2012-05-16

Support Vector Machines for Classification and Imputation

Spencer David Rogers
Brigham Young University - Provo

Support Vector Machines for Classification and Imputation

Spencer D. Rogers

A selected project submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Statistics

Brigham Young University

June 2012

ABSTRACT

Support Vector Machines for Classification and Imputation

Spencer D. Rogers
Department of Statistics, BYU
Master of Science

Support vector machines (SVMs) are a powerful tool for classification problems. SVMs have only been developed in the last 20 years with the availability of cheap and abundant computing power. SVMs are a non-statistical approach and make no assumptions about the distribution of the data. Here support vector machines are applied to a classic data set from the machine learning literature and the out-of-sample misclassification rates are compared to other classification methods. Finally, an algorithm for using support vector machines to address the difficulty in imputing missing categorical data is proposed and its performance is demonstrated under three different scenarios using data from the 1997 National Labor Survey.

Keywords: support vector machines, SVM, imputation, binary classification, handwritten digit recognition, EM algorithm, NLSY97

ACKNOWLEDGMENTS

I would like to thank everyone who has helped me on this project. Particularly Dr. Christensen, my advisor, and the other committee members, Dr. Grimshaw and Dr. Lawson, for their time and input. Finally, I want my wife to know how thankful I am for her putting up with me during tough weeks and late nights and for encouraging me every step of the way, I literally could not have done this without her.

CONTENTS

CHAPTER 1

INTRODUCTION

In many fields of science there are important questions that involve classifying observations into one of two categories. Examples in everyday life include a bank deciding whether a potential customer will default on a loan, a biologist deciding whether a particular gene is related to cancer, an email program deciding whether to label an incoming email as spam, or an insurance company deciding whether an individual should be classified as high-risk. These types of problems are called binary classification problems. Myriad methods have been devised by statisticians and other researchers to approach these types of problems, including discriminant function classification, logistic regression, neural networks, regression trees, and random forests, to name a few.

In this paper another binary classification method will be presented. This method is called Support Vector Machines (SVMs). First, the method will be motivated and the details of the SVM classification technique will be presented including a review of the relevant academic literature. An extended example using a classic data set from the machine learning literature to compare SVMs to the aforementioned competing methods will demonstrate that SVMs perform quite well relative to other methods in terms of out-of-sample misclassification rates.

The final portion of the paper will be devoted to extending SVMs into the realm of data imputation. Here a new algorithm for imputing missing categorical data will be proposed and compared to the results obtained by naively applying the EM algorithm.

CHAPTER 2

TECHNICAL BACKGROUND AND LITERATURE REVIEW

2.1 Overview

It is important to note that SVMs were developed by computer scientists under the machine learning paradigm and not by statisticians in a regression or probabilistic paradigm. Thus, the terminology related to SVMs can sometimes be confusing to statisticians. In this paper SVMs will be discussed using the terminology of statistics wherever possible instead of the traditional lexicon of machine learning. The area of machine learning that deals with questions of classification is called "supervised learning". Supervised learning refers to the situation where the researcher fits a model using training data and then tests his or her model on a test set of data to see how well it does at predicting class membership. In statistics this is referred to as model fitting and prediction using cross-validation. SVM regression does exist—the responses are continuous as opposed to categorical—but that type of SVM will not be addressed in this paper.

In the case of SVMs, the training data will be a set of n points, $(y_i, \mathbf{x}_i)$, where $i =$ 1,...,n. Each y_i is a tag that assigns one of two classes to the observation. Each $\mathbf{x}_i$ is a vector of p covariates taking values in $\mathbb{R}$ that represent different features of the observation. Thus each of the n observations is a single row of the data matrix that is comprised of p covariates coupled with a tag indicating which of the two responses that observation corresponds to. Each observation can be thought of as a point or vector in p space. In order to understand what the SVM algorithm does to classify observations, let us look at a simple case where there are only two columns of $\mathbf{x}$. Some fabricated sample data points are plotted in Figure 2.1 below.

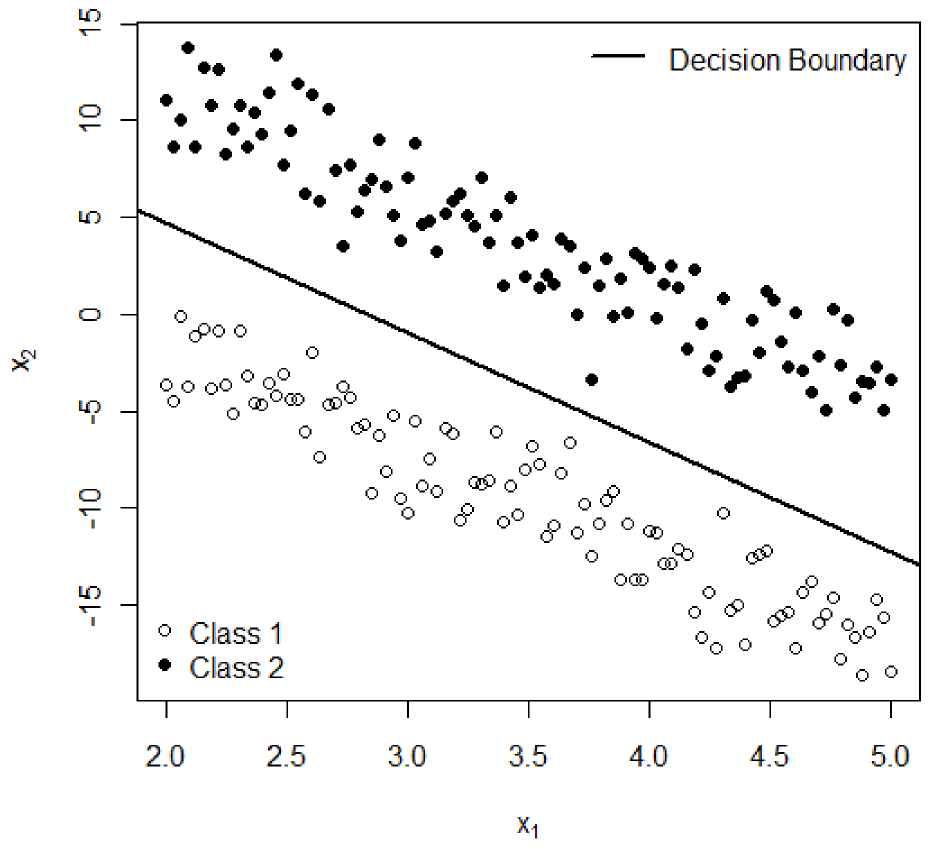

Figure 2.1: Linearly Separable Data

Although this example is trivial, as these data points are very easily separable into their respective categories, it demonstrates an important first assumption with SVMs: that the points are linearly separable by a hyperplane. This assumption will be relaxed later on. In $\mathbb{R}^2$ this separation can be made using a line, as demonstrated above. In higher dimensions this line becomes a plane or hyperplane. The classification rule is then simply to assign any future data point to Class 1 if it falls below the hyperplane and to Class 2 if it falls above the hyperplane. The difficulty then lies in choosing which hyperplane, or decision boundary, best separates the data into their respective classes such that future observations will be classified correctly as often as possible. That task is at the heart of SVMs.

2.2 Foundations of Support Vector Machines

The first paper to propose a method for linear classification was by R.A. Fisher, who in 1936 proposed the use of the discriminant function, $\mathbf{a} = \mathbf{S}_{pl}^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$, in order to predict group membership (Fisher 1936). Then in 1957 Frank Rosenblatt came up with a new idea for pattern recognition and linear classification that he called the "perceptron" and he published a book on it in 1962 (Rosenblatt 1962). Rosenblatt's perceptron laid the groundwork for neural networks and, later, SVMs. The perceptron was based on the following decision function:

$$f(x) = \begin{cases} 1, & \text{if } \ w \cdot x + b > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

where w is a vector of weights, $w \cdot x$ is a dot product, and b is a bias term. The algorithm changes the vector of weights at each iteration until the observations are all classified correctly. Because this is a linear classifier, if the training data is not linearly separable the algorithm will fail to converge.

2.3 Generalized Portraits for Linearly Separable Data

Building on the perceptron concept, in 1963 Vladimir Vapnik and A. Lerner proposed an algorithm for pattern recognition and classification that they referred to as a "generalized portrait" (Vapnik and Lerner 1963). In their paper they explained that if observations in the training data are linearly separable then a good classification algorithm would maximize the margin between the decision boundary and the nearest points of each class. Once the margins have been maximized they come to rest on (or are *supported* by) a small subset of the n observations, hence the phrase *support vectors.* This concept is demonstrated in Figure 2.2.

To understand how this margin space is maximized to find an optimal decision function, some mathematical notation is needed. First we convert the class values of the response,

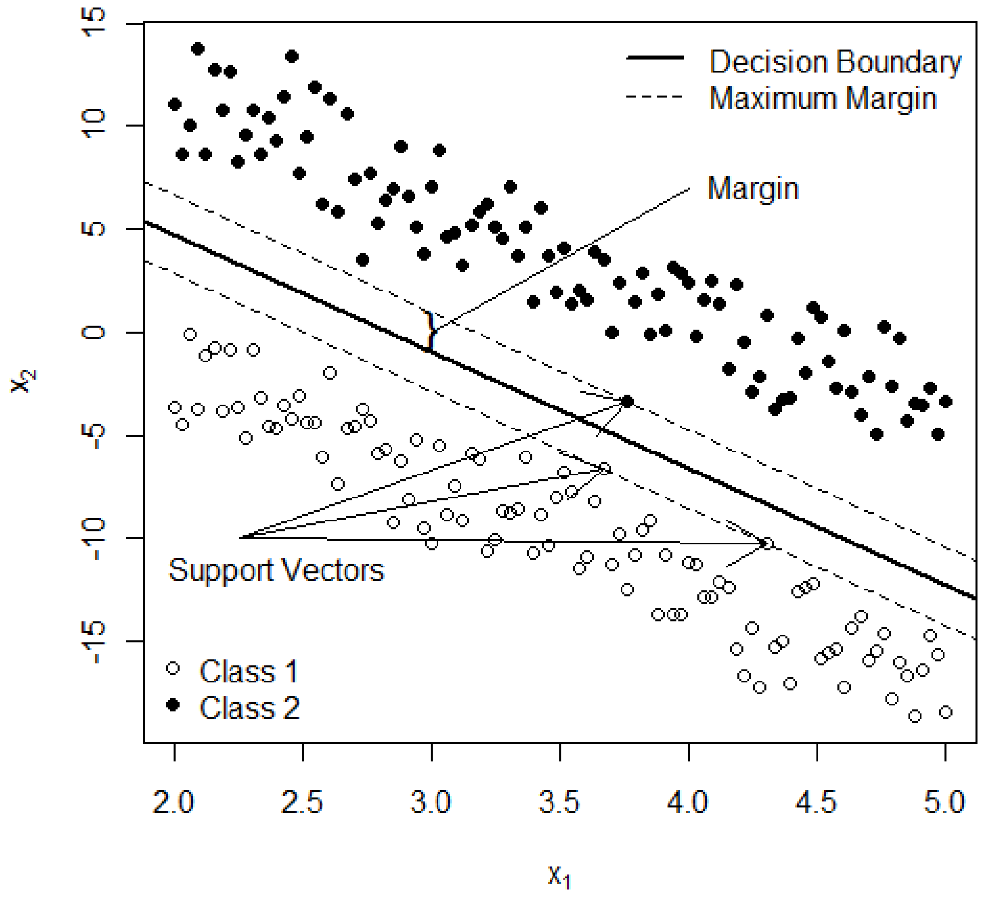

Figure 2.2: Generalized Portrait Algorithm on Linearly Separable Data

$\mathbf{y}$, to numeric values according to the following rule:

$$y_i = \begin{cases} 1, & \text{if } y_i \in \text{Class 1} \\ -1, & \text{if } y_i \in \text{Class 2.} \end{cases}$$

The equation of the decision boundary hyperplane is then given by $f(\mathbf{x}) \equiv \mathbf{w} \cdot \mathbf{x} + b = 0$, where $\mathbf{w}$ is an orthogonal vector to the hyperplane and b is an offset parameter, note the similarity to the perceptron algorithm. The decision function will then be represented as $D(\mathbf{x}_i) = \text{sign}(f(\mathbf{x}_i))$. In general there will be infinitely many hyperplanes separating the data, thus one must maximize this function in order to find the hyperplane with the "fattest" margins. The perpendicular distance between the two margins is given mathematically as $2 \times \text{margin} = 2(\mathbf{w} \cdot \mathbf{w})^{-1/2}$. Finally, the optimization problem of finding the maximum margin can be shown to be equivalent to minimizing $\frac{1}{2}(\mathbf{w} \cdot \mathbf{w})$ subject to the constraint that $y_i * (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \ \forall i$. Solving this optimization in its dual form involves the calculation of the gram matrix, G, which is just the matrix of dot products of all of the $\mathbf{x}_i$'s. Although the details of how this optimization is carried out are important, it is outside the scope of this paper to present them here. The interested reader can find the details in Boser et al. (1992); Hamel (2009); Press et al. (2007).

2.4 Optimal Marginal Classifiers for Nonlinear Classification

Although generalized portraits were a powerful new development in classification analysis they remained largely unused for many years due to the unrealistic assumption that the data must be linearly separable. Until this assumption was relaxed, SVMs were rather useless in practice as real data are typically too noisy to be neatly separated by a line.

That was until the early 1990s, when Vladimir Vapnik was working at AT&T Bell Laboratories on the problem of handwriting identification. He and his collaborators were working with the U.S. Postal Service to create an algorithm that could identify the zip code digits on U.S. mail. While working on this problem, Vapnik and his collaborators came up with an idea they called Optimal Marginal Classifiers (OMC). OMC was the logical extension to the generalized portrait algorithm in that it allowed for a nonlinear separation boundary between the classes. In 1992, at a workshop on machine learning, Vapnik presented a paper along with Bernhard Boser and Isabelle Guyon on OMC (Boser et al. 1992).

The generalization that was made with OMC, in order to allow for nonlinear decision boundaries, was that the p dimensional covariate vector, $\mathbf{x}_i$, can be mapped into a much higher dimensional space using a kernel function (Aizerman et al. 1964). The resulting algorithm is quite similar to that described in Section 2.3 except that the dot products in the gram matrix, $\mathbf{x}_i \cdot \mathbf{x}_j$, are replaced with a nonlinear kernel function denoted $k(\mathbf{x}_i, \mathbf{x}_j)$. The basic idea is that even if the data points are far from being linearly separable in the p space in which they reside, they can almost always be mapped into a much higher dimensional space where they can be separated by a linear hyperplane. Even though the classifier is still a linear hyperplane in this higher dimensional space, it will be nonlinear in the original p space. The concept is demonstrated in the plot below.

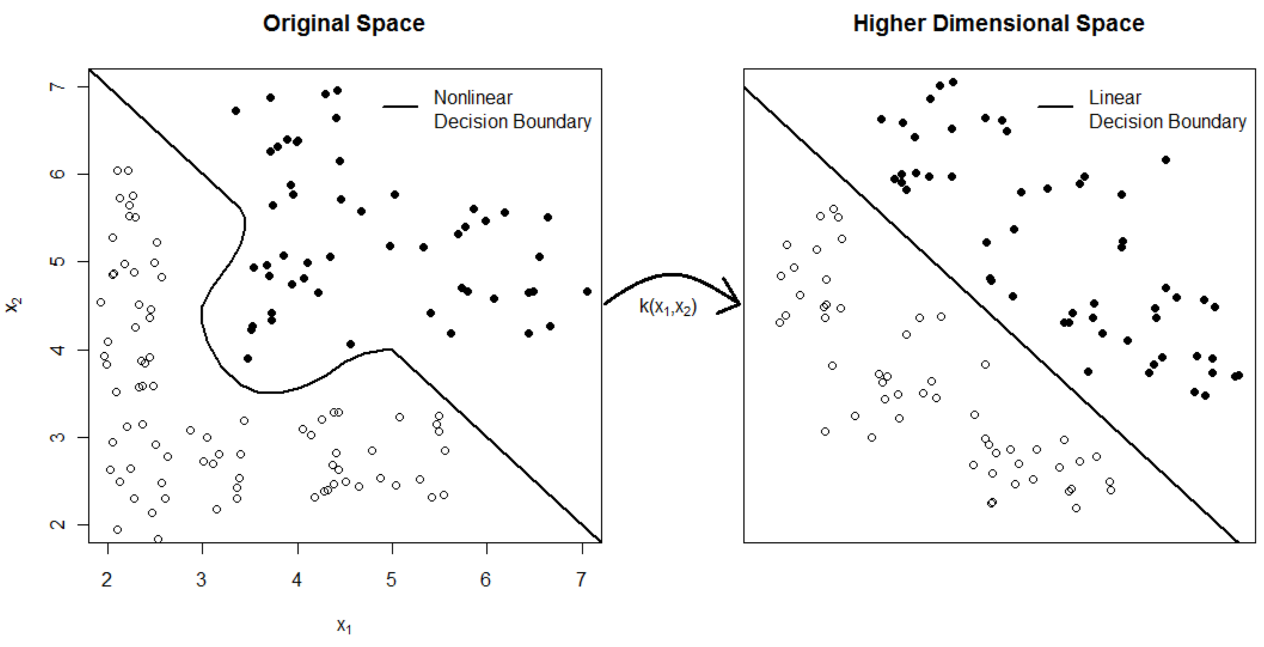

Figure 2.3: Kernel Mapping

Although the implementation of the kernel trick is fairly straight forward, the details of it are beyond the scope of this paper. The theory surrounding this kernel mapping is complicated, involving the use of reproducing kernel Hilbert spaces and other topics from topology; the reader is referred to Boser et al. (1992); Mangasarian (1965); Aronszajn (1950); Aizerman et al. (1964); and Press et al. (2007) for more details. The most common kernel functions that are used for SVMs are the linear, polynomial, sigmoid, and Gaussian radial basis functions. These kernels are easily computable and generalize well to work with almost any dataset. It is this kernel trick that gives SVMs their power, bringing them into mainstream use.

2.5 Support Vector Machines for Soft-Margin Classification

Although it allowed for a separating hyperplane to be found for any data set, no matter the level of noise, the kernel trick had the inevitable effect of over-fitting models to the training data at the expense of poor out-of-sample performance. In 1995, three years after his initial paper on optimal marginal classifiers, Vapnik published another paper on the topic along with Corinna Cortes entitled *Support Vector Networks* (Cortes and Vapnik 1995). In this paper Vapnik relaxed the assumption that all the training data necessarily be classified correctly by the separating hyperplane. This is the point at which he began calling this type of classifier a support vector machine instead of an optimal margin classifier or generalized portrait.

In order to allow for misclassification in the model, Vapnik and Cortes introduced a slack variable, ξ_i, for each data point (Smith 1968). If the data point is classified correctly by the hyperplane, then $\xi_i = 0$; if it is not, then $\xi_i > 0$ in the amount of the discrepancy. Now the optimization problem also becomes a regularization problem. Here another parameter, λ, is introduced representing the "cost" trade-off of misclassification. As λ is varied in the range $0 < \lambda < \infty$, a trade-off curve is explored. As λ approaches 0 a "wiggly" separation between classes is formed classifying every data point correctly at the cost of potentially being over-fit. As λ gets larger a smooth boundary between the two classes is formed allowing outliers, or otherwise difficult data points, to be misclassified. Finally, it can be shown that optimizing this problem becomes equivalent to minimizing $\frac{1}{2}\mathbf{w} \cdot \mathbf{w} + \lambda \sum_i \xi_i$ subject to the constraints that $\xi_i \geq 0 \; \forall i$ and $y_i * (\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1 - \xi_i \; \forall i$. Again, the kernel trick can be used here to map this problem into a higher dimensional space. In the present vernacular, this type of SVM, which relaxes the assumption of linear separability, is referred to as a soft-margin SVM while the former implementation, requiring all the training data to be classified correctly, is referred to as a hard-margin SVM.

In no place have there been any assumptions made regarding statistical distributions. Thus, it should be clear that the classifications made by an SVM algorithm are not probabilistic in nature. That is to say, the SVMs do not produce estimates of $P(y_i = 1|\mathbf{x}_i)$, they simply state whether an observation falls on one side or the other of a decision boundary. That being said, some efforts have been made in recent years to understand the statistical properties of SVMs.

In 2004, Wu and his collaborators showed that when certain assumptions are made an SVM model can indeed yield $P(y_i = 1|\mathbf{x}_i)$ or the posterior probabilities of class membership (Wu et al. 2004). In 2008, Jiang and his collaborators derived the consistency and asymptotic normality of the prediction error estimators as well as propose a procedure that allows for confidence intervals to be obtained on the out-of-sample misclassification rates (Jiang et al. 2008). Also in 2008, Steinwart and Christmann showed the consistency and robustness of SVMs under certain conditions and derived their learning rates (Steinwart and Christmann 2008).

CHAPTER 3

EXAMPLE USING CLASSIC MACHINE LEARNING DATASET

There is a package in R for doing SVMs; the package is titled `e1071` and is built upon an award-winning implementation of SVMs written in C++ called `libsvm` (Xie 2007). The data that will be used here to demonstrate support vector machines is the same zip code data for which SVMs were invented and is a classic data set used in the machine learning literature. On the website `<http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/>` there is a dataset called `zip.digits` that contains two data files: a training data set and a test data set of handwritten zip code digits scanned from U.S. mail. The scanned images have automatically been de-slanted and normalized to all be the same size and orientation. The training and test data contain 7,291 and 2,007 observations respectively. The observations in each dataset are distributed among the digits as follows:

	0	1	2	3	4	5	6	7	8	9	Total
Training	1,194	1,005	731	658	652	556	664	645	542	644	7,291
Test	359	264	198	166	200	160	170	147	166	177	2,007

There are 257 columns in the data; the first corresponds to the digit number and the remaining 256 columns of the data correspond to a measure of darkness of each pixel in a 16×16 grid whose values range from -1 to 1. A value of -1 indicates that there was no writing found in the pixel and a value of 1 indicates that the given pixel was completely colored in. By transforming the pixel values to a scale from 0 to 1, they can be used as gray-scale values in order to plot the handwritten digit on a grid. To give the reader an idea of the layout of the data, a plot of a single observation is given in Figure 3.1 below and the average values at each pixel for each digit in the training data are plotted in Figure 3.2.

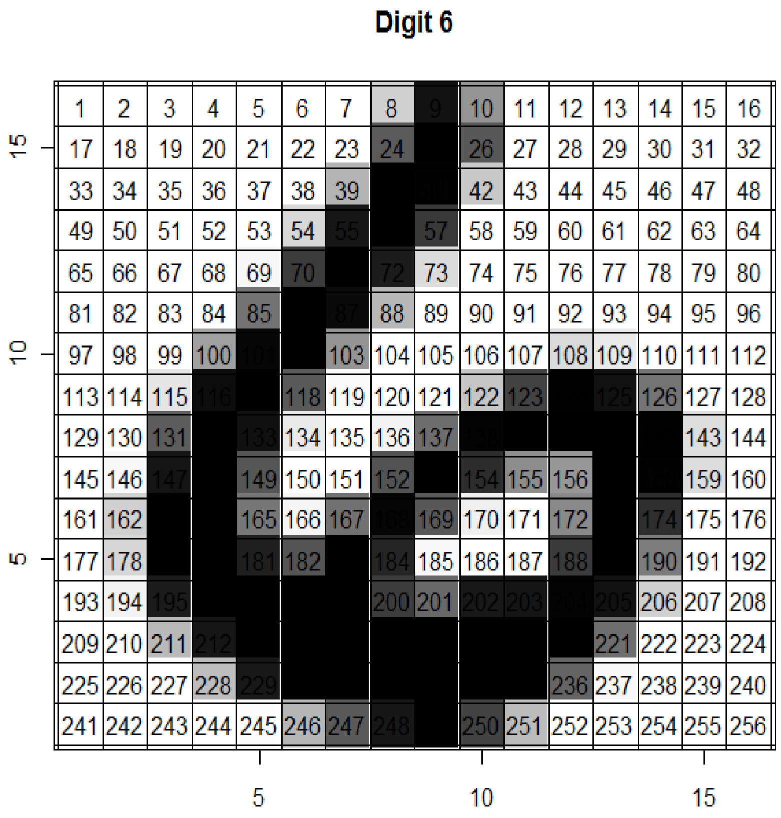

Figure 3.1: First Observation in Training Data Set

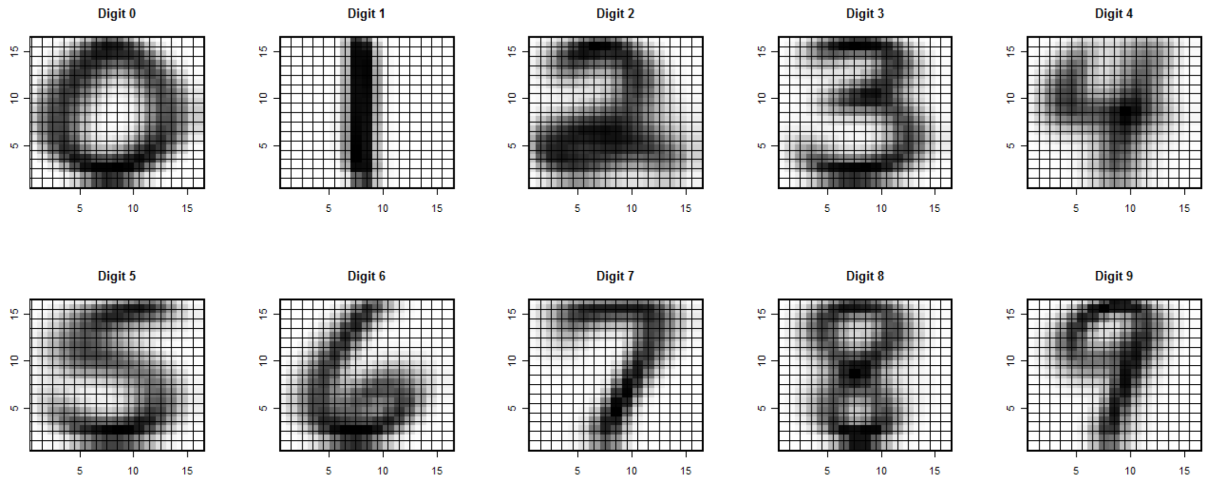

Figure 3.2: Average Grayscale Values for Each Digit

Reassuringly the average values of the ten digits resemble the ten digits 0–9. Since the SVM algorithm was designed for binary classification problems, it will be applied first to the problem of tellings fours and nines apart. The numbers four and nine are purposefully chosen here because these two digits share the most pen strokes and are heuristically the most difficult to tell apart. After this is done, the example will be generalized to the true objective of recognizing any given digit.

3.1 Two-Digit Classification

In order to run the SVM algorithm, one must first decide which kernel function will work well for the data and then tune the values for the parameters corresponding to that particular kernel function. The cost parameter λ, which controls the tradeoff between overfit and underfit, also has to be tuned. In the `svm` function in R, the user has the following choices of kernel functions:

$$\text{Linear} = u' * v$$

$$\text{Polynomial} = (\gamma(u' * v) + \beta_0)^d$$

$$\text{Radial Basis (Gaussian)} = \exp\left\{\gamma(u - v)^2\right\}$$

$$\text{Sigmoidal} = \tanh\left(\gamma(u' * v) + \beta_0\right)$$

In the case of the zip code digit data, it has been shown that the Gaussian and polynomial kernels work well (Christianini and Shawe-Taylor 2000). In order to tune the parameters of these kernels, the `tune.svm` function is employed. The way this tuning function works is that the user first feeds the function vectors of possible parameter values. Then `tune.svm` does a greedy search comparing the leave-one-out cross validation error rate of all possible models of the `svm` function using each of the possible combinations of parameter values (Chang and Lin 2011). The user must be careful here as the number of combinations gets large very quickly. For example, if three parameters are being tuned and the user provides ten different candidate values for each parameter, the tuner will have to fit $10^3 = 1000$ different models in order to report the best fit!

After running the tuner on the four and nine training data, the following parameter values seemed to work best for each kernel. For the Gaussian kernel (using the notation from above) $\gamma = 0.003$ and $\lambda = 100$. For the polynomial kernel $\gamma = 0.004$, $\beta_0 = 0$, $d = 4$, and $\lambda = 10$. Running the `svm` algorithm with these kernels and parameter values yielded the following prediction results:

		In-Sample	Out-of-Sample
	Kernel	Misclassification Rate	Misclassification Rate
Digit 4	Gaussian	0.000	0.030
	Polynomial	0.000	0.020
Digit 9	Gaussian	0.000	0.028
	Polynomial	0.000	0.028

Table 3.1: Classification Comparison of Digits 4 and 9

No matter which kernel function is used, the algorithm will mis-classify an observation about 3% of the time. It is interesting to note that both kernels were able to perfectly separate the training data, and the classification of digit four improved moving from a Gaussian kernel to a polynomial kernel but stayed the same for digit nine. Now that it has been shown that the algorithm works well at telling two digits apart, the real goal of identifying any given digit is addressed.

3.2 Ten-Digit Classification

It has been made clear that SVMs are used for binary classification problems. In the zip code data however, the problem is a ten-way classification problem. There have been several different methods proposed for extending SVMs to multi-class situations, including a "one-against-one" approach, a "one-against-all" approach, and an approach called DAGSVM. The `svm` algorithm in R implements the one-against-one approach because research indicates that this approach performs as well or better than the other methods in most instances (Hsu and Lin 2002). The one-against-one method is implemented by breaking the k-way classification problem down into $\binom{k}{2} = \frac{k(k-1)}{2}$ binary classification problems. In this example $k = 10$ so the algorithm will fit $\binom{k}{2} = 45$ binary classification models. Once the algorithm has fit the

45 models, it uses a voting scheme and assigns the classification tag to an observation based on which digit it was most often assigned to in the 45 models. In the case of a tie, the algorithm is less thoughtful and simply assigns the class based on which class appears first in the array holding the class names (Chang and Lin 2011).

The same process of choosing kernels and tuning parameters was followed for the ten-way classification as in the two-way classification mentioned previously. For the Gaussian kernel $\gamma = 0.009$ and $\lambda = 100$. For the polynomial kernel $\gamma = 0.004$, $\beta_0 = 0$, $d = 4$, and $\lambda = 100$. After fitting this model, several other competing methods were employed in order to appreciate how SVMs perform in comparison. The results of `svm` as well at the other methods are summarized in the Table 3.2 below. They are placed in order of out-of-sample misclassification rates, from best to worst.

Table 3.2: Classification Prediction Comparisons

Classification Method	In-Sample Error Rate	Out-of-Sample Error Rate
SVM (Polynomial)	0.00014	0.04584
SVM (Gaussian)	0.00014	0.04684
K-Nearest Neighbors (k = 5)	0.02057	0.05531
Random Forests	0.02963	0.06129
Neural Networks	0.00000	0.07225
Discrim Classification (Linear)	0.07800	0.11430
Discrim Classification (Quadratic)	0.03240	0.14310
Discrim Classification (Normal Kernel)	0.00000	0.17890
Discrim Classification (3 Nearest Neighbors)	0.10990	0.22560
Rpart (Regression Tree)	0.23138	0.27504

As shown in the table above, SVMs outperform all competitors in predicting out-of-sample class membership for this particular data set. As an aside, more time was spent tuning the parameters of the SVM model than the others, thus k-nearest neighbors, random forests, and neural networks can all be seen as performing as well or nearly as well as SVMs on this particular dataset. Interestingly, despite performing best out-of-sample, SVMs do not have the lowest *in-sample* misclassification rate, as there is one observation that is failing to fit. Closer examination of this data point reveals one of the strengths of the SVM algorithm. The observation that the algorithm is failing to classify correctly in the training data is plotted in Figure 3.3.

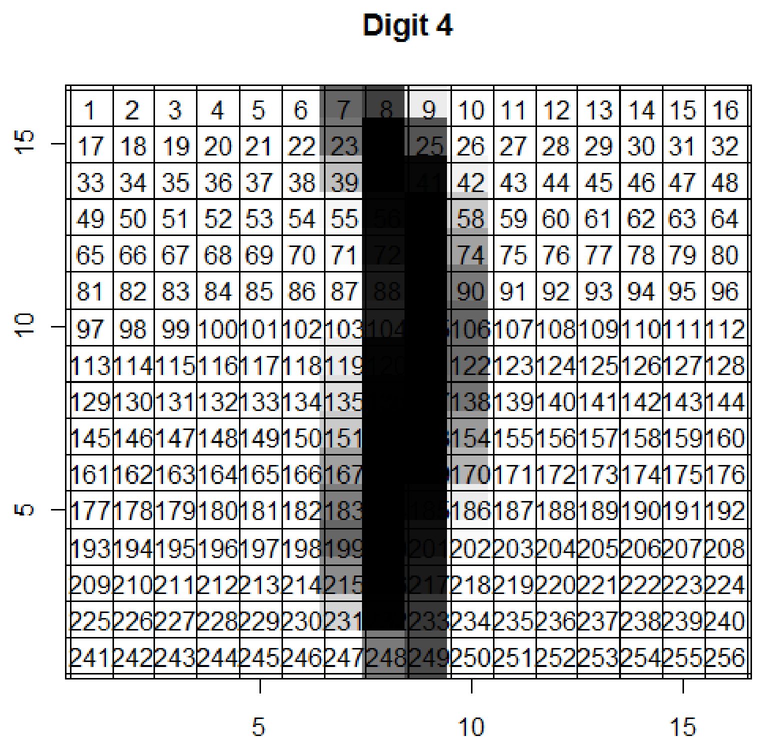

Figure 3.3: This observation is coded as a four in the training data but the SVM model classifies it as a one

The digit above is coded as a four in the training data but it clearly does not resemble a four. It is reassuring to see that the SVM algorithm does not sacrifice out-of-sample prediction by trying to overfit this in-sample observation. This demonstrates the value of relaxing the linear separability assumption and including the slack parameter ξ when fitting an SVM model.

CHAPTER 4

IMPUTATION

4.1 Categorical Data Imputation Using SVMs

An important new area of applicability for SVMs is in the imputation of missing data. Almost all data sets have missing values, especially social science data and data coming from surveys. One simplistic way of dealing with the problem of missing data is to delete any observation with a missing value. The problem with this approach is that even with a modest rate of missingness, a researcher may end up having to throw away sizable portions of his or her data, which obviously affects any inferences made using the data. The deletion of missing data is particularly egregious if the data are not missing-at-random, as the observations being deleted are systematically different from the complete observations and thus inferences based on the reduced data set will be necessarily biased.

Several approaches have been taken to impute missing values in order to avoid the deletion of observations. As imputation is not the core focus of this paper, not all of these methods will be discussed. Early methods for imputing missing data values included simply substituting the mean in for the missing value, or the "hot deck" approach where another observation with similar characteristics is randomly selected to replace the observation with the missing value. One of the most popular algorithms for imputing missing data is the expectation maximization (EM) algorithm (Dempster et al. 1977). Although EM is a more general algorithm, when it is referred to henceforth in this paper it will be understood to be the method that calculates the maximum likelihood estimator for the missing values using multivariate regression. One problem with this implementation of the EM algorithm is that it assumes multivariate normality; it is not intended to impute categorical values (in fact the algorithm will not even run if the categorical variables are not coded numerically). A method for imputing missing values using SVMs will be proposed and demonstrated in the following sections.

4.2 The SVM Imputation (SVMI) Algorithm

The idea behind SVMI is fairly simple. SVM models are tuned and fit on each of the categorical variables in the data set using the observations that do not contain any missing values. Subsequently, initial guesses are generated for each of the missing values in the entire data set. Finally, the algorithm then performs SVM prediction for the missing values of the categorical variables, followed by multivariate regression to impute the missing values of the continuous variables. This process iterates until some convergence criteria is met. The algorithm is formally laid out in the boxed figure on the next page. SVMI was written in the statistical computing package R and the code can be found in the appendix.

In SVMI, convergence is defined as

$$\left[\sum_{i=1}^{n} \sum_{j \in \text{cat.vars.}} \mathcal{I}_{\{\text{old}_{i,j} \neq \text{new}_{i,j}\}} + \sum_{i=1}^{n} \sum_{j \in \text{cts.vars.}} \text{abs}(\text{old}_{i,j} - \text{new}_{i,j})/s_j\right] < \epsilon$$

where $\text{old}_{i,j}$ is the entry in the i^{th} row and the j^{th} column of the data matrix in the previous iteration, $\text{new}_{i,j}$ is the entry in the i^{th} row and the j^{th} column of the data matrix in the current iteration, and s_j is the sample standard deviation of the j^{th} column of the data matrix in the previous iteration. Convergence is chosen in this way so that the difference in the non-imputed values will be zero, the penalty for changing the imputation of a categorical variable is 1, and the penalty for changing the imputation of a continuous value is the scaled absolute difference between the old value and the new value. The algorithm iterates until the sum of these penalties falls below some user-defined tolerance level, ϵ. Since ϵ will always be less than one, this ensures that the algorithm will not converge until all of the imputed categorical values have stopped changing from iteration to iteration.

SVM Imputation Algorithm (SVMI)

1. Initialize the algorithm providing the following objects:
    a) A data matrix with missing values, preferably with the continuous variables pre-transformed to normality using Box-Cox transformations (Box and Cox 1964)
    b) A list of pre-tuned and fitted SVM models for each of the categorical variables that contain missing values; the training data for these models are the observations in the data that contain no missing values
    c) A vector indicating the column numbers of the categorical variables
    d) A tolerance threshold for convergence and maximum permitted iterations; the defaults are 1×10^{-5} and 100, respectively
2. Impute the missing values for any observation that has only one single missing value on a categorical variable.
3. Fill in all of the remaining missing values with an initial guess; for continuous variables use the mean. For categorical variables, use the mode.
4. Begin Loop:
    1) Cycle through each of the categorical variables column by column imputing the missing values using the previously fitted SVM models
    2) Impute the missing values of the continuous variables using multivariate regression
    3) Check for convergence
    4) If convergence criteria is met, break from loop; if not, repeat loop
5. Return the complete imputed dataset.

4.3 The NLSY97 Data

The data on which SVMI will be demonstrated come from a national longitudinal survey conducted by the Bureau of Labor Statistics (BLS) and can be obtained at the URL `<http://www.bls.gov/nls/nlsy97.htm>`. The data set is called NLSY97 and it is described on the BLS web page as follows.

> "The NLSY97 consists of a nationally representative sample of approximately 9,000 youths who were 12 to 16 years old as of December 31, 1996...The NLSY97 is designed to document the transition from school to work and into adulthood. It collects extensive information about youths' labor market behavior and educational experiences over time...Aside from educational and labor market experiences, the NLSY97 contains detailed information on many other topics. Subject areas in the questionnaire include: Youths' relationships with parents, contact with absent parents, marital and fertility histories, dating, sexual activity, onset of puberty, training, participation in government assistance programs, expectations, time use, criminal behavior, and alcohol and drug use."

There are literally thousands of variables in the NLSY97 survey data, but for this analysis, only 15 variables will be used. The reason for the limited scope is that this imputation question was motivated by a BYU public policy master's student who is using these particular variables to build a logistic regression model for predicting whether or not a youth will graduate from high school based on a set of covariates.

Of the 9,000 youths initially enrolled in the survey, 8,983 were followed to the end of the study. Table 4.1 defines the 15 variables used in this analysis and Table 4.2 contains summary statistics for these variables.

Table 4.1: Description of NLSY97 Variables

Variable	Description
`absent`	# of days absent during the most recent fall term
`tchrgood`	My teacher is good: 1=strongly agree, 2=agree, 3=disagree, 4=strongly disagree
`gender`	0 = female, 1 = male
`ethnicity`	1 = Hispanic, 0 = not Hispanic
`race`	1=White, 2=Black, 3=Native American, 4=Asian/Pacific Islander, and 5=Other
`PTA`	How often parent participates in PTA: 1 = often, 2 = sometimes, 3 = never
`PVolClass`	How often parent volunteers in class: 1 = often, 2 = sometimes, 3 = never
`age`	Age in years
`region`	1 = Northeast, 2 = Central, 3 = South, 4 = West
`momeduc`	# of years of education of biological mother
`RaceEthn`	1 = Black, 2 = Hispanic, 3 = Mixed (non-Hispanic), 4 = non-Black/non-Hispanic
`asvab`	Armed Services Vocational Aptitude Battery math/verbal percentile
`hs20`	1 if youth graduated high school by age 20, 0 otherwise
`FamStruc97`	4=Two biol. parents, 3=Two parents, one biol., 2=One biol. parent only, 1=Other
`avginc10k`	Average income of household measured in ten thousand dollars

Table 4.2: Summary Statistics of NLSY97 Variables

Variable	Type	Min	Max	Mean	Mode	# Missing
`absent`	Continuous	0	200	4.89	0	297
`tchrgood`	Categorical	1	4	1.98	2	19
`gender`	Indicator	0	1	0.51	1	0
`ethnicity`	Indicator	0	1	0.21	0	24
`race`	Categorical	1	5	1.81	1	80
`PTA`	Categorical	1	3	2.05	2	1322
`PVolClass`	Categorical	1	3	2.37	3	1322
`age`	Continuous	12	18	14.35	15	0
`Region`	Categorical	1	4	2.64	3	0
`momeduc`	Continuous	1	20	12.44	12	694
`RaceEthnicity`	Categorical	1	4	2.79	4	0
`asvab`	Continuous	0	100	45.31	–	1891
`hs20`	Indicator	0	1	0.86	1	1422
`FamStructure97`	Categorical	1	4	3.06	4	31
`avginc10k`	Continuous	0	42.56	5.08	–	289

4.4 Demonstration of SVMI and the EM Algorithm

There are 4,979 observations in the NLSY97 data set that contain no missing values. Since these values are all known, they will be referred to as the test data matrix. Using the data described in section 4.3, the performance of SVMI, as well as a naive application of the EM algorithm, will be demonstrated under three different scenarios which will be referred to as cases A, B, and C, respectively. In case A, values in the test data matrix will be deleted to mimic the missingness in the full data set. In case B, values will again be deleted in a way that attempts to recreate the multivariate missingness of the full data but at a much higher rate. This will be done in order to see how sensitive the algorithm is to higher rates of missingness. Finally, in case C, values in the test data matrix will be deleted univariately in order to preserve the column by column rates of missingness in the full data but without regard for any multivariate relationships in the missingness. In each of these cases, SVMI and the EM algorithm will be applied, and then the imputed values from the two methods will be compared to the true values in order to investigate the success of the algorithms in correctly imputing the missing values.

Case A: *Mimic the Missingness of the Full Data*

In attempting to mimic the missingness of the full data set, the following method was used. First a matrix of equal dimension to the full data matrix was created containing a 1 anywhere there was a missing value and 0's everywhere else. After doing this it was possible to calculate a correlation matrix of this missing table. This correlation matrix describes the joint missingness of the various variables of the data. For each row of the test data a row from this missing matrix was randomly sampled and values were set to missing in the test data at the same places as the rows sampled from the missing matrix. In this way the multivariate missingness of the full data was preserved. In the full data set 4,979/8,983 = 55.4% of the rows did not have missing values. In the test data set, after performing this procedure to recreate the missingness of the full data set, 2,775/4,979 = 55.7% of the rows were complete. To determine whether the multivariate rates of missingness were mimicked successfully, the correlation matrices of the missing matrix of the full data set as well as that of the test data set were examined and found to be quite similar. Table 4.3 compares the column-by-column rates of missingness of the full and test data sets, illustrating their similarity.

Table 4.3: Missingness in Case A

	Rate Missing Full Data	Rate Missing Case A Data
`absent`	0.0331	0.0337
`tchrgood`	0.0021	0.0024
`gender`	0.0000	0.0000
`ethnicity`	0.0027	0.0014
`race`	0.0089	0.0121
`PTA`	0.1472	0.1549
`PVolClass`	0.1472	0.1547
`age`	0.0000	0.0000
`Region`	0.0000	0.0000
`momeduc`	0.0773	0.0779
`RaceEthnicity`	0.0000	0.0000
`asvab`	0.2105	0.2113
`hs20`	0.1583	0.1497
`FamStructure97`	0.0035	0.0024
`avginc10k`	0.0322	0.0299

Once the values are set to missing, it is possible to pass the case A data through SVMI and verify how well it did at imputing the missing values. Different metrics are used to assess the performance of the algorithm depending on whether the variable is continuous or categorical. If the variable is categorical, the metric to assess performance is simply the number of correctly imputed values divided by the number of values that were imputed. If the variable is continuous, the metric used is called average scaled error (ASE) and is calculated as follows:

$$\text{ASE}_j = \sum_i \frac{|true_{ij} - imputed_{ij}|/s_j}{n}$$

where $i = 1,...,n$, n is the number of imputed values on the j^{th} continuous variable, and s_j is the sample standard deviation of the true values of variable j. Dividing the errors by their standard deviations in this way "normalizes" the error rates and allows us to compare ASE across variables.

For case A these metrics were calculated for each variable after applying the SVMI algorithm. These metrics were also calculated for the variables after applying the EM algorithm. It should be noted that the imputed values of the categorical variables returned by the EM algorithm have been rounded to the nearest integer value in order to compare them to SVMI. The results are reported in Table 4.4 below. The variables `gender`, `age`, `Region`, and `RaceEthnicity` are not reported since there were no values imputed for these variables.

Table 4.4: Comparison of Imputation Techniques Under Case A

Variable	Missing Rate	% Correct SVMI	% Correct EM	ASE SVMI	ASE EM
`tchrgood`	0.0024	0.7500	0.7500	–	–
`ethnicity`	0.0014	1.0000	1.0000	–	–
`race`	0.0121	0.8667	0.7667	–	–
`PTA`	0.1549	0.4293	0.4462	–	–
`PVolClass`	0.1547	0.5208	0.4532	–	–
`hs20`	0.1497	0.8899	0.8899	–	–
`FamStructure97`	0.0024	0.5833	0.5833	–	–
`absent`	0.0337	–	–	0.7375	0.7345
`momeduc`	0.0779	–	–	0.6686	0.6790
`asvab`	0.2113	–	–	0.6551	0.6534
`avginc10k`	0.0299	–	–	0.6674	0.6567

As the table demonstrates, SVMI performs about as well or better for every categorical variable. This is reassuring since the objective of the algorithm is to improve the imputation for the categorical variables. For the continuous variables, the two imputation methods performed quite similarly, which makes sense since both algorithms are using iterative multivariate regression to impute the missing values. It is also reassuring that in performing the SVM imputation on the categorical variables, the imputation of the continuous variables has not been disrupted in some way.

Case B*: Increased Missingness*

For case B, an approach similar to case A was taken. As in case A, the rows of the missing matrix were randomly drawn in order to determine where to delete values, but in case B, rows that had missing values were sampled at a higher rate of frequency in order to increase the missingness. As a result, case B ended up with 1,260/4,979 = 25.3% complete rows as opposed to 55.7% as in case A. Just as in case A, the correlation matrices of the full and case B missing value data sets were compared and found to be similar. This indicates that the multivariate dependencies of the missingness have been preserved. The rates of missingness in the case B data are provided in Table 4.5 below along with the rates of missingness of the full data demonstrating that for each variable the rates of missingness for case B are significantly higher than the rates of missingness in the original data.

Table 4.5: Missingness in Case B

	Rate Missing Full Data	Rate Missing Case B Data
`absent`	0.0331	0.0588
`tchrgood`	0.0021	0.0036
`gender`	0.0000	0.0000
`ethnicity`	0.0027	0.0052
`race`	0.0089	0.0133
`PTA`	0.1472	0.2511
`PVolClass`	0.1472	0.2511
`age`	0.0000	0.0000
`Region`	0.0000	0.0000
`momeduc`	0.0773	0.1187
`RaceEthnicity`	0.0000	0.0000
`asvab`	0.2105	0.3591
`hs20`	0.1583	0.2617
`FamStructure97`	0.0035	0.0070
`avginc10k`	0.0322	0.0510

The SVMI and EM algorithms were applied to the case B data to impute the missing values. The performance of the imputations is reported in Table 4.6 below.

Table 4.6: Comparison of Imputation Techniques Under Case B

Variable	Missing Rate	% Correct SVMI	% Correct EM	ASE SVMI	ASE EM
`tchrgood`	0.0036	0.5556	0.5556	–	–
`ethnicity`	0.0052	0.9231	0.8846	–	–
`race`	0.0133	0.8788	0.8333	–	–
`PTA`	0.2511	0.4448	0.4464	–	–
`PVolClass`	0.2511	0.5088	0.4416	–	–
`hs20`	0.2617	0.8918	0.8895	–	–
`FamStructure97`	0.0070	0.6857	0.3714	–	–
`absent`	0.0588	–	–	0.7933	0.7907
`momeduc`	0.1187	–	–	0.7012	0.6953
`asvab`	0.3591	–	–	0.6734	0.6750
`avginc10k`	0.0510	–	–	0.6412	0.6404

SVMI performs as well or better for each of the categorical variables but here, with the higher rates of missingness, the improvement in using SVMI over EM becomes more pronounced. Again, SVMI performs almost identically to the EM algorithm for all four continuous variables.

Case C*: Mimicking Column-by-Column Missingness*

In case C, only the column-by-column missingness of the full data was preserved. The column-by-column rates of missingness of the full data were calculated by counting the missing values and dividing that number by the number of rows, 8,983. By deleting values in this way, it is expected that the overall number of observations that have missing values will actually be greater than before, as the variable-by-variable dependencies in missingness have not been captured. Indeed, this is the case: there are now only 2,019/4,979 = 40.55% of the rows that are complete as opposed to 55.7% in case A. The column-by-column rates of missingness are reported in Table 4.7, demonstrating that these rates have been successfully imitated.

Table 4.7: Missingness in Case C

	Rate Missing Full Data	Rate Missing Case C Data
`absent`	0.0331	0.0329
`tchrgood`	0.0021	0.0026
`gender`	0.0000	0.0000
`ethnicity`	0.0027	0.0030
`race`	0.0089	0.0108
`PTA`	0.1472	0.1476
`PVolClass`	0.1472	0.1488
`age`	0.0000	0.0000
`Region`	0.0000	0.0000
`momeduc`	0.0773	0.0717
`RaceEthnicity`	0.0000	0.0000
`asvab`	0.2105	0.2085
`hs20`	0.1583	0.1639
`FamStructure97`	0.0035	0.0032
`avginc10k`	0.0322	0.0299

The case C data, with the values missing as described above, were passed through the SVMI and EM algorithms, and the performance of the imputations was calculated as before. These values are reported in Table 4.8.

Table 4.8: Comparison of Imputation Techniques Under Case C

Variable	Missing Rate	% Correct SVMI	% Correct EM	ASE SVMI	ASE EM
`tchrgood`	0.0026	1.0000	1.0000	–	–
`ethnicity`	0.0030	0.9333	0.8000	–	–
`race`	0.0108	0.8889	0.7778	–	–
`PTA`	0.1476	0.5048	0.4939	–	–
`PVolClass`	0.1488	0.5425	0.5277	–	–
`hs20`	0.1639	0.8897	0.8897	–	–
`FamStructure97`	0.0032	0.7500	0.1875	–	–
`absent`	0.0329	–	–	0.7718	0.7799
`momeduc`	0.0717	–	–	0.6784	0.6819
`asvab`	0.2085	–	–	0.6821	0.6799
`avginc10k`	0.0299	–	–	0.6602	0.6734

Once again, SVMI does as well or much better than the EM algorithm at imputing the missing values for the categorical variables but performs essentially the same as the EM algorithm on the continuous variables. These results are not surprising, as they affirm what was shown in cases A and B. It is interesting to note that, when the data are missing more at random, the rates of correct prediction seem to be marginally higher across the board. Observing the three cases together a few trends surface.

1. It was expected that the EM algorithm would perform almost as well as SVMI for the ordinal categorical variables. In this data set all of the categorical variables can be viewed as ordinal or loosely ordinal except `race`, `ethnicity` and `FamStructure97`. These exceptions are where SVMI outperforms the EM algorithm by the widest margin on average.

2. SVMI outperforms the EM algorithm in the imputation of categorical values but does not improve the imputation of the continuous variables in any demonstrable way. Reassuringly, SVMI does not worsen the imputation of the continuous variables.

3. Variables that have little correlation with other variables in the data set, such as `PTA`, `PVolClass`, and `FamStructure97`, are imputed correctly much less often than variables that do correlate highly with others. This was to be expected.

4. The more normally distributed the continuous variable, the lower the ASE for imputing that variable. The variables `momeduc`, `asvab`, and `avginc10k` were all much closer to normality than `absent`, even after the Box-Cox transformations, and ASE is lower for these continuous variables in each of the three cases.

5. Correctly imputing the data does not seem to be very sensitive to whether the data are missing at random for this particular data set.

4.5 Imputation on the Full NLSY97 Data Set

Now that the performance of SVMI has been demonstrated in a controlled setting, and has been shown to work tolerably well, it can be used to impute the missing values for the entire NLSY97 data matrix. In this application, since the true values are unknown, there is no sure way to know how well it does at imputing the missing values. One thing that can be done, though, is to look at the summary statistics of the different variables to ensure that they are not changing wildly and that none of the imputations are falling outside the realm of possible values.

When the full NLSY97 data matrix with all 8,983 observations was first passed to SVMI, the algorithm failed to converge. This was in stark contrast to cases A, B and C above, where it converged quickly in about 15 iterations. