on the other hand, will result in better training classification results, but will make the model more influenced by outliers the smaller the margin gets. In general, one would like a classification model that is not sensitive to outliers, but also produces quality classification results. While literature (Hastie et al. 2017, 2013; Campbell and Ying 2011; Kecman 2005) discusses the use of a tuning parameter, C, to determine the optimal size of the margin, we maintain a constant value of one, which can be observed in Figure 1.4. Whether the size of the margin remains constant or is tuned, the goal of support vector classification is to define the hyperplane that minimizes the average distance that every observation inside the margin and on the wrong side of the hyperplane has to the correct margin. We capture this loss by using the hinge loss function (Hastie et al. 2013).

1.4.4 Hinge Loss

In general, the SVM model seeks to minimize the number of classification errors that the function, produced by the learning algorithm, makes. By defining

$$\psi_i = y_i(a_1 x_{i1} + a_2 x_{i2} + \ldots + a_n x_{in} + \alpha) \tag{1.5}$$

and observing the fact that a correct classification takes on the following property

$$\psi_i > 0, \tag{1.6}$$

we can define an error function by

$$e(\psi_i) = \begin{cases} 1 & \text{if } \psi_i \leq 0, \text{and} \\ 0 & \text{otherwise} \end{cases} \tag{1.7}$$

Therefore, the general classification model seeks to minimize

$$\sum_{i=1}^{m} e(\psi_i) \tag{1.8}$$

However, Equation 1.8 is nonconvex, since it is a discontinuous step function. Instead, a continuous, convex approximation to Equation 1.8, called the hinge loss, defined by

$$\max(0, 1 - \psi_i), \tag{1.9}$$

replaces Equation 1.8. Figure 1.5 provides a visual representation of Equation 1.9.

The value of 1 in Equation 1.9 represents the SVM margin, and the overall hinge loss function can be interpreted through the observation of three different cases. The first case is when $\psi_i \geq 1$, and this means that the classifier not only places an observation on the correct side of the hyperplane, it also places it on the correct side of the margin. In this case, there is no loss. The second case, when $0 \leq \psi_i < 1$, happens when the classifier places an observation on the wrong side of the margin but on the correct side of the hyperplane. Lastly, when $\psi_i < 0$, the classifier places the observation on the wrong side of the margin as well as the wrong side of the hyperplane. In cases two and three, the loss linearly increases

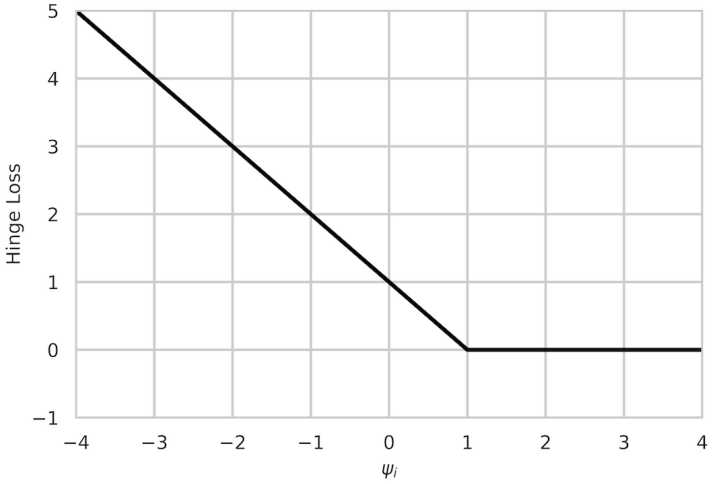

Figure 1.5. Hinge Loss Function

(Figure 1.5). Therefore, the SVM model's goal in the non-separable case is to minimize the average hinge loss for cases two and three as applied to the entire set of training data. We define this model by

$$\min \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - \psi_i) \tag{1.10}$$

An important observation regarding the hinge loss function is that it provides an upper bound on the average number of classification errors that the SVM model gives. This is for two reasons: first, a correct classification can give a non-zero loss value (case two, above). Secondly, incorrect classifications can return a loss value greater than 1. In Equation 1.7, the loss values were at most 1, but in Equation 1.9, the loss values are not so limited.

1.4.5 Regularization

A danger one needs to keep in mind when creating a classifier is the notion of over-fitting during the training phase. This refers to making a classifier that is so specific and tailored to the training data, that, while the classifier performs well on the data it knows, once the classifier is applied to an unknown set of data (the test set), it performs terribly. A way to avoid over-fitting, in the SVM case, is to implement a penalty for any non-zero solution that the learning algorithm applies to a coefficient decision variable, a_i. This method forces the learning algorithm to explore the trade space between a low hinge loss value for correctly

classifying certain observations and the penalty associated with doing so by giving non-zero values to whatever decision variables necessary in order to get the low hinge loss value. In this manner, if the benefit of achieving a low hinge loss value is worth the penalty, then coefficients will be given non-zero solutions. On the other hand, coefficients not providing enough benefit will remain zero. Like the tuning parameter, C, the penalty value is capable of being tuned by the user, and the method helps the SVM model to not be influenced by outliers in the training data. While the classification error associated with the training data may be higher with the penalty, the intent is to achieve better classification results on the test set than would be without the penalty. A penalty commonly associated with the SVM model (Hastie et al. 2013) is called the L1 norm, defined by

$$\|a\|_1 = \sum_{i=1}^{n} |a_i| \tag{1.11}$$

While this helps to avoid issues with overfitting, a secondary interpretation of applying a regularization penalty is being able to distinguish between those features that actually have an effect on the dependent variable, and those that do not. This is referred to as variable selection or feature selection (Hastie et al. 2013), and it is of significant value when n is large. If a set of data has, say, 1,000 features, but only a handful actually influence the dependent variable, then forcing a learning algorithm to explore the solution space over all 1,000 dimensions is extremely inefficient. Regularization methods can improve the computational efficiency of our learning algorithms by reducing the number of dimensions in the problem. Aside from computational efficiency, this also informs the user as to the practical interpretation of the relationship between the data's independent and dependent variables.

1.4.6 SVM Linear Program

Equation 1.10 needs transforming before being implemented in a mathematical program. By introducing an auxiliary variable, z_i, to represent the hinge loss function, and ρ as the regularization parameter, we are able to formulate the SVM model into a convex linear program defined by

$$
\begin{aligned}
\min_{a,\alpha,z} \frac{1}{m}\sum_{i=1}^{m} z_i + \rho \sum_{j=1}^{n}(u_j + v_j) & \\
\text{s.t. } 1 - y_i(a_1 x_{i1} + a_2 x_{i2} + \ldots + a_n x_{in} + \alpha) \leq z_i & \quad \forall i \in \{1, \ldots, m\} \\
0 \leq z_i & \quad \forall i \in \{1, \ldots, m\} \\
a_j \leq u_j & \quad \forall j \in \{1, \ldots, n\} \\
-a_j \leq v_j & \quad \forall j \in \{1, \ldots, n\}
\end{aligned}
\tag{1.12}
$$

1.4.7 Alternative SVM Algorithms

We present the above linear program as a means of solving the SVM model; however, many alternative methods exist that are also capable of solving the model. Various algorithms, such as Pegasos (Shalev-Shwartz et al. 2011), use an iterative process of computing the subgradient of an approximation to Equation 1.10 in each step to reach the optimal solution. Additional subgradient-type methods are further discussed by Nam et al. (2013). Other algorithms take a dual approach to solving the SVM model. Decomposition methods such as Sequential Minimal Optimization (Platt 1999) and SVM-Light (Joachims 1999) break down the overall problem into smaller sub-problems and leverage properties of the Lagrange multiplier to converge to the solution. Related to the decomposition method, Tseng and Yun (2010) use a coordinate gradient descent algorithm to solve the SVM model. Other algorithms exist, but it is clear that the linear program in Equation 1.12 is only one of many possible ways to solve the SVM model.

1.4.8 Nonlinear SVM

The SVM model discussed so far is referred to as the linear, or affine, SVM model, since a single hyperplane is used to classify data. However, this limits the model from being able to classify nonlinear patterns, such as a curve or a bend, through the data's feature space. Because of this, various smooth kernal methods (Hastie et al. 2013) have been incorporated into the SVM model in order to classify nonlinear data, and it is this nonlinear capability that distinguishes true support vector machines from linear support vector classifiers. While these kernel methods strengthen the SVM model, they simultaneously translate into nonlinear optimization problems and must transform the data into a higher dimension. An

alternative approach to the nonlinear SVM model is to use a piecewise-affine objective function.

1.5 Piecewise-Affine Functions

When a function's domain is divided into intervals, and the function's range is then uniquely defined on each interval using sub-functions, the result is called a piecewise function. The nature of a piecewise function is characterized by the form that these sub-functions take on each interval, and when every sub-function is affine, the overall function is piecewise-affine. We can define an affine function to be a function, $\gamma : \mathbb{R}^n \rightarrow \mathbb{R}$, that takes the form

$$\gamma(x) = a^T x + \alpha \tag{1.13}$$

where $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ (Adeeb and Troitsky 2016). It has been shown (Gorokhovik et al. 1994; Royset 2017) that a piecewise affine function can be constructed in the following way

$$f(x) = \max\{\gamma_k(x)\} - \max\{\gamma_l(x)\} \tag{1.14}$$

where γ_k and γ_l are sets of affine functions. A nice property of Equation 1.14 is that it is mesh free on the domain of x.

1.5.1 Mesh-Free Defined

In the context of Equation 1.14, mesh-free means that each affine component's domain is adaptable and not pre-selected (Royset 2017). Mesh-free functions, therefore, do not require the user to pre-determine component parameters, which, for high-dimension optimization problems, is a nice property. Not only does this reduce the up-front work required from the user, it also reduces the problem in terms of computational complexity. For each dimension n, the number of required parameters for each mesh increases exponentially (2^n), which makes optimization problems involving a mesh computationally inefficient. Additionally, approximations involving a mesh may be sub-optimal, since the user must define the endpoints. The mesh-free function in Equation 1.14 has two key advantages: it grows at a linear rate based on the number of pieces, thus making it computationally attractive and preferred for high-dimension problems, and the domain for each affine sub-function is optimally chosen by the solver. Piecewise-polynomial functions are also mesh

free, but their computational complexity grows at a faster rate compared to piecewise-affine functions.

1.6 Motivation and Overview

The SVM model has been the topic of study for over twenty years (Boser et al. 1992; Cortes and Vapnik 1995), and novel approaches to classification using SVM continue to be developed, with recent work combining the Difference-of-Convex Algorithm (DCA), which we discuss in more detail in Chapter 2, and piecewise methods to solve various aspects of the problem. Neumann et al. (2005) and Le Thi et al. (2008) provide methods of SVM feature selection by using DCA to minimize over a nonconvex regularization term, and linear, domain-localized approaches to nonlinear SVM have also been developed. Profile SVM (Cheng et al. 2010) and a piecewise-linear SVM model (Ye et al. 2013) use clustering techniques and a maximum likelihood estimate, respectively, to create a separate, linear SVM model for each of their model's subspaces. While individual DCA and piecewise-linear approaches to SVM, like those mentioned above, have been documented, we believe that these approaches can actually be combined in a more general sense to develop a robust, novel approach to nonlinear SVM.

