Support Vector Machines & an Application to Handwriting Analysis

Chandler Zachary
MATH 6388 Final Project
CU Denver

Submitted December 13, 2018

1 Introduction

Since their introduction by Vladimir Vapnik, support vector machines ("SVMs") have earned a reputation as a superb "out-of-the-box" classifier. They make no assumptions about the underlying data, and they can be used for a wide range of classification problems, especially image recognition and handwriting analysis. The result of an SVM classifier is a decision boundary, instead of a probability, as in logistic regression. They are less sensitive to outliers than logistic regression and linear discriminant analysis. However, they are computationally demanding. In the event that logistic regression is suitable for a binary outcome, it may be computationally more feasible to apply logistic regression than to apply SVMs.

This paper carefully reviews the mathematics of SVMs. I discuss the construction of hyperplanes, margins, and kernels and demonstrate how they fit together to construct SVMs. My treatment of the mathematical exposition follows closely the original development in Cortes and Vapnik (1995) and the subsequent Burges (1998). Furthermore, like Cortes and Vapnik (1995), I apply support vector machines to the handwrittten zip code data, although my application differs in two ways: I use a subset of the data, and I apply a radial basis kernel. The remainder of this paper is constructed as follows: the next section discusses the math in detail, followed by an application of SVMs to the handwritten zip code data.

2 Method

Support vector machines are in the category of *discriminative* classifiers which separate classes based on a decision boundary. This category includes classifiers such as linear and quadratic discriminant and logistic regression. They are distinguished from *generative* classifiers which separate classes based on the distributions of the classes themselves and which include classifiers such as k-means. SVMs estimate decision boundaries called *separating hyperplanes*. I develop the mathematics in three stages, beginning with an introduction to hyperplanes, moving to support vector classifiers, and finally SVMs.

2.1 Separating Hyperplanes & Maximal Margin Classifiers

2.1.1 Separable Case

Support vector machines classify observations according to decision boundaries called *separating hyperplanes*. A separating hyperplane is a flat, affine *(p-1)*-dimensional subspace of a p-dimensional predictor space. In the simplest case to visualize, a separting hyperplane of a two-dimensional predictor space is a line, and observations are classified according to which side of the line they lie. Affine subspaces of three-dimensional space are planes, and those of four-dimensional space are surfaces. In high dimensional predictor spaces, visualizing a hyperplane is not so simple.

Momentarily abstracting away from any reference to data or predictor spaces for the sake of exposition, for a p-dimensional space, a hyperplane can be defined by the following equation:

$$\{\mathbf{x}|f(\mathbf{x}) = b + \mathbf{w}^T\mathbf{x} = 0\}, \tag{1}$$

where $\mathbf{x} = (x_1, x_2, ..., x_{p-1})^T$ is a vector of length *p-1*, and $\mathbf{w}^T$ are the respective weights for the point $\mathbf{x}$; b is a constant. An observation $\mathbf{x}$ lies on either side of the hyperplane based upon whether the value of $\hat{f}(\mathbf{x})$ is positive or negative. This is the foundation of classification using SVMs: assuming a hyperplane exists for the training data, a classifier can be constructed that classifies observations based upon the sign of $\hat{f}(\mathbf{x})$.

Separating hyperplanes are estimated by maximizing the *margin*, which is the smallest perpendicular distance between training observations and the hyperplane. A linear classifier for a two-class response is known as the *maximal margin classifier* because observations are classified by whether they lie further away from the hyperplane than the maximum margin for their respective class. Thus the classifier is constructed using both the sign and the magnitude of $\hat{f}(\mathbf{x})$. If the margins in the test data are similar to those of the training data, then the decision boundary will have a low misclassification rate.

2.1.2 From Maximal Margin Classifiers to Support Vector Classifiers

The maximal margin classifier (also called the *optimal separating hyperplane*) generally solves the following optimization problem:

$$\begin{aligned} &\max_{b,\mathbf{w},\|\mathbf{w}\|=1} \quad M \\ &\textit{subject to} \quad y_i(b + \mathbf{x}_i^T\mathbf{w}) \geq M,\ i = 1, ..., N. \end{aligned} \tag{2}$$

The term M represents the margin as described above. I now show how this problem can be rewritten as a convex optimization problem. This is desirable because of the advantages of solving convex optimization. [Cite the Rockafellar paper here.] I will also show how the problem can be solved for linearly separable classes and thereby lay the groundwork for building the support vector machine.

Consider a training data set $(y_i, \mathbf{x}_i), i \in \{1, ..., k\},\ y_i \in \{-1, 1\},\ \mathbf{x}_i \in \mathbb{R}^p$. If $\mathbf{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$ exist such that

$$\begin{aligned} b + \mathbf{w} \cdot \mathbf{x}_i \geq 1 \quad if \quad y_i = 1 \\ b + \mathbf{w} \cdot \mathbf{x}_i \leq 1 \quad if \quad y_i = -1 \end{aligned} \tag{3}$$

are satisfied, then the training data are linearly separable. The inequalities in (3) can be rewritten to resemble the constraint in (2):

$$y_i(b + \mathbf{x}_i^T\mathbf{w}) \geq 1,\ i = 1, ..., k \tag{4}$$

Observe that three distinct hyperplanes can be defined, and all three are necessary for

constructing the maximal margin classifier:

$$\begin{aligned} b + \mathbf{w} \cdot \mathbf{x}_i &= 1 \\ b + \mathbf{w} \cdot \mathbf{x} &= 0 \\ b + \mathbf{w} \cdot \mathbf{x}_i &= -1 \end{aligned} \tag{5}$$

The second of these hyperplanes is the hyperplane which linearly separates the classes in the training data, and the first and third of these hyperplanes define the margin. The maximal margin classifier maximizes the perpendicular distance between the separating hyperplane and the margin. The perpendicular distances between the margin and the hyperplane can be given by $|1 - a|/||\mathbf{w}||$ and $|-1 - a|/||\mathbf{w}||$, where $||\mathbf{w}||$ is the Euclidean norm. Therefore, the distance between each of the margin hyperplanes and the separating hyperplane is given by $1/||\mathbf{w}||$, and the total width of the margin is $2/||\mathbf{w}|| = M$. Minimizing this total width is given by $\min_{a,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w}$. Thus, the optimization problem (2) can be rewritten as

$$\begin{aligned} &\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ &\textit{subject to} \quad y_i(b + \mathbf{x}_i^T\mathbf{w}) \geq 1,\ i = 1, ..., k \end{aligned} \tag{6}$$

The Lagrangian for this minimization problem is:

$$\mathcal{L} = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=l}^{l} \alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1), \tag{7}$$

where each α_i is an inequality constraint from (6), and the first-order conditions for the

Lagrangian can be written succinctly as:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=l}^{l} \alpha_i y_i \mathbf{x}_i = 0 \\ \frac{\partial \mathcal{L}}{\partial b} &= \sum_{\alpha_i} \alpha_i y_i = 0. \end{aligned} \tag{8}$$

Following optimization procedure, the first condition in (8) is solved for $\mathbf{w}$, and these conditions are then substituted into (7):

$$\mathcal{L} = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{9}$$

This is the advantage of reproducing (2) as a convex optimization problem: minimizing (7) is identical to maximizing (9) subject to the constraints that the derivatives with respect to α_i vanish and $\alpha_i \geq 0$, and the optimization will produce equivalent solutions. The formulation in (7) is called the *Lagrange primal*, denoted $\mathcal{L}_P$, and the formulation in (9) is called the *Wolfe dual*, denoted $\mathcal{L}_D$. The Wolfe dual maximization problem is easier to solve and has fruitful implications for the *kernel trick* described in the next section when using SVMs for non-linear relationships. Moreover, the minimization problem has the advantage that local minima will be global minima.

The solution this optimization problem is the result of solving the first condition in (8) for $\mathbf{w}$, which I reproduce for reference:

$$\hat{\mathbf{w}} = \sum_i \hat{\alpha}_i y_i \mathbf{x}_i. \tag{10}$$

This is an affine subspace of the original predictor space: a hyperplane. The Kuhn-Tucker Theorem adds an additional constraint for $\mathcal{L}_P$ stated above, which allows us to rewrite the

solution in a familiar way and to add some helpful intuition:

$$\alpha_i[y_i(b + \mathbf{x}_i^T\mathbf{w}) - 1] = 0, \quad \forall i. \tag{11}$$

This additional constraint demonstrates two things. First, observe that the optimal separating hyperplane is contained in this condition by noting that for a training observation $\mathbf{x}_i^T$, the decision boundary is defined by:

$$\hat{f}(\mathbf{x}) = \hat{b} + \mathbf{x}_i^T\hat{\mathbf{w}}. \tag{12}$$

and y_i is classified by the sign of $\hat{f}(\mathbf{x})$. Second, having defined the optimal separating hyperplane, we see that $\hat{\alpha}_i$ determines which $\mathbf{x}_i^T$ are actually relevant in determining the optimal separating hyperplane. Non-zero values of $\hat{\alpha}_i$ occur only for the points $\mathbf{x}_i^T$ for which $y_i(b + \mathbf{x}_i^T) = 1$, and these are the points *on the margin M*. When $\hat{\alpha}_i = 0$, this condition is not satisfied, and these are points $\mathbf{x}_i^T$ far away from the margin. This is also the origin of the term *support vector*, which is a point $\mathbf{x}_i^T$ lying on the margin. It merits attention that the elegance of the hyperplane rests on being defined by what it is *not*: the separating hyperplane exists exactly in the center of a corridor where there can be no observations.

2.1.3 Non-Separable Case

When the classes are non-separable such that observations from different classes are comingled, the algorithm must be modified by relaxing the inequality constraints in (6). This is done via *slack* variables $\xi \geq 0, i = 1, ..., k$. The optimization problem then becomes:

$$\begin{aligned} &\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + c\sum_{i=1}^{k}\xi_i \\ &\textit{subject to} \quad y_i(b + \mathbf{x}_i^T\mathbf{w}) \geq M(1 - \xi), \quad \xi_i \geq 0, \quad \forall i \in \{1, ..., k\}. \end{aligned} \tag{13}$$

The role of the ξ_i is directly tied to the effect of misclassification. The maximal margin hyperplane *perfectly* separates classes and is thus prone to overfitting. To add flexibility to the model, we want it to be wrong sometimes. That is to say, we want to capture the instances where $\hat{f}(\mathbf{x})$ is on the wrong side of the hyperplane. This happens when $\xi_i > 1$, and the term $\sum \xi_i$ bounds the number of times $\hat{f}(\mathbf{x})$ is allowed to be on the wrong side. The constant c is a *cost* parameter that defines the width of the margin and has the effect of assigning a cost to violating that margin. A small cost parameter assigns a wide margin which means potentially more support vectors violating the margin, and a large cost parameter assigns a narrow margin which means fewer support vectors violating the margin.

This algorithm is called the *soft margin hyperplane* or the *support vector classifier*, and it is the second step in formulating the support vector machine. It defines a linear boundary for classifying non-separable data. The advantage of the support vector classifier is its flexibility in defining a *nearly* separating hyperplane for the cases where (2) has no solution with $M > 0$. It uses only the observations that lie on or that violate the margin to define the hyperplane. This highlights the role of the parameter c: it can be thought of as moderating the bias-variance trade-off for the the soft margin. Larger values of c expand the number of observations that determine the hyperplane, which means low variance but large bias, and smaller values of c restrict the number of observations that determine the hyperplane, which means high variance but low bias. Furthermore, the support vector classifier uses only those observations that are at or across the margin and boundary, thus mitigating the effects of observations further away. This is a distinguishing characteristic from logistic regression and linear discriminant anaylysis which require all observations.

Solving problem (13) is analogous to solving for the optimal separating hyperplane. First, state the Lagrange primal function and solve its first order conditions to create the Wolfe

dual, then apply the Kuhn-Tucker Theorem. The Lagrange primal is:

$$\mathcal{L}_P = \frac{1}{2}\mathbf{w}^T\mathbf{w} + c\sum_{i=1}^{k}\xi_i - \sum_{i=l}^{k}\alpha_i[y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - (1 - \xi_i)] - \sum_{i=1}^{l}\mu_i\xi_i, \tag{14}$$

where μ_i ensures $\xi_i \geq 0$. The first order conditions are:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \\
\frac{\partial \mathcal{L}}{\partial b} &= \sum_i \alpha_i y_i = 0 \\
\frac{\partial \mathcal{L}}{\partial \xi} &= c - \alpha_i - \mu_i = 0,
\end{aligned} \tag{15}$$

and solving these yields,

$$\begin{aligned}
\mathbf{w} &= \sum_i \alpha_i y_i \mathbf{x}_i \\
\sum_i \alpha_i y_i &= 0 \\
\alpha_i &= c - \mu_i.
\end{aligned} \tag{16}$$

Substituting these solutions into the primal gives the Wolfe dual:

$$\mathcal{L}_D = \sum_{i=1}^{k} -\frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \cdot \mathbf{x}_j, \tag{17}$$

which is maximized according to the constraints that $0 \leq \alpha_i \leq c$ and $\sum_i \alpha_i y_i = 0$. Addi-

tionally, the Karush-Kuhn-Tucker conditions demand:

$$\begin{aligned} &\alpha_i[y_i(\mathbf{x}_i^T\mathbf{w}+b)-(1-\xi_i)]=0 \\ &y_i(\mathbf{x}_i^T\mathbf{w}+b)-(1-\xi_i)\geq 0 \\ &\mu_i\xi_i=0, \end{aligned} \tag{18}$$

and the solution is given by:

$$\hat{\mathbf{w}}=\sum_i \hat{\alpha}_i y_i \mathbf{x}_i. \tag{19}$$

Here again, only non-zero $\hat{\alpha}_i$ pick out the support vectors. Observe the role of $\hat{\xi}_i$ and c in this solution: together, they determine the soft margin. Based on the Karush-Kuhn-Tucker conditions (18), whenever $\hat{\xi}_i = 0$, more or fewer $\mathbf{x}_i^T$ are allowed to lie on or violate the margin according to the third constraing given in (16).

2.2 Support Vector Machines

The previous discussion of maximal margin and support vector classifiers does most of the work in developing the support vector machine. Until now, classifiers have been defined based on linear relationships in the data. However, relationships are not always linear. Moreover, one may not always be able to make an informed decision *a priori* about whether the relationships in the data are linear or non-linear. The so-called *kernel trick* is the final building block that extends the previous classifiers to non-linear decision boundaries and adds greater flexibility. This deceptively simple addition gives SVMs properties that can easily tackle a wide range of classification problems.

Consider a mapping $\Phi : \mathbb{R}^p \to \mathbb{R}^q$, with $p < q$, that expands the input space into larger dimensions. Once the input space is expanded, approximately linear boundaries can be found that separate training observations. In the expanded space, the data become $\Phi(\mathbf{x}) =$

$\Phi_1(\mathbf{x}_i), ..., \Phi_l(\mathbf{x}_i), i = 1, ..., l$. An observation is then classified by the function

$$f(\mathbf{x}) = b + \mathbf{w} \cdot \Phi(\mathbf{x}), \tag{20}$$

where

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \Phi(\mathbf{x}_i) \tag{21}$$

solves the optimization in the expanded space. The "trick" in the kernel trick is to define a kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ that generates support vectors in the expanded input space without requiring an explicit definition of Φ. The mapping Φ can be any combination of basis functions, but one does not need to know what those functions are because the kernel does the heavy lifting. The kernel can do the heavy lifting because dot products are linear. The affine subspaces that solve the optimization problems are dot products, and $f(\mathbf{x})$ in (20) is determined by those dot products. Thus we can write:

$$f(\mathbf{x}) = b + \mathbf{w} \cdot \Phi(\mathbf{x}) = b + \sum_{i=1}^{l} \alpha_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) \tag{22}$$

Since a kernel is defined to be $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$, (22) becomes

$$f(\mathbf{x}) = b + \sum_{i=1}^{l} \alpha_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) = b + \sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}), \tag{23}$$

where the $\mathbf{x}_i$ are the support vectors. Thus reveals the simplicity of the kernel trick. As long as $\mathbb{R}^p$ and $\mathbb{R}^q$ both are dot product spaces whose norms satisfy Cauchy convergence, there exists a mapping Φ that can transform the original non-linear space into a linear space of higher dimensions. Moreover, specification of Φ is not required because the kernel maps the transformed data to be used by the classifier $f(\mathbf{x})$. I refer the reader to either of Cortes and Vapnik (1995) or Burges (1998) for details on Mercer's condition which $K(\mathbf{x}_i, \mathbf{x}_j) =$

$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ must satisfy.

The classifier for a support vector machine is given by:

$$\hat{f}(\mathbf{x}) = \hat{b} + \sum_i \hat{\alpha}_i y_i K(\mathbf{x}_i, \mathbf{x}). \tag{24}$$

There are several popular kernels for SVMs, three of which I reproduce here. The polynomial, radial basis, and neural network kernels, respectively are:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (1 + \mathbf{x} \cdot \mathbf{y})^k \\ K(\mathbf{x}, \mathbf{y}) &= e^{-||\mathbf{x}-\mathbf{y}||^2/2\sigma^2} \\ K(\mathbf{x}, \mathbf{y}) &= tanh(\kappa \mathbf{x} \cdot \mathbf{y} - \delta) \end{aligned} \tag{25}$$

In the expanded input space, optimally separating hyperplanes can be estimated over linear boundaries that are non-linear in the original input space. For example, the radial kernel in reduced dimensions may appear to produce a circular decision boundary with a circular margin. The polynomial kernel may resemble a map of traffic patterns. In the expanded inpute space, $\hat{\xi}_i$ and c perform the same roles as in the soft margin classifier to moderate the number of observations that are allowed to violate the margin.

3 Application

I now proceed with an application of support vector machines to handwriting analysis. SVMs generally have seen tremendous success in image classification. Images that have been processed for digital storage are not necessarily linearly separable, so SVMs are well suited to the task. Furthermore, they take less time to train than neural networks and decision trees, and so are useful for circumstances of limited computational resources. As we will see, though, they are painfully slow for very lage data sets without sufficient computational

Table 1: Distribution of Digits in Training & Test Data

	0	1	2	3	4	5	6	7	8	9
Training	1,149	1,005	731	658	652	556	664	645	542	644
Test	359	264	198	166	200	160	170	147	166	177

power. It should be noted that Cortes and Vapnik (1995) also apply the support vector machine for the same purpose. My study differs from theirs in two ways: first, they use the complete data set of 70,000 observations. I use only a small fraction of this data. Second, they apply a polynomial kernel, and I use a radial kernel.

3.1 Data

The data are 16-pixel by 16-pixel images of handwritten numbers scanned from zip codes on envelopes by the United States Postal Service. Each row of data is a digit from 0 to 9, followed by 256 grayscale values, which represent the amount of gray in a pixel. The data are separated into training and test sets, with 7,291 observations and 2,007 observations, respectively. Figure 1 is a set of images of ten digits taken from the training data. Table 1 gives the distribution of digits in the training and test sets. Note the concentrations on zero and one: they make up almost 30% of the data and 31% of the test data.

3.2 Model Selection

Model selection for SVMs is generally best performed using cross-validation and requires specification of a *cost* parameter and, if a kernel is used, one or more kernel-specific parameters. The radial kernel requires a γ parameter. Cross-validation is then conducted in concert with a grid search over specified ranges for cost and γ. One common method for doing this is to start with a wide grid search and then narrow the range around optimal values.

To give an idea of the effects of different values of cost and γ, I present the true digits against the predicted digits below for the test data. Table 2 shows the results of the support vector

Figure 1: Images of Handwritten Data

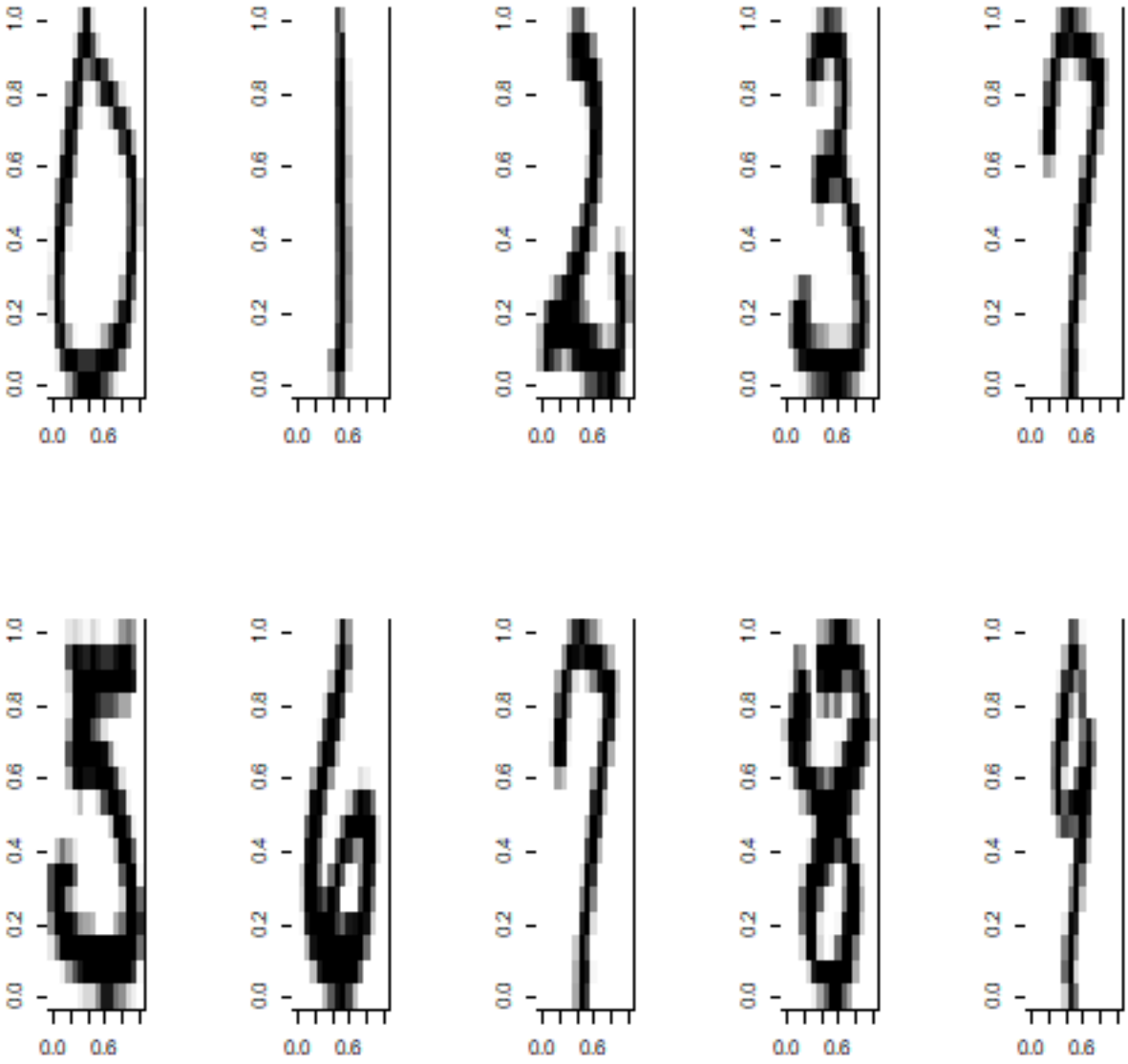

Table 2: Support Vector Classifier, $cost = 1 \times 10^{-4}$

	0	1	2	3	4	5	6	7	8	9
0	345	0	1	3	2	0	6	0	1	1
1	0	258	0	0	3	0	3	0	0	0
2	5	0	157	6	13	2	4	1	10	0
3	4	0	4	141	0	11	0	1	3	2
4	0	5	7	0	178	0	2	1	1	6
5	8	0	0	9	5	129	1	0	3	5
6	7	0	3	0	3	2	154	0	1	0
7	0	2	0	0	9	0	0	124	1	11
8	6	2	2	16	6	4	0	1	123	6
9	0	3	0	0	5	1	0	3	3	162

classifier which only requires the *cost* parameter to be specified. Table 3 shows the results of the support vector machine applied to the test data for $\gamma = 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}$ holding *cost* constant, followed by $cost = 1 \times 10^{-2}, 1 \times 10^{-1}, 1$ holding γ constant.

It is important to note that, although it may not be apparent from the few examples of cost here, using a cost value that is too small risks overfitting the data.

The e1071 package simplifies model selection with the tune() function. The tune() function takes ranges for cost and γ and performs ten-fold cross-validation for each possible combination of the two parameters defined in the ranges, and the optimal combination is associated with the lowest cross-validation error. If one is moving from a wide grid to a narrow grid, then the narrow grid can be defined around the optimal values uncovered in the wider grid, and the tune() function can be executed a second time. This process can be iterated to converge on a global optimum.

This is a good opportunity to highlight one of the more notorious problems with SVMs. The grid search cross-validation can be quite slow and computationally expensive. Since the tune() function performs ten-fold cross validation for every combination of cost and γ, for

Table 3: Support Vector Machine for Differing Values of γ and *cost*

	0	1	2	3	4	5	6	7	8	9
0	350	0	0	0	4	0	4	0	0	1
1	0	260	0	0	3	0	1	0	0	0
2	14	6	138	11	20	1	3	2	0	3
3	16	3	5	128	2	0	2	1	0	9
4	2	20	3	0	169	0	3	1	0	2
5	55	5	1	40	16	23	10	0	0	10
6	44	2	3	0	3	0	118	0	0	0
7	0	6	1	0	24	0	0	59	0	57
8	27	17	6	20	45	0	3	2	5	41
9	1	14	0	0	36	0	0	1	0	125

	0	1	2	3	4	5	6	7	8	9
0	349	0	1	1	2	0	4	0	1	1
1	0	254	0	0	4	0	5	0	0	1
2	5	0	166	4	9	2	2	1	9	0
3	3	0	7	140	0	12	0	1	1	2
4	0	2	8	0	184	0	1	1	0	4
5	7	0	0	9	3	134	0	1	1	5
6	6	0	4	0	3	3	153	0	1	0
7	0	0	2	0	9	0	0	131	0	5
8	5	2	4	15	2	3	0	1	130	4
9	0	3	0	0	4	1	0	3	3	163

	0	1	2	3	4	5	6	7	8	9
0	320	0	35	0	1	0	2	0	1	0
1	0	250	3	0	5	0	4	0	0	2
2	4	0	185	1	3	0	0	0	5	0
3	0	0	49	109	0	7	0	0	1	0
4	0	1	60	0	134	0	1	1	0	3
5	2	0	80	1	0	74	0	0	1	2
6	4	0	28	0	2	0	135	0	1	0
7	0	0	31	0	3	0	0	110	0	3
8	4	0	46	7	2	5	0	0	101	1
9	0	0	11	0	715	0	0	2	0	157

	0	1	2	3	4	5	6	7	8	9
0	350	0	0	0	4	0	4	0	0	1
1	0	260	0	0	3	0	1	0	0	0
2	14	6	138	11	20	1	3	2	0	3
3	16	3	5	128	2	0	2	1	0	9
4	2	20	3	0	169	0	3	1	0	2
5	55	5	1	40	16	23	10	0	0	10
6	44	2	3	0	3	0	118	0	0	0
7	0	6	1	0	24	0	0	59	0	57
8	27	17	6	20	45	0	3	2	5	41
9	1	14	0	0	36	0	0	1	0	125

	0	1	2	3	4	5	6	7	8	9
0	349	0	1	1	2	0	4	0	1	1
1	0	254	0	0	4	0	5	0	0	1
2	5	0	166	4	9	2	2	1	9	0
3	3	0	7	140	0	12	0	1	1	2
4	0	2	8	0	184	0	1	1	0	4
5	7	0	0	9	3	134	0	1	1	5
6	6	0	4	0	3	3	153	0	1	0
7	0	0	2	0	9	0	0	131	0	5
8	5	2	4	15	2	3	0	1	130	4
9	0	3	0	0	4	1	0	3	3	163

	0	1	2	3	4	5	6	7	8	9
0	320	0	35	0	1	0	2	0	1	0
1	0	250	3	0	5	0	4	0	0	2
2	4	0	185	1	3	0	0	0	5	0
3	0	0	49	109	0	7	0	0	1	0
4	0	1	60	0	134	0	1	1	0	3
5	2	0	80	1	0	74	0	0	1	2
6	4	0	28	0	2	0	135	0	1	0
7	0	0	31	0	3	0	0	110	0	3
8	4	0	46	7	2	5	0	0	101	1
9	0	0	11	0	7	0	0	2	0	157

even a wide, cursory grid search of five values for each parameter, that means estimating $5 \times 5 \times 10 = 250$ models. When predictor spaces are very large, this means expanding the predictor space into even higher dimensions. For large data sets and large predictor spaces this can be lengthy and computationally prohibitive.

For demonstration purposes, to avoid this complication, I perform model selection using only one-fourth of the training data, randomly selected, and apply the resulting model to the test data. I begin with broad ranges $\gamma = 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}$ and $cost = 10^{-3}, 10^{-2}, 10^{-1}$, and I define a narrower range around the optimal values found in the first execution. It is possible that I will not converge on a global optimum, but this is for demonstration purposes, so a global optimum is unnecessary. The tables below present the results of the coarse grid search on the left and the fine grid search on the right.

gamma	Cost	Error
0.0001	0.001	0.8353660
0.001	0.001	0.8353660
0.01	0.001	0.8353660
0.1	0.001	0.8353660
1.0	0.001	0.8353660
10.0	0.001	0.8353660
0.0001	0.01	0.8353660
0.001	0.01	0.6981565
0.01	0.01	0.7014442
0.1	0.01	0.8353660
1.0	0.01	0.8353660
10.0	0.01	0.8353660
0.0001	0.1	0.7003573
0.001	0.1	0.1531316
0.01	0.1	0.4072329
0.1	0.1	0.7535909
1.0	0.1	0.8353660
10.0	0.1	0.8353660

gamma	Cost	Error
0.00075	0.0975	0.1893292
0.001	0.0975	0.1558548
0.0025	0.0975	0.1344623
0.005	0.0975	0.1690476
0.00075	0.1	0.1832913
0.001	0.1	0.1525641
0.0025	0.1	0.1322705
0.005	0.1	0.1652045
0.00075	0.125	0.1520116
0.001	0.125	0.1350027
0.0025	0.125	0.1190927
0.005	0.125	0.1547739

The lowest cross-validation in the coarse grid search is approximately 0.1531, for γ and cost of 10^{-3} and 10^{-1}. I conduct a finer grid search around these values: $\gamma = 7.5 \times 10^{-4}, 1.0 \times 10^{-3}, 2.5 \times 10^{-3}, 5.0 \times 10^{-3}$ and $cost = 9.75 \times 10^{-2}, 1.00 \times 10^{-1}, 1.25 \times 10^{-1}$.

The lowest error in the finer grid search is 0.1191, and the γ and cost are 2.5×10^{-3} and 1.25×10^{-1}. This is smaller than the error from the wider grid search, so the finer search is successful. The model with the lowest cross-validation error is stored in *zipTune2$best.model*, and this is the model I will use for prediction with the test data. Table 4 shows the predic-

Table 4: Model Predictions

0	1	2	3	4	5	6	7	8	9	
0	341	0	11	0	1	0	5	0	1	0
1	0	254	0	0	4	0	5	0	0	1
2	4	0	165	5	10	1	3	1	9	0
3	6	0	16	130	1	6	1	0	4	2
4	2	2	18	0	163	1	3	3	0	8
5	10	0	19	6	3	111	5	0	2	4
6	14	0	6	0	3	3	144	0	0	0
7	0	0	8	0	9	0	0	122	0	8
8	5	2	17	8	5	5	2	1	117	4
9	1	3	2	0	17	0	0	7	3	144

tions. The model has a Misclassification rate = 1 - 1,691/2,007 $\approx$ 0.1574.

Cortes and Vapnik (1995) achieve a 1.1% misclassification rate across all 70,000 observations from the MNIST data set with a 4th degree polynomial kernel.

4 Conclusion

This paper briefly discusses the support vector machine by developing the mathematical components that it uses in classification. It also highlights some of the differences between the SVM and other discriminative classifiers. SVMs are an excellent choice when simpler classifiers do not perform well at separating classes. However, for constrained resources, they may prove computationally impractical. Also included is an application of SVMs to image analysis.

References

[1] Burges, Christopher J.C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2), 121-167. doi: 10.1023/A:1009715923555

[2] Cortes, C. and Vapnik, V. (1995). Support-Vector Networks.*Machine Learning*, 2(3), 273-297. doi: https://doi.org/10.1023/A:1022627411411

[3] Friedman, J. et al. (2017) The Elements of Statistical Learning, New York, NY: Springer (12th printing)

[4] James, G. et al., (2017) An Introduction to Statistical Learning: with Applications in R, New York, NY: Springer (8th printing)

[5] LeCun, Yann. (1990) The MNIST Database of Handwritten Digits. Can be accessed at: http://yann.lecun.com/exdb/mnist/

[6] Hsu, Chih-Wei, et al. (2016) "A Practical Guide to Support Vector Classification". National Taiwan University. Retrieved from: https://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf

[7] Meyer, D. et al. (2018). e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R package version 1.7-0. https://cran.r-project.org/package=e1071

[8] Rockafellar, R. Tyrrell (1993). "Lagrange multipliers and optimality". SIAM Review. 35 (2): 183–238. doi:10.1137/1035044