In this work, we utilize the mesh-free properties of piecewise-affine functions to develop a piecewise-affine classifier based on the Affine SVM (A-SVM) model. The resulting Piecewise-Affine SVM (PA-SVM) model is nonconvex, so we use two algorithms, DCA and Stochastic Gradient Descent (SGD), to solve, and we then present metrics, such as hinge loss and computational run-time, that these two algorithms provide and require, respectively. We examine hinge loss on training data and classification error on test data, determine the level of generalization the model can provide, and then compare results with A-SVM. In Chapter 2, we provide a formulation of the PA-SVM classifier, a visualization of its effects on low dimensional data, and a detailed methodology for how it will be applied to larger data sets. In Chapter 3, three real-world data sets are introduced, training errors on these data sets are presented, and the model's inner workings, based on the model's performance on this data, are discussed. Chapter 4 shows the test errors associated with Chapter 3's results and explores the effect of changing the model's regularization parameters on a subset of these test scenarios. Lastly, Chapter 5 provides conclusions on the model.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Methodology

This chapter provides a formulation of the PA-SVM model and a visual demonstration of the model on low-dimension, synthetic data. The chapter also discusses two unique algorithms that are used to solve the model, and it introduces a methodology for how the model is applied to real-world data in Chapter 3.

2.1 Piecewise-Affine SVM

We seek to combine the A-SVM model with Equations 1.13 and 1.14 to create the nonlinear PA-SVM model. The initial function looks like

$$f(x) = \max_{k=1,\dots,q} \{a_k(x) + \alpha_k\} - \max_{l=1,\dots,r} \{b_l(x) + \beta_l\} \tag{2.1}$$

with q and r being the number of pieces in each set of affine functions. In this manner, we create a function that uses $\{q,r\}$ affine pieces as classifiers, not just a single affine hyperplane as in A-SVM. The unknown space over which we optimize can be considered to be in the general class of upper semi-continuous functions, so using a piecewise affine function for the purpose of classification is a natural choice since it can approximate every upper semi-continuous function arbitrarily well as the number of pieces increase (Royset 2017). A one-dimensional visualization of what we seek to accomplish is depicted in Figure 2.1 and two-dimensional representations are depicted in Figures 2.2 and 2.3.

The form of Equation 2.1 makes $f(x)$ nonconvex for $r > 1$, and it thus requires the use of nonconvex algorithms to solve. We present two such methods below.

2.2 Difference-of-Convex Functions

The first algorithm we discuss that assists in solving nonconvex problems leverages the structure of a certain function class: the Difference-of-Convex (DC) function. Such a function is defined by

$$f = g - h \tag{2.2}$$

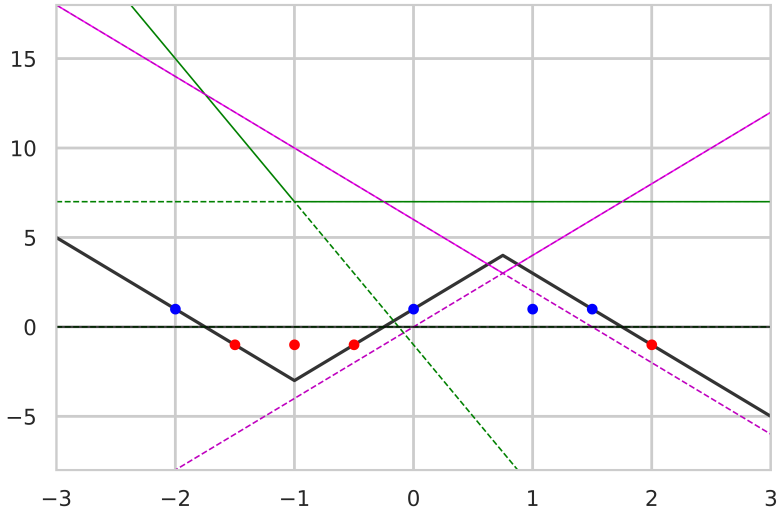

Figure 2.1. PA-SVM Visualization on One-Dimensional Synthetic Data

The first set of affine functions in Equation 2.1 are represented by the green lines and the second set by the purple lines. The line is solid when it is the maximizing affine function within the set, and the line is dashed when it is not. The geometry of the classifier, therefore, is based on the solid, maximizing, lines. The solid black line represents the output of Equation 2.1. Plotting code was provided by a fellow student working on similar material (Samudio 2019).

where g and h are convex functions (Hartman 1959). While both g and h are themselves convex, Equation 2.2 is nonconvex, since $-h$ is concave (for a visualization of this rule, refer to Figures 1.1 and 1.2 and notice what happens to $f(x) = x^2$ when it is multiplied by -1). DC functions are well documented, have been developed and used in optimization problems over the last thirty years, and have been shown to be an effective problem-structure for solving large-scale nonconvex problems (Pham Dinh and Le Thi 2018). The DCA (Pham Dinh and Le Thi 1998) is an iterative process that minimizes over f in Equation 2.2 by solving many convex sub-problems by transforming $-h$ into a convex approximation during each step, and then leverages properties of duality to ensure local optimality conditions are met (Pham Dinh and Le Thi 1998; Horst and Thoai 1999; Pham Dinh and Le Thi 2005). Convergence to a local minimum in a finite number of iterations has been shown, but the study of DCA convergence is still ongoing (Pham Dinh and Le Thi 2018).

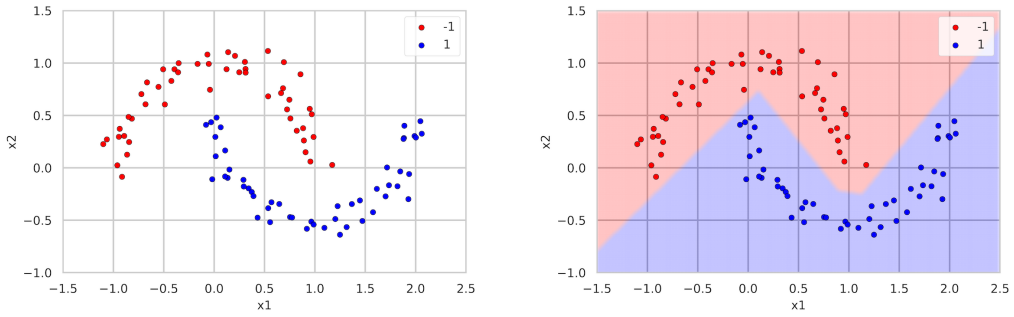

Figure 2.2. PA-SVM Visualization on the Make Moons Function

This nonlinear data would clearly not be separable with the A-SVM model. However, using an appropriate combination of piecewise-affine functions in Equation 2.1, the data becomes separable. The synthetic data comes from the makers of scikit-learn (Pedregosa et al. 2011).

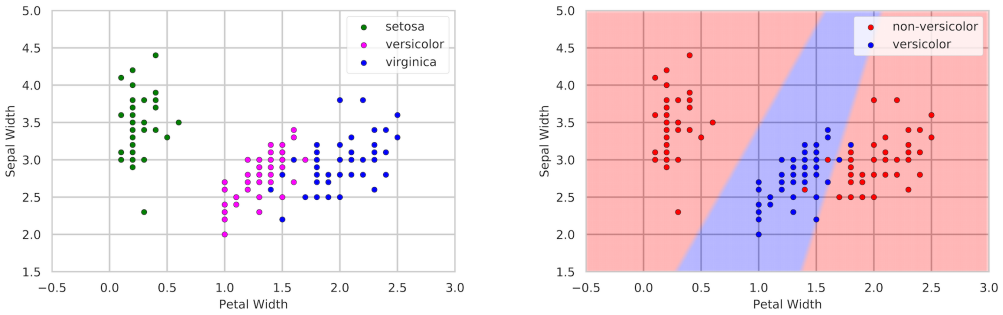

Figure 2.3. PA-SVM Visualization on Iris Data

The data used in this example comes from a popular, real-world source (Fisher 1936). Since the PA-SVM model classifies binary data, we split the three-class data into one of the two toughest classes for a binary, linear classifier to have to deal with: Versicolor and Non-versicolor. Using the PA-SVM algorithm, 96% of the data is accurately classified.

2.2.1 DCA on the PA-SVM Model

In practice, one must choose how to decompose f into a DC function before even implementing DCA, but our piecewise-affine minimization problem (Equation 2.1), already exists as a DC function, so we do not have to worry about that step (in our case, $g = \max_{k=1,\ldots,q} \{a_k(x)+\alpha_k\}$ and $h = \max_{l=1,\ldots,r} \{b_l(x) + \beta_l\}$). Like in the A-SVM case, we ultimately want to minimize the average number of classification errors that Equation 2.1 makes over a set of training data, x_i, $i = 1, \ldots, m$, and from Chapter 1, we know that we need to formulate this error by incorporating the hinge loss function. The difference in this case, however, is that we

must separate the problem based on the training data's class label, since the negative sign in Equation 2.1 will have different implications for each class. To begin, we assign those observations that are in the "1" class to the set M^+ and those observations in the "-1" class to the set M^-. After this step, the minimization problem looks like

$$\begin{aligned}\min_{\substack{a,\alpha,\\ b,\beta}} \quad & \frac{1}{M^+}\sum_{i\in M^+}\max(0, 1-\max_{k=1,\ldots,q}\{a_k(x)+\alpha_k\}+\max_{l=1,\ldots,r}\{b_l(x)+\beta_l\}) \\ & +\frac{1}{M^-}\sum_{i\in M^-}\max(0, 1+\max_{k=1,\ldots,q}\{a_k(x)+\alpha_k\}-\max_{l=1,\ldots,r}\{b_l(x)+\beta_l\})\end{aligned} \tag{2.3}$$

The next step in the DCA process is to approximate the nonconvex portion of the objective function into a convex form. The nonconvex parts in Equation 2.3 come from the $-\max$ terms, so those need to be transformed. As we see in Figure 2.1, the affine function that has an effect on the overall problem is the maximizing function, therefore, we define

$$\begin{aligned} k_i^* &\in \operatorname*{argmax}_{k=1,\ldots,q}\{a_k(x_i)+\alpha_k\} \quad \forall i \in M^+ \\ l_i^* &\in \operatorname*{argmax}_{l=1,\ldots,r}\{b_l(x_i)+\beta_l\} \quad \forall i \in M^- \end{aligned} \tag{2.4}$$

and limit Equation 2.3 to only optimize over the maximizing function for each x_i. By combining Equations 2.3 and 2.4, the convex approximation takes the form

$$\begin{aligned}\min_{\substack{a,\alpha,\\ b,\beta}} \quad & \frac{1}{M^+}\sum_{i\in M^+}\max(0, 1-a_{k_i^*}(x)-\alpha_{k_i^*}+\max_{l=1,\ldots,r}\{b_l(x)+\beta_l\}) \\ & +\frac{1}{M^-}\sum_{i\in M^-}\max(0, 1+\max_{k=1,\ldots,q}\{a_k(x)+\alpha_k\}-b_{l_i^*}(x)-\beta_{l_i^*}).\end{aligned} \tag{2.5}$$

In this manner, we restrict the problem, but in doing so, we formulate a convex approximation that is now capable of being implemented and solved for in a mathematical program.

2.2.2 The PA-SVM Linear Program

By using auxiliary variables, as we did in the A-SVM case in Chapter 1, to represent the hinge loss and penalty values, we can transform Equation 2.5 into the PA-SVM Linear Program (PA-SVM LP), defined below.

$$\min_{\substack{a,\alpha,\\ b,\beta,\\ z}} \frac{1}{m}\sum_{i=1}^{m} z_i + \rho\sum_{j=1}^{n}(\sum_{k=1}^{q}(u_{kj}+v_{kj}) + \sum_{l=1}^{r}(s_{lj}+t_{lj})) + \lambda(\sum_{k=1}^{q}(w_k+p_k) + \sum_{l=1}^{r}(d_l+n_l))$$

$$\begin{aligned}
\text{s.t. } 1 - a_{k_i^*}(x) - \alpha_{k_i^*} + b_l(x) + \beta_l \le z_i \quad & \forall i \in M^+,\ l = 1,\dots,r \\
1 + a_k(x) + \alpha_k - b_{l_i^*}(x) - \beta_{l_i^*} \le z_i \quad & \forall i \in M^-,\ k = 1,\dots,q \\
0 \le u_{kj}, v_{kj}, s_{lj}, t_{lj} \quad & \forall k,l,j \\
a_{kj} \le u_{kj}, -a_{kj} \le v_{kj}, b_{lj} \le s_{lj}, -b_{lj} \le t_{lj} \quad & \forall k,l,j \\
0 \le w_k, p_k, d_l, n_l \quad & \forall k,l \\
\alpha_k \le w_k, -\alpha_k \le p_k, \beta_l \le d_l, -\beta_l \le n_l \quad & \forall k,l
\end{aligned} \tag{2.6}$$

There are three new elements to this linear program that should be explained. A new penalty term, λ, has been added to the model to penalize the size of the origin offset variables, α_k and β_l. While this is not necessary in the A-SVM case, it is required for the PA-SVM model. Secondly, the max terms in Equation 2.5 are replaced by a constraint for each of the q, r pieces, thus demonstrating the linear growth in the overall model. Finally, the maximizing pieces, k_i^* and l_i^*, must be arbitrarily chosen and fed into the linear program, and we do this by initiating the decision variables at some starting location prior to running the model. Once we determine these pieces, we are ready for the final step in DCA, which is the iterative process of solving the PA-SVM LP (Equation 2.6) until a solution converges to a local minimum.

2.2.3 Convexification

Suppose we run the model and obtain a solution. One must be curious if this is indeed a good solution, since we not only arbitrarily chose the maximizing pieces beforehand, but we also kept them constant throughout the optimization process. The natural concern is that the maximizing pieces have changed, and if they have, then our entire model could actually be sub-optimal, since it is indeed the maximizing pieces that influence the outcome. This concern turns out to be valid, and the solution needs to be verified.

We do this verification by testing the solution back on the training data to obtain a *convexified* objective function and then compare this value to the actual objective function obtained

through the linear program. If the difference between the two is small, then we keep the solution. If a large difference exists, then we instantiate another linear program, but this time, we initialize the decision variables at the current solution. Since these variables are already at a *relatively* optimal solution, only minor tweaks should take place, and the goal is for the maximizing pieces that are defined prior to solving the linear program to still be the maximizing pieces after solving the linear program, thus the solution obtained through the linear program will exhibit the same behavior back on the training data.

We continue this verification process until the convexification error, the difference in objective functions, is small enough (so small that the error is assumed to only be due to computational numerical error), or until an iteration limit is met. These acceptable threshold values, some ϵ representing the convexification error along with the number of iterations we allow in order for ϵ to be reached, become the stopping conditions for DCA. Figure 2.4 shows the behavior of the DCA iterative process on 300 separate PA-SVM linear programs. In this case, $|\epsilon| = 1 \cdot 10^{-9}$ and the iteration limit is set to 15. The difference values are negative because of the structure of the comparison: (*convexified* objective function)-(actual objective function). The *convexified* objective function, a relaxation, will always be smaller than the actual objective function, a restriction. As can be seen in the figure, many trials reach the ϵ threshold value before hitting the 15 iteration limit.

2.2.4 The PA-SVM DCA

The DCA approach to solving the PA-SVM model requires parameters to be defined prior to running the linear program. These include the number of $\{q, r\}$ pieces and the regularization values (ρ,λ). Additionally, the decision variables need to be initiated at some starting location, and the stopping (convexification) conditions need to be defined in terms of an ϵ error and an iteration limit. The below pseudocode describes the DCA process on our problem.

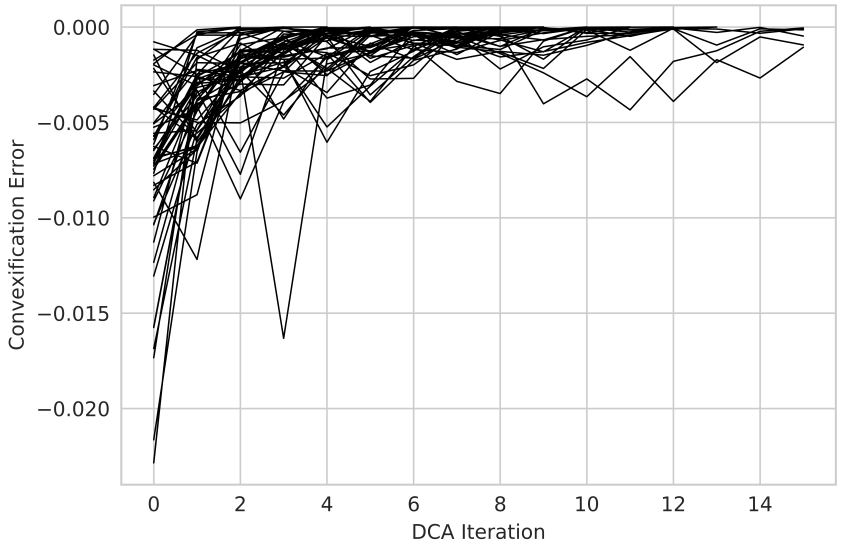

Figure 2.4. Convexification Error on MTVR Data Set

The tendency of DCA to converge to a local minimum through several iterations.

Algorithm 1: PA-SVM DCA

Results: Model solution and hinge loss

Initialize *Parameters*, *Variables*, *Counter*, Not Optimal = *True*;

Run PA-SVM LP;

while *Not Optimal* **do**

    if *convexification conditions satisfied* **then**

        Record current/best solution, hinge loss;

        Not Optimal = *False*;

    else

        Initialize *Variables* = current solution;

        Run PA-SVM LP;

        Counter+=1;

    end

end

2.3 Stochastic Gradient Descent

The second algorithm that we discuss is SGD (Robbins and Monro 1951). A standard gradient descent algorithm is an iterative process that takes the form

$$x_{k+1} = x_k - t_k \nabla f(x_k) \tag{2.7}$$

where x_k is the parameter value (or vector of values) at iteration k, t_k is the learning rate, or step-size, at iteration k, and $\nabla f(x_k)$ is the gradient, or first-order partial derivative of the objective function's parameter (or vector of parameters) x at iteration k. The idea is that after enough iterations, the parameter values converge to either a global minimum (in the convex case) or a local minimum (in the nonconvex case). Being able to solve an optimization problem using gradient descent methods is attractive, since the algorithm does not necessarily need the objective function to be in a convex form in order for it to converge to a solution. All notions of convexity (from an algorithmic point of view) are discarded. Thus, for nonconvex problems such as PA-SVM, the objective function does not need to be transformed into an approximate convex version pre-solve, like DCA above. Though challenges exist in determining the appropriate step-size and possibly the gradient, Equation 2.7 has yet another benefit in that it is a simple algorithm to implement (via code) that does not require mathematical programming solvers. A drawback, however, to using standard gradient descent algorithms is that they utilize the entire set of data at each iteration step, thus making the algorithm computationally taxing on data with a large number of observations. SGD, on the other hand, only takes a single random sample, or a small batch of samples, from the data at each iteration step. While the algorithm may take many more iterations to converge to a solution, each iteration only requires a fraction of the computational work compared to the standard method, thus providing a much more efficient algorithm. Because of this, SGD has been and continues to be used in the "big-data" fields of machine learning and neural networks, and unique versions of SGD continue to be developed (Newton et al. 2018; Huang and Toyoizumi 2017).

2.3.1 SGD on the PA-SVM Model

There are several ways to tune the SGD algorithm, from simply varying batch sizes and step sizes, to making more complex enhancements by incorporating adaptive methods, such as Adam (Kingma and Lei Ba 2015) and Ada-grad (Duchi et al. 2011), that use

information discovered in previous iterations to scale the current iteration's gradient. While these methods have proven effective in some scenarios, we do not find it successful for our problem. Our results are similar to those of others (Keskar and Socher 2017; Wilson et al. 2017). Pre-trial experiments show that scaling the gradient by a constant step size (which is data dependent) produces the best results, and the final algorithm mimics a tuned version of Rosenblatt's perceptron learning algorithm as described by Hastie et al. (2017). Additionally, we compute the gradient during each iteration by randomly sampling one data point. Figure 2.5 shows the behavior of SGD on our problem.

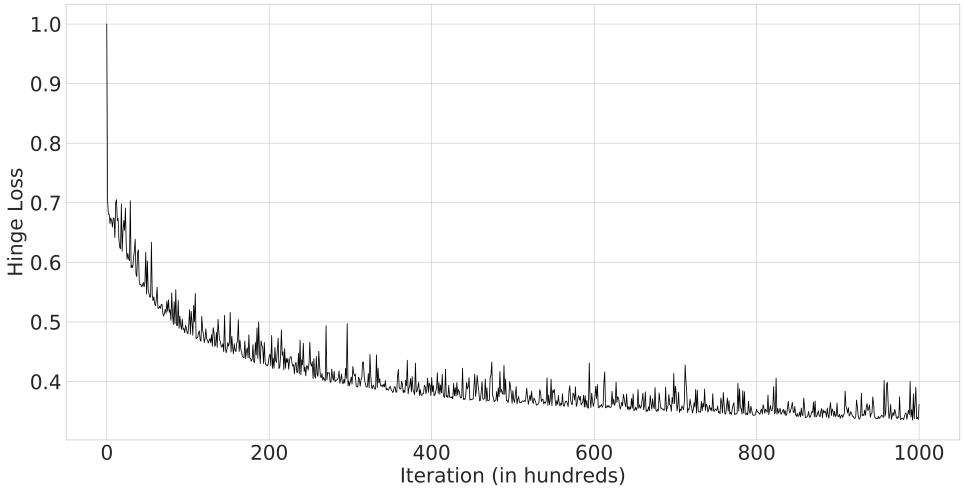

Figure 2.5. SGD Applied to QSAR Data

SGD applied to the QSAR data set where $q = 3$, $r = 3$, and t_k is held constant at 0.03.

2.4 Model Parameters

The above nonconvex algorithms require some initial exploration in order to determine their appropriate parameter settings before running full experiments. Independent of algorithm, the following parameters need to be defined: regularization terms, the number of pieces to use, variable initialization method, and total number of trials. Additionally, parameters specific to each algorithm need to be explored and determined in order to ensure convergence to a local optimum solution. In the DCA approach, the convexification threshold values used above need to be verified, and in the SGD case, the step size and total number of iterations

for each set of data need to be determined. Since these parameters need to be explored, the beginning of Chapter 3 is devoted to introducing data sets and providing these parameter values along with the reasoning that accompanies them. Afterwards, full experiments on the data are implemented.

CHAPTER 3:
Results on Training Data

In this chapter, we determine appropriate PA-SVM model parameters and explore the model's ability to reduce the hinge loss on three sets of data. We set convexification threshold limits and evaluate those limits on each data set. We confirm that the hinge loss approximates the misclassification rate by applying training solutions back onto the training data to determine the number of classification errors. Finally, PA-SVM results are compared to results from the A-SVM model.

3.1 Data Sets

In the previous chapter, we demonstrate the PA-SVM model on low-dimensional data in order to gain an intuition on how the model works and what we aim to do. In this chapter, the model is applied to the three sets of higher-dimension data described below.

3.1.1 Wisconsin Prognostic Breast Cancer (WPBC)

This data set (UCI Machine Learning Repository 2019a) contains 32 descriptors of a digitized image of a fine needle aspirate of a breast mass. A total of 198 samples are given, however, we remove four due to missing data. The outcome variable is whether the mass is from a recurrent or a non-recurrent mass. While many classification tests are possible with this data set, we choose simply to apply the PA-SVM model in determining whether the sample is recurrent or non-recurrent.

3.1.2 Quantitative Structure Activity Relationship (QSAR)

The QSAR data comes from the Milano Chemometrics and QSAR Research Group in Milano, Italy, and has been "used to study the relationships between chemical structure and biodegredation of molecules" (UCI Machine Learning Repository 2019b). 41 molecular descriptors characterize 1,055 unique molecules as either ready or not-ready biodegradable. The goal, therefore, is to classify the data into these categories.

3.1.3 Medium Tactical Vehicle Replacement (MTVR)

This data set consists of electronically captured MTVR engine data from various United States Marine Corps ground units on both the east and west coasts of the United States between 2012 and 2014 (Koyak 2019). Each data point is a unique MTVR engine and the associated data is the amount of time the engine spends within a certain RPM band. The data is divided into two classes: east coast units and west coast units, so the goal is to classify the data into these two categories. While this may sound trivial, engine maintenance issues experienced at a specific coast may be due to time spent in unique RPM bands, and these would be identified through our classification method. Knowing the difference in engine usage between coasts, therefore, could help pinpoint maintenance issues.

3.1.4 Training and Test Set Summary

Table 3.1 provides numeric summary data on each of the three sets of data described above. Additionally, the table details the breakdown of each data set into its respective training and test sets. We separate the training/test sets by using an 80/20 split on the entire data, and we give an equal proportional representation of each class in each set.

Table 3.1. Training and Test Set Summary

Data Set	Number of Features	Training Set Observations		Test Set Observations	
		Pos Class	Neg Class	Pos Class	Neg Class
WPBC	32	118	36	30	10
QSAR	41	284	559	72	140
MTVR	16	514	728	129	182

3.2 Setting the Parameters

At the end of Chapter 2, we mention several parameters that must be defined prior to running the PA-SVM model. Some are general parameters that apply to both DCA and SGD, while others are unique to each algorithm. These parameters, while necessary for the model to be used, serve the additional purpose of scoping this work. As an upfront disclaimer, this model, using DCA at least, is not fast, and because of time constraints, we are not able to explore the full factorial design of these parameters. Because of this, we provide

a general exploration of each parameter individually, and we then choose a subset of each parameter's values to create a final, designed experiment. This experiment will be used to obtain and present final results on hinge loss and training/test set misclassification rates. We incrementally build this designed experiment throughout this section.

3.2.1 DCA Parameters

Regularization Terms

We introduce the penalty term, ρ, in Chapter 1 for use on the coefficient variables in the A-SVM model, and we then add an additional term, λ, in Chapter 2 for use on the origin offset variables in the PA-SVM model. While ρ serves a strictly practical purpose in A-SVM, it turns out that the new term, λ, in conjunction with ρ, is actually a necessary component of the PA-SVM linear program in order to maintain numerical control. Use of DCA without these penalty terms results in decision variables taking on values on the order of $1 \cdot 10^{15}$, despite data being on the order of $1 \cdot 10^{1}$. In these cases, the hinge loss appears normal, indicating that solutions fit the training data quite well. These extreme values, however, fail to classify new, unknown data. This is an obvious consequence of overfitting, but in the PA-SVM case, overfitting does not just affect a few test samples, it affects the majority of them. Hinge loss values indicating around 88% training classification resulted in 33% test classification. Regularization is indeed the word to describe the purpose behind these two penalty terms, but it now goes beyond just the practical meaning applicable to A-SVM. One may argue instead for upper and lower limits on the decision variables; while that would certainly work given that one knows the optimal limits, we choose to use penalty terms and let the program optimize for us.

Early exploration of the PA-SVM model shows that after a certain value of ρ and λ, numerical control is guaranteed for all trials. Therefore, we apply these values, not yet optimized for experiments on test data but capable of producing coherent solutions, to the three sets of data for experiments in this chapter. Table 3.2 provides a summary. While values lower than these produce numerically controlled results on *some* trials (and, consequently, lower objective function values), we choose the values in the table for their consistency. Optimizing these terms for test set experiments is left as a subject for the next chapter.

Table 3.2. Data Set Regularization Values

	WPBC	QSAR	MTVR
ρ	$5 \cdot 10^{-4}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
λ	$5 \cdot 10^{-4}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$

Variable Initialization

As this is a nonconvex problem, the algorithm's starting position has a direct impact on the solution that is obtained, and one changes the starting position by initializing the decision variables at a certain value prior to running the model. We explore the following starting position options:

1. Initialize one affine piece to the solution obtained by using the A-SVM model. Initialize everything else at zero.
2. Initialize everything at zero.
3. Initalize everything at a random ($U \sim [0, 1]$) number.

The first two options result in only two solutions (one for each option), since their values never change. It is hypothesized, however, that for the first choice, since the A-SVM solution is optimal in the convex case, it may provide a good starting position in the nonconvex case. For the second option, initializing everything at zero is a natural choice, and it also inherently supports the notion of regularization. The third option is the only one that can truly explore the data's solution space since each trial starts in a new location. The problem with this option, however, is the fact that more than one trial must be conducted, and the quality of the solution is not guaranteed from trial to trial.

To determine which starting point methodology performs the best, we conduct three small tests on the QSAR data set. Each test uses a different starting method, and, as the number of pieces, the Cartesian product of $q = \{1, 2, 3, 4, 5, 6\}$ and $r = \{1, 2, 3, 4, 5, 6\}$, thus providing 36 unique combinations to be explored. In the case of options one and two, the model runs only one trial at each $\{q, r\}$ combination. For the third option, however, we run 25 trials, since each trial has the potential to provide a unique solution, and, therefore, a possibly better solution.

In the above experiments, initializing the variables at a random location at the beginning of each trial outperformed the other starting point options in 33 out of the 36 scenarios within the course of 25 trials. Despite taking more time, we choose to initialize the variables at a random number in hopes of obtaining the best possible solutions.

Number of Pieces

We assume that as the number of affine functions, or pieces, increases, the hinge loss decreases, since better approximations of nonlinear patterns can be achieved as the number of pieces increases. To test this, we use the same Cartesian set of $\{q, r\}$ combinations as in the above experiments and conduct 25 trials (at random starting positions each time) on both the WPBC and MTVR data. We do not conduct another experiment on the QSAR data, since we already have output from the above experiment. We then examine the best solution at each $\{q, r\}$ combination. Figure 3.1 shows these results from the QSAR data set.

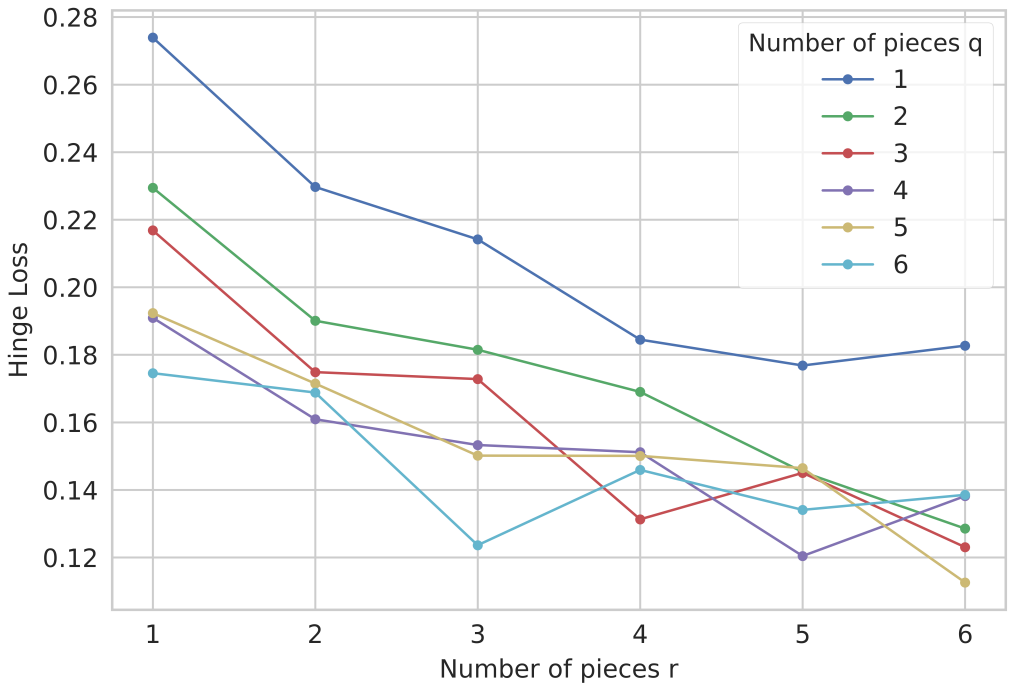

Figure 3.1. Hinge Loss Obtained at Various $\{q, r\}$ Combinations

A general trend confirms our assumption, but Figure 3.1 (as well as results from the other

two sets of data) does not provide overwhelming evidence that more pieces equates to better solutions. Indeed, the overall best solution is obtained by using fewer than $\{6, 6\}$ pieces. Additionally, there seems to be an asymptotic trend as the number of pieces increases. It could be that we merely need to conduct more trials in order to get better results, or it could be that there exists a geometric limitation once certain $\{q, r\}$ values are achieved. While these ideas are definitely worth exploring, we limit ourselves in this work to conducting further trials on only three $\{q, r\}$ combinations: the two that achieve the best results on each set of data (we choose from separate q pieces) and then $\{8, 8\}$. The latter is chosen to further test our motivating assumption. Table 3.3 summarizes these pieces.

Table 3.3. Number of Pieces Tested

	WPBC	QSAR	MTVR
$\{q, r\}$ Combination	{5,5}	{4,5}	{3,6}
	{6,3}	{5,6}	{5,6}
	{8,8}	{8,8}	{8,8}

Number of Trials

We know that by running multiple trials, we are able to explore the solution space, but how many trials must we conduct before we are confident that we have the *best* solution? To answer this question, we conduct 1,000 trials (random restarts) on the QSAR data set using $\{q, r\} = \{3, 3\}$, since, from Figure 3.1, it seems like that combination has the potential to provide a better solution given the trend of the line where $q = 3$. We are interested in two things: how many trials it takes to see the best solution and how many trials it takes to see a better solution than the best one so far. Figure 3.2 shows the lowest (running) hinge loss obtained over the course of this experiment. From the figure, we see that a lot of progress is made by the 100^{th} trial, but it takes 600 additional trials before we see a better solution. While this best solution outperforms almost all of the other solutions in Figure 3.1, it took 30 hours to reach this point, and we unfortunately cannot devote that much time to each $\{q, r\}$ piece. Due to seeing such vast improvements early on, and not seeing an improvement until much longer, we limit our experiments to 100 trials.

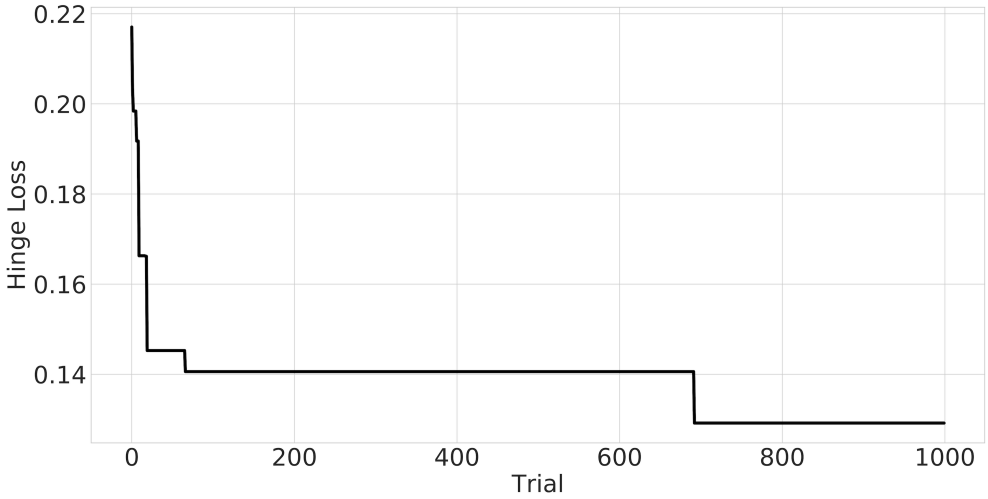

Figure 3.2. Lowest Running Hinge Loss Obtained Over 1,000 Random Restarts

Convexification Thresholds

The parameter values we discuss above, other than the regularization terms, correlate to the quality of a particular solution compared to the quality of another solution. However, there are parameters internal to DCA that impact the quality that an individual solution brings, irrespective of other solutions, and these are what we refer to in Chapter 2 as the convexification threshold parameters. There, we provide an example of DCA's internal behavior on ϵ over the course of 15 internal iterations. In that example, $|\epsilon|$ is set to $1 \cdot 10^{-9}$, and most of the observations reach that value before having to go through 15 iterations. In early experiments, such as the one in Chapter 2, we find that these values are achievable, so they enter our final design as such.

3.2.2 SGD Parameters

To solve the PA-SVM model using SGD, we use all of the parameter values that we obtain in the above experiments, except the convexification threshold values, in order to maintain consistency. There are two additional parameters that must be defined, however, that are specific to the SGD algorithm: number of iterations and step size.

Number of Iterations and Step Size

These parameters, determined in tandem, ensure convergence to a local minimum, and defining them is not a triviality (Chapter 2 cites only a few of the various methods of getting SGD to converge optimally). Many small experiments went into determining the parameter values that offered the best solutions, and Table 3.4 summarizes them.

Table 3.4. SGD Specific Parameters

	WPBC	QSAR	MTVR
Total Iterations (in 1,000's)	120	100	120
Step Size	0.016	0.03	0.016

3.3 The Designed Experiment

Table 3.5 summarizes the parameter values we use to solve the PA-SVM model on the previously discussed data sets. We use the remainder of this chapter to discuss the results.

Table 3.5. Summary of Parameter Values

		WPBC	QSAR	MTVR
DCA and SGD	ρ	$5 \cdot 10^{-4}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
	λ	$5 \cdot 10^{-4}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
	Variable Initialization	Random	Random	Random
	$\{q, r\}$ Combination	$\{5, 5\}$	$\{4, 5\}$	$\{3, 6\}$
		$\{6, 3\}$	$\{5, 6\}$	$\{5, 6\}$
		$\{8, 8\}$	$\{8, 8\}$	$\{8, 8\}$
	Number of Trials	100	100	100
DCA	ϵ	$1 \cdot 10^{-9}$	$1 \cdot 10^{-9}$	$1 \cdot 10^{-9}$
	Internal iterations	15	15	15
SGD	Iterations in 1,000's	120	100	120
	Step Size	0.016	0.03	0.016

3.4 Hinge-Loss

Hinge loss results are summarized in Table 3.6. It is clear from the table that merely increasing the number of pieces does not equate to a guaranteed decrease in hinge loss. The QSAR data set is the only one that achieves its best solution using $\{8, 8\}$ pieces. Additionally, using SGD on the WPBC data set results in an extremely large hinge loss value. While SGD is a proven classification technique, it appears that, in this case, some data points are very far away from the solution.

Table 3.6. Lowest Hinge Loss Obtained

	WPBC	QSAR	MTVR
PA-SVM (DCA)	0.048 {6,3}	0.105 {8,8}	0.058 {5,6}
PA-SVM (SGD)	14.486 {6,3}	0.674 {8,8}	0.246 {5,6}

3.5 DCA Convexification

Table 3.7 summarizes how well each data set meets our convexification thresholds. The top row provides a percentage of trials that meet the ϵ value on or before 15 iterations. For those trials that do not, the remaining rows in the table provide the minimum, maximum, and average difference between the linear program hinge loss and the *convexified* hinge loss that was obtained.

Table 3.7. Summary Statistics on Achieving DCA Convexification Threshold Values

	WPBC	QSAR	MTVR
% Trials Converged	17.0	78.3	86.3
Min Difference	$1 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	$1 \cdot 10^{-8}$
Max Difference	$1 \cdot 10^{-6}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$
Avg Difference	$1 \cdot 10^{-7}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-5}$

An analysis on the trials that do not meet the convexification thresholds shows that the solutions exhibit two behaviors between iterations: either the solutions are far apart and each iteration gives a unique solution, or the solution stays the same from iteration to

iteration, thus indicating that a local solution has been found, and the algorithm cannot improve upon it. The latter case informs yet another convexification parameter, in that if a solution has been visited twice in a row, then the algorithm should end.

3.6 Run Time

We mention at the beginning of this chapter that the PA-SVM model using DCA is not very fast. We show in Figure 3.3 the distribution of run times from the QSAR data set and the hinge loss obtained at those run times. This accomplishes three purposes: first, it gives the reader an idea as to how long it takes for the PA-SVM model to run using DCA; second, it shows that trials that take longer do seem to provide better solutions, but a long run time does not guarantee a solution to be good; and finally, it confirms a natural assumption that more pieces equate to longer run times, in general. While we only show results from the QSAR data set, the same behavior is apparent in the other data sets as well.

Run times using SGD are much faster, less than 200 seconds for all trials, but we do not go into detail due to SGD not providing superior results.

3.7 Classification Error

It is known that the hinge loss approximates the classification error on the training data, and we confirm that here, in Figure 3.4. Notice also that the hinge loss is an upper bound to the error rate (see Chapter 1).

As a means of comparison, we provide A-SVM results along with the PA-SVM results in Table 3.8. While our PA-SVM model using DCA outperforms A-SVM, the SGD method does not share the same result. An interesting takeaway from Tables 3.6 and 3.8 is that the differences between the SGD hinge loss values and their respective misclassification rates are much larger than the differences found from DCA. While the SGD method did not perform relatively well, the hinge loss may not be the best value to compare to other methods without also showing the training error that accompanies each method.

While being able to classify training data is an important aspect of any classification model, it is the model's ability to classify new, unknown data that really matters.

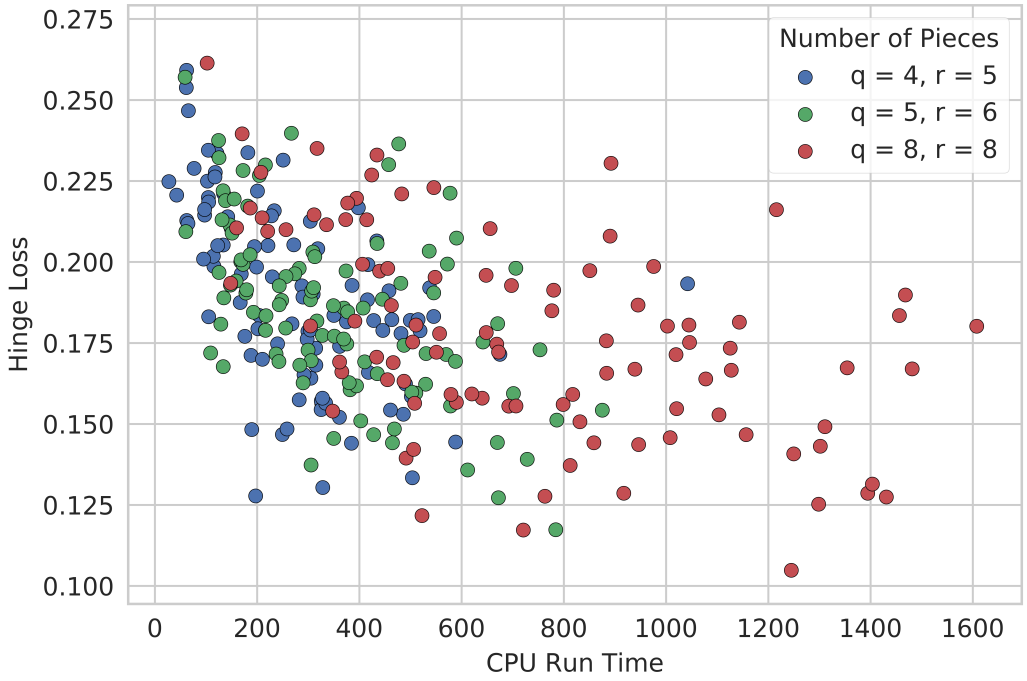

Figure 3.3. Hinge Loss vs. CPU Run Time (Seconds)

Table 3.8. A-SVM and PA-SVM Training Error Comparison

	WPBC	QSAR	MTVR
A-SVM	0.162	0.101	0.077
PA-SVM (DCA)	0.013 {6,3}	0.038 {8,8}	0.022 {3,6}
PA-SVM (SGD)	0.221 {5,5}	0.337 {4,5}	0.077 {5,6}

While the SGD algorithm performed worse or the same as the A-SVM model on two of the three sets of data, better results were found by using penalty values of zero. This may be a result of a sub-optimal SGD technique, but we wanted similar conditions between each nonconvex algorithm in order to compare them.

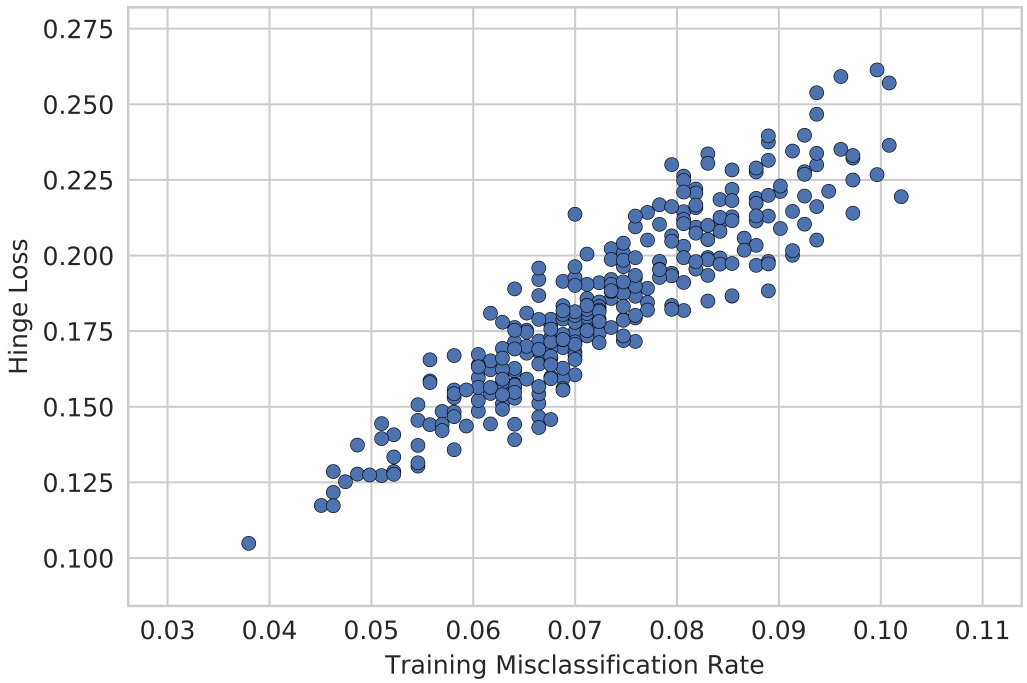

Figure 3.4. Relationship between Hinge Loss and Training Classification Error

CHAPTER 4:
Results on Test Data

In this chapter, we focus on the PA-SVM model's ability to classify novel data. As mentioned in Chapter 1, a classifier's worth is not just characterized by how closely it can fit known data. In fact, there is a point when doing better on a training set translates into worse results on a test set. Therefore, the classifier's ability to accurately predict unfamiliar data holds more value than being able to predict training data really well. In the previous chapter, our two nonconvex algorithms solve for decision variables that optimally classify training data. Now, we put our solutions to the test by applying them on new data in the test sets. We start by providing a summary of test set classification errors, and then we adjust the regularization parameters and conduct further tests on a subset of our designed experiment to determine if better test classification results can be achieved. We end with a general discussion of the model's capabilities and limitations.

4.1 Classification Error

Table 4.1 summarizes the test results (the *best* results from PA-SVM) obtained from the A-SVM and PA-SVM models, and we see that PA-SVM outperforms A-SVM on each set of data. While the SGD method performs better than DCA on the WPBC data set and reduces the test error by 36.4% compared to A-SVM, SGD does not produce similar results on the other sets of data. DCA, on the other hand, reduces the test error by 22.6% on the QSAR data set and 67.2% on the MTVR data set, compared to A-SVM. This represents only a marginal increase in the number of correctly classified data points (4, 7, and 12 in data set order), but the fact that there is any improvement at all demonstrates that the PA-SVM model provides decent test results.

4.2 Tuning the Regularization Parameters

An important aspect of conducting classification experiments on test sets is optimizing the regulatory parameters in order to achieve the best test results possible. This involves finding a balance between overfitting the model on the training data and generalizing the model so much that it not only results in poor training classification but also poor test classification.

Table 4.1. A-SVM and PA-SVM Test Error Comparison

	WPBC	QSAR	MTVR
A-SVM	0.275	0.146	0.058
PA-SVM (DCA)	0.200 {5,5}	0.113 {4,5}	0.019 {5,6}
PA-SVM (SGD)	0.175 {8,8}	0.335 {5,6}	0.061 {5,6}

To do this, we adjust our regularization parameters, ρ and λ, and conduct further trials on the QSAR data set using $\{q, r\} = \{4, 5\}$ since that combination provides the best test results in Table 4.1. Our results are summarized in Table 4.2, and the best solution is in bold.

Table 4.2. Test Error on the QSAR Data Set, $\{q, r\} = \{4, 5\}$

		λ							
		0.00001	**0.0001**	**0.001**	**0.0025**	**0.005**	**0.0075**	**0.01**	**0.025**
ρ	**0.00001**	0.132	0.137	0.132	0.146	0.118	0.127	0.132	0.151
	0.0001	0.127	0.137	0.127	0.137	0.132	0.141	0.127	0.156
	0.001	0.127	0.127	0.113	0.127	0.127	0.123	0.118	0.118
	0.0025	0.123	0.128	0.123	0.123	0.118	0.118	0.113	0.118
	0.005	0.141	0.137	0.123	0.118	0.113	**0.109**	0.123	0.123
	0.0075	0.137	0.123	0.123	0.113	0.113	0.123	0.118	0.123
	0.01	0.137	0.137	0.132	0.118	0.118	0.118	0.123	0.118
	0.025	0.156	0.156	0.151	0.151	0.142	0.137	0.146	0.146

We conducted 200 additional trials with $\rho = 0.005$ and $\lambda = 0.0075$, but the results from those trials only matched the result in the table.

While not obvious in Table 4.2, changing the regularization parameters on the PA-SVM model results in the same behavior on the training and test set error rate as is common in other classification models. In Figure 4.1, we see that achieving a low training classification error rate usually does not translate into a low test error rate. In this scenario, the penalty values are too low, and the classifier overfits the data. On the other hand, applying too much penalty also results in bad test classification rates. The desired result is somewhere in the middle, where one accepts some training error in order to generalize the model just enough to perform well on test data. The combination of ρ and λ that provides the lowest test error in Table 4.2 is indeed in this middle ground.

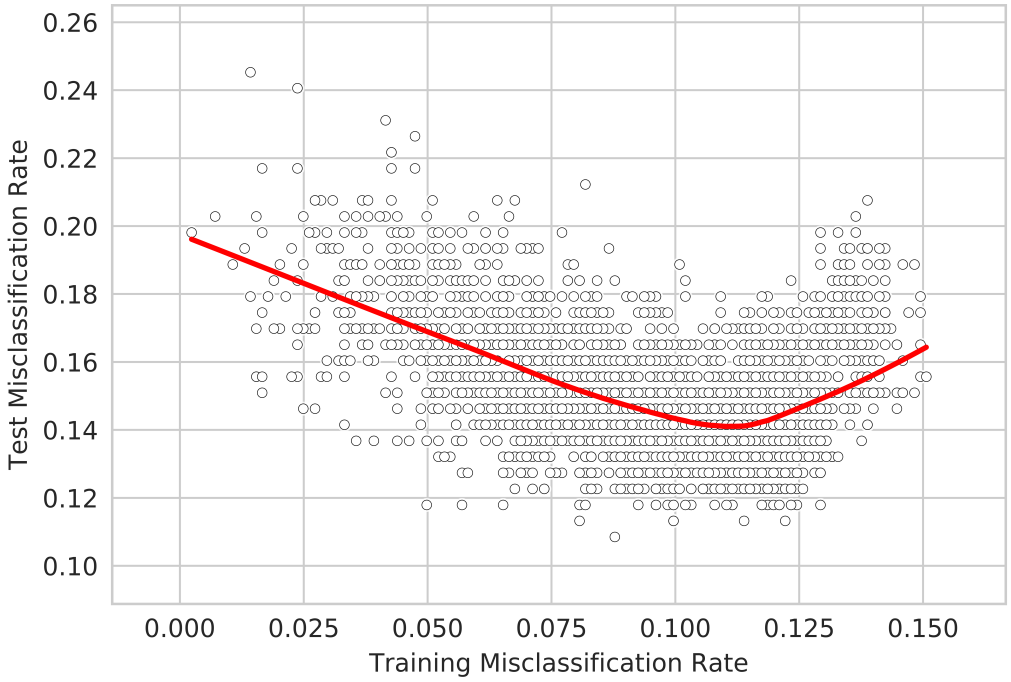

Figure 4.1. Classification Error as a Result of Adjusting Regularization Parameters

While the general trend is obvious, the red lowess curve is added to show where many data points overlay each other.

4.3 Revisiting the Number of Pieces

A concern might be that, regardless of how we tune the regularization parameters above, we are already overfitting the data because of how we choose the three $\{q, r\}$ combinations in Chapter 3. It is natural that more pieces will fit the training data better than fewer pieces, but we might be setting ourselves up for failure from the beginning by choosing the best combinations as applied to the training data. We see decent test results from the PA-SVM model in the tables above, but can we do better? To answer this question, we return to the original experiment on the number of pieces in Chapter 2 and test the solutions. Table 4.3 shows the lowest test error from these observations. We see that the best results come from trials with a lower $\{q, r\}$ combination, and, using the same logic from Chapter 3's experimental set-up, we conduct 100 more trials at these combinations in hopes of finding a better solution.

Table 4.3. Test Error on Original QSAR $\{q, r\}$ Experiment

		r					
		1	**2**	**3**	**4**	**5**	**6**
q	**1**	0.142	**0.118**	0.137	0.132	0.137	0.137
	2	0.127	0.136	**0.118**	0.132	0.132	0.137
	3	0.137	**.0.118**	0.127	0.127	0.127	0.127
	4	0.137	0.137	0.123	0.132	0.132	0.127
	5	0.132	0.127	0.127	0.123	0.123	0.127
	6	0.132	0.127	0.132	0.127	0.127	0.138

The results from these extended experiments are captured in Table 4.4, and we see that with only {2,3} pieces we match the best result in Table 4.2 where we use {4,5} pieces. This is an important discovery, as it is yet another indication that more pieces does not guarantee better results. Additionally, this result is obtained without adjusting the regularization parameters as we had to do in the {4,5} case. This seems to imply that using additional pieces may offer good test results, but in order to achieve those results, the regularization parameters must be fine-tuned. Considering that we had to fine-tune over 2,000 trials to get the best {4,5} result, and we only had to conduct 125 trials to get the same result using {2,3} pieces, the practical benefits of using more pieces may be limited.

Table 4.4. Lowest Test Error after 100 Trials on the Three Best $\{q, r\}$ Combinations in Table 4.3

{1,2}	**{2,3}**	**{3,2}**
0.118	**0.109**	0.118

450 additional trials at various ρ and λ values for {2,3} pieces did not result in a lower test error.

4.4 Considerations on the Number of Trials

In this work, while we conduct a general exploration of the PA-SVM model and determine how classification results change based on adjustments to the model's parameters, we must limit ourselves on how many trials we conduct at each of these parameter combinations. We do this due to time constraints, but we admit that this may indeed be the most influential

factor in finding the best solution. In Chapter 3, Figure 3.2 shows that the best result (in that experiment) is obtained after around 700 trials. While we are interested in obtaining the best result, we do not have thirty hours to devote to a single combination of parameters. That long trial, however, indicates that better solutions may exist for all combinations of parameters, one just needs to conduct enough trials to obtain that result. The two examples in this chapter alone prove that this is the case.

In Table 4.3, we see that the combination $\{4, 5\}$ does not produce the same result found in Table 4.1. The only difference between the two experiments is the number of trials that are conducted to obtain the results. Additionally, Table 4.4 provides a better result than Table 4.3, and again, the only difference in experiments is that the better result comes from conducting more trials. This scenario is a double-edged sword in that the results we find in this work may not demonstrate the true capability that the PA-SVM model has, but in order to utilize the model's potential, one needs a machine capable of solving hundreds of trials quickly. We accept that the need for many trials is not the model's only limitation.

4.5 Interpretability

The A-SVM model benefits from being an inherently easy model to interpret. By merely looking at the scale and sign of a coefficient, one can determine the relationship between that factor and the outcome. This is a benefit one gets for free just by using the A-SVM model to classify data. Granted, the response variable, being either -1 or 1, has no practical connection to the data itself other than providing a means of separation, the coefficients still provide a way of explaining relative differences between categories. The PA-SVM model, on the other hand, does not benefit from being so interpretable. Now, not only are there multiple affine pieces, each having its own set of coefficients, we take the maximum over one set of affine pieces and subtract its value from the maximum of another set. After two dimensions, the geometric consequence of our piecewise affine function muddles the interpretation of the model. We see that the PA-SVM model produces better test results than the A-SVM model, and this is certainly our goal, but the ability to develop intuition and gain valuable insights from any model is a priceless trait to offer. If there truly is a nonlinear pattern in a set of data, the PA-SVM model would be capable of following it, but it would not be easy to know where, or for how long, the pattern exists. Unfortunately, this is precisely the type of information that is often the most insightful.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusions

Classification models assist decision makers on a daily basis by transforming quantitative data into a category of interest. These models provide a means of analyzing the hundreds or thousands of observations in a set of data that the human brain just cannot handle alone. As information becomes easier to gather in quantitative form, the need for these models is sure to increase. Fortunately, advances in computer technology enable the implementation of ongoing optimization research that focuses specifically on developing better classification models.

In this work, we further the research mentioned above and present a novel approach to nonlinear SVM by using a piecewise affine function as a classifier. We show that this PA-SVM model can be solved by using two nonconvex algorithms, DCA and SGD, and that both are capable of making fewer training and test set errors than A-SVM. In general, DCA provides better results than SGD; however, with so many versions of the SGD algorithm, better results may still be possible.

While not the end goal, the PA-SVM model appears to be capable of classifying training data with almost 100% accuracy given enough pieces. This supports the theory behind piecewise affine functions and their ability to approximate upper semi-continuous functions arbitrarily well as the number of pieces increase. Additionally, the mesh free property of our model makes it quite easy to classify the training data with such accuracy. One needs to simply apply just enough penalty to assure numerical control and then provide the model with enough pieces. Defining the domain for each piece is not necessary. A surprising result from our work, however, appears to show that obtaining the best solution does not always require more pieces.

In Chapter 3, some of the best training set results come from using fewer pieces than the most pieces that we incorporate into the experiments, and in Chapter 4, we see that test set accuracy using many pieces and the right combination of regularization values can be matched by using fewer pieces that do not have to be regulated as much. This implies that the common task of finding the optimal regularization values may not be as significant of a

factor when it comes to producing quality test set results as long as the right combination of pieces is used. If this is the case, more computational time can be devoted to conducting additional trials on experiments that use fewer pieces. This would prove beneficial since results from Chapters 3 and 4 show that better solutions are obtained as more trials are conducted. We limited ourselves to the number of trials conducted at each combination of parameter values in order to explore the effect that changing these values had on our model. The solutions we obtain, therefore, are more than likely not the best solutions possible.

We do admit that the model suffers from some limitations. In comparison to the A-SVM model which provides interpretable solutions capable of offering insight into the data, our model's solutions are not so interpretable. The structure of the piecewise affine function makes it difficult to capture the effect that each coefficient has on the outcome, and gaining insight into any nonlinear pattern is even more challenging. Additionally, even though we may be able to save computational time by using only a few pieces, the problem is still nonconvex. Therefore, multiple trials are required, and there is no guarantee on the quality of any one solution.

While we offer a general exploration of the PA-SVM model in this work, there still exists many opportunities for further research. We present an alternative nonlinear approach to the SVM model, but we do not compare our model's abilities and limitations to those of current, existing nonlinear models. By merely comparing PA-SVM results to A-SVM results, we are not necessarily comparing similar models. The utility of our model would be better illustrated by comparing it to other models with similar, nonlinear capabilities. Additionally, applying the PA-SVM model to sets of data in much higher dimensions would truly test the theory that motivated the model's creation. We prove in this work that the model can handle data in up to 41 dimensions, but we believe it is qualified to handle more.

A limitation we identify in this work is the number of trials we are able to run. Therefore, any further research that allows more trials would be beneficial, as it would offer valuable insight into the return on that investment. Using a more powerful computer and/or mathematical programming solver would be an easy start. Additionally, the PA-SVM LP could be replaced by another algorithm other than a linear program. We discuss several alternative approaches to solving the A-SVM model in Chapter 1, and theoretically, any one of them could be used to solve the PA-SVM model as well. Since many deal with computing various types of

gradients, a process that does not require as much relative memory, they may offer added computational efficiency that would allow for more trials.

The research is clear that, due to its ability to minimize hinge loss and reduce test set error, the PA-SVM model is a strong contender in addressing emerging classification problems. A successful application of nonconvex optimization techniques and algorithms, the PA-SVM model reduces classification errors compared to its affine version by only incorporating a few more affine pieces. Since the domain of these pieces does not have to be pre-determined based on each unique set of data, we are optimistic in the model's application to data in higher dimensions than those we experiment on in this study. Although better solutions may still be out there, and despite room for future work, the PA-SVM model has proven its success in that it exhibits the ability to be a robust classifier.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: Computations

A.1 Computation Time

We performed all computations on a 2012 laptop with a 2.9 GHz Intel Core i7 processor and 8 GB RAM using the Python computer language. We implemented all linear programs in the Pyomo (Hart et al. 2017) environment using the CBC (Lougee-Heimer 2003) solver.

A.2 Distribution of Solutions

In Chapters 3 and 4, we discuss PA-SVM results as obtained by using the DCA and SGD algorithms, but we do not mention their distributions. SGD solutions are fairly consistent from trial to trial and thus do not warrant supplementary discussion, but results from DCA deserve further examination. In the following figures, we show kernel density plots of the solutions from the experiments we summarize in Tables 3.8 and 4.1.

While kernel density plots provide some visual inaccuracies at the tails, we choose to use this type of plot for a number of reasons. First, they provide a general reference as to the quality of any one random DCA trial. Second, since we break out the distributions according to each $\{q, r\}$ combination, the reader sees how the distributions can change depending on the combination. Lastly, we provide for reference the A-SVM solution and thus demonstrate both the capabilities and limitations inherent to using the PA-SVM model. We use a Gaussian kernel and limit parameter adjustments to only those necessary in order to factually represent the data.

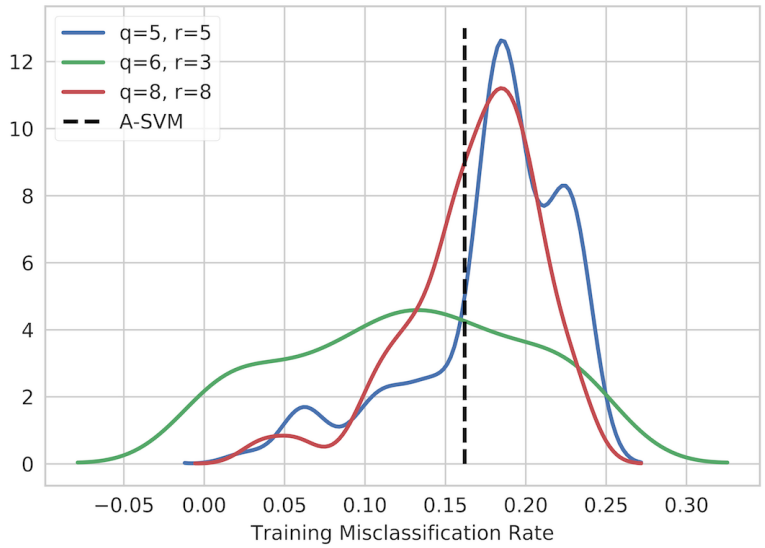

Figure A.1. Distribution of WPBC Training Set Misclassification

Kernel densities do not consider whether a value should be positive or negative. While visually incorrect at the left tail, no actual negative values are obtained in this experiment.

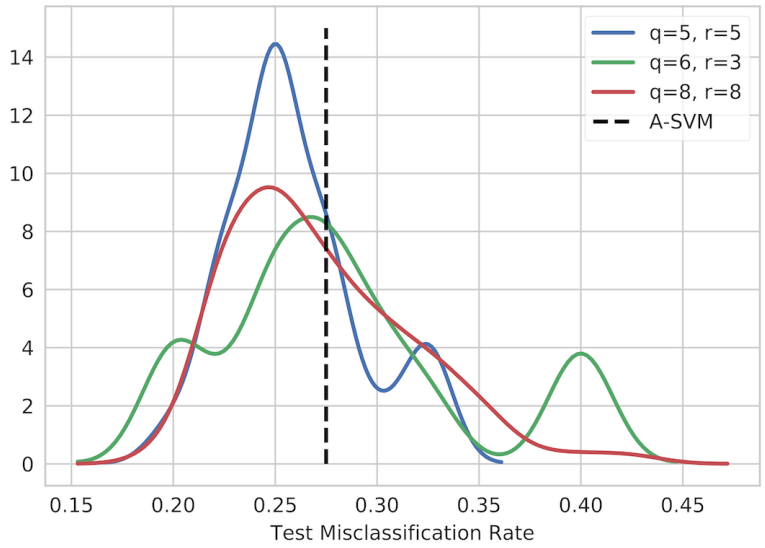

Figure A.2. Distribution of WPBC Test Set Misclassification

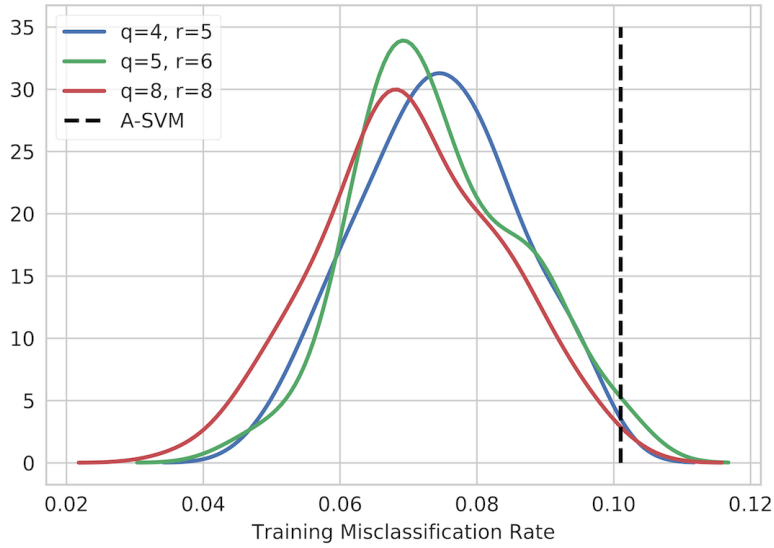

Figure A.3. Distribution of QSAR Training Set Misclassification

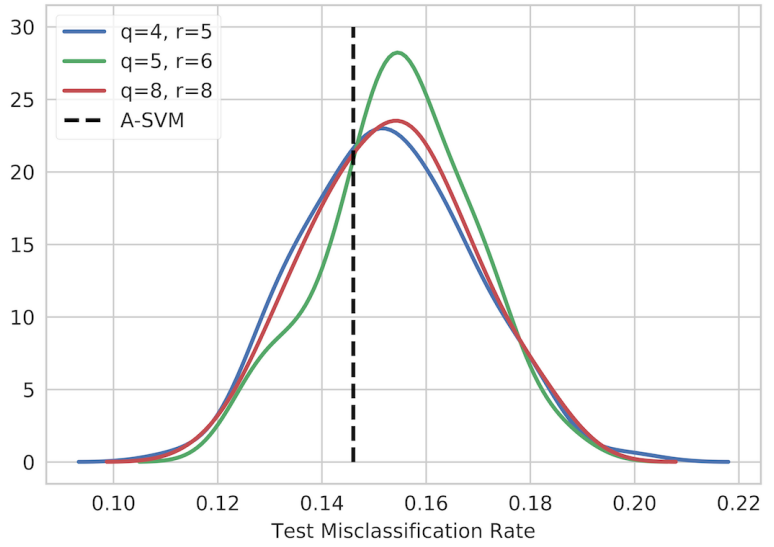

Figure A.4. Distribution of QSAR Test Set Misclassification

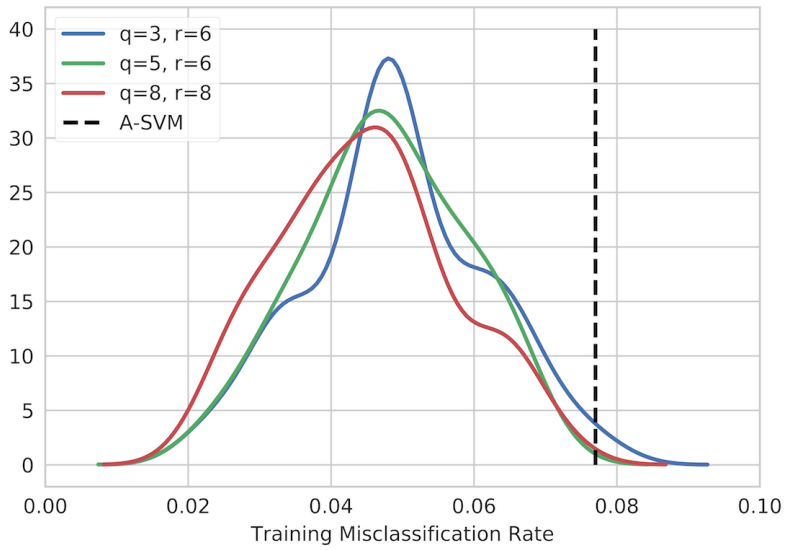

Figure A.5. Distribution of MTVR Training Set Misclassification

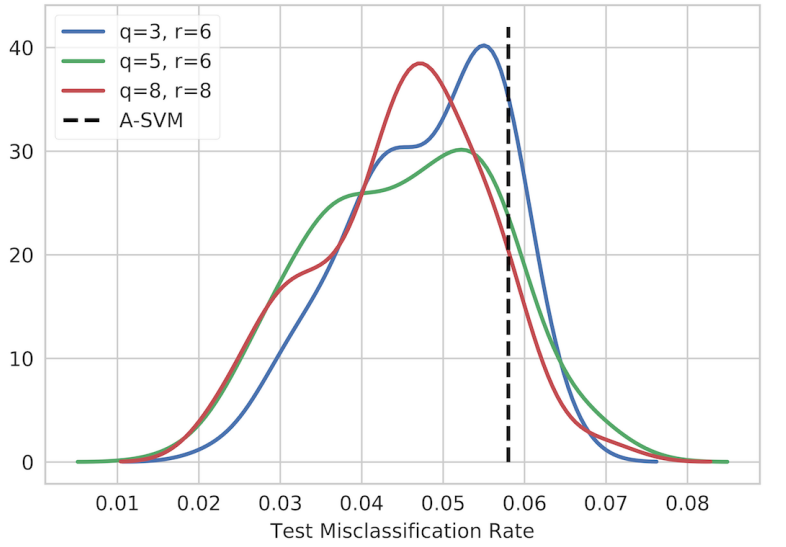

Figure A.6. Distribution of MTVR Test Set Misclassification

List of References

Adeeb S, Troitsky V (2016) Locally piecewise affine functions and their order structure. *Positivity* 21(1):213–221.

Beck A (2014) *Introduction to Nonlinear Optimization-Theory, Algorithms, and Applications with MATLAB* (SIAM, Philadelphia, PA).

Boser B, Guyon I, Vapnik V (1992) A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on computational learning theory* 144–152.

Campbell C, Ying Y (2011) *Learning with Support Vector Machines* (Morgan and Claypool Publishers, CA).

Cheng H, Tan P, Jin R (2010) Efficient algorithm for localized support vector machine. *IEEE Transactions on Knowledge and Data Engineering* 22(4).

Cortes C, Vapnik V (1995) Support-vector networks. *Machine Learning* 20:273–297.

Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12:2121–2159.

Fisher R (1936) The use of multiple measurements in taxonomic problems. *Annals of Eugenics* 7(2).

Gorokhovik V, Zorko O, Birkhoff G (1994) Piecewise affine functions and polyhedral sets. *Optimization* 31(3):209–221.

Hart W, Laird C, Watson J, Woodruff D, Hackebeil G, Nicholson B, Siirola J (2017) *Pyomo–optimization modeling in python* (Springer Science & Business Media).

Hartman P (1959) On functions representable as a difference of convex functions. *Pacific Journal of Mathematics* 9(3):707–713.

Hastie T, James G, Tibshirani R, Witten D (2013) *An Introduction to Statistical Learning with Applications in R*.

Hastie T, Tibshirani R, Friedman J (2017) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction,* 2nd ed. (Springer, New York, NY).

Horst R, Thoai N (1999) Dc programming: Overview. *Journal of Optimization Theory and Applications* 103(1).

Huang H, Toyoizumi T (2017) Reinforced stochastic gradient descent for deep neural network learning, https://arxiv.org/abs/1701.07974.

Joachims T (1999) Making large-scale support vector machine learning practical. Schölkopf B, Burges C, Smola A, eds., *Advances in Kernal Methods - Support Vector Learning*, 169–185 (The MIT Press, Cambridge, MA).

Kecman V (2005) Support vector machines - an introduction. Wang L, ed., *Support Vector Machines: Theory and Applications*, 1–47 (Springer, Berlin Heidelberg).

Keskar N, Socher R (2017) Improving generalization performance by switching Adam to SGD, https://arxiv.org/abs/1712.07628.

Kingma D, Lei Ba J (2015) Adam: A method for stochastic optimization. *Intl Conf on Learning Repr* (The Hilton Resort and Spa, San Diego, CA).

Koyak R (2019) Data provided to the author via email.

Le Thi H, Minh Le H, Nguyen V, Pham Dinh T (2008) A dc programming approach for feature selection. *Advances in Data Analysis and Classification* 2:259–278.

Lougee-Heimer R (2003) The common optimization interface for operations research. *IBM Journal of Research and Development* 47:57–66.

Nam NM, An NT, Le H (2013) Subgradient algorithm, stochastic subgradient algorithm, incremental subgradient algorithm, and set location problems, https://arxiv.org/abs/1303.3735v3.

Neumann J, Schnörr C, Steidl G (2005) Combined svm-based feature selection and classification. *Machine Learning* 61:129–150.

Newton D, Pasupathy R, Yousefian F (2018) Recent trends in stochastic gradient descent for machine learning and big data. *Proceedings of the 2018 Winter Simulation Conference*, 366–380 (The Swedish Exhibition and Congress Centre, Gothenburg, Sweden).

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Pham Dinh T, Le Thi H (1998) A d.c. optimization algorithm for solving the trust-region subproblem. *SIAM J. OPTIM.* 8(2).

Pham Dinh T, Le Thi H (2005) The dc (difference of convex functions) programming and dca revisited with dc models of real world nonconvex optimization problems. *Annals of Operations Research* 133:23–46.

Pham Dinh T, Le Thi H (2018) Dc programming and dca: thirty years of developments. *Math. Program. Ser. B* 169:5–68.

Platt J (1999) Fast training of support vector machines using sequential minimal optimization. Schölkopf B, Burges C, Smola A, eds., *Advances in Kernal Methods - Support Vector Learning*, 185–209 (The MIT Press, Cambridge, MA).

Robbins H, Monro S (1951) A stochastic approximation method. *The Annals of Mathematical Statistics* 22(3):400–407.

Rockafeller R (1970) *Convex Analysis* (Princeton University Press, Princeton, New Jersey).

Royset J (2017) Approximations of semicontinuous functions with applications to stochastic optimization and statistical estimation, https://arxiv.org/abs/1709.06730.

Samudio G (2019) Code to plot output from Equation 2.1 provided to the author via email.

Schölkopf B, Smola A (2002) *Learning with Kernals: Support Vector Machines, Regularization, Optimization, and Beyond* (The MIT Press, Cambridge, MA).

Shalev-Shwartz S, Singer Y, Srebro N, Cotter A (2011) Pegasos: Primal estimated subgradient solver for svm. *Mathematical Programming* 127:3–30.

Tseng P, Yun S (2010) A coordinate gradient descent method for linearly constrained smooth optimization and support vector machines training. *Computational Optimization and Applications* 47:179–206.

UCI Machine Learning Repository (2019a) Characteristics of the cell nuclei from a fine needle aspirate of a breast mass. Accessed April 2, 2019, https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29.

UCI Machine Learning Repository (2019b) Molecular descriptors of ready and non-ready biodegradable chemicals. Accessed April 2, 2019, https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation.

VanderPlas J (2016) *Python Data Science Handbook* (O'Reilly Media, CA).

Wilson A, Roelofs R, Stern M, Srebro N, Recht B (2017) The marginal value of adaptive gradient methods in machine learning. *NIPS* (Long Beach, CA).

Ye Q, Han Z, Jiao J, Liu J (2013) Human detection in images via piecewise linear support vector machines. *IEEE Transactions on Image Processing* 22(2).

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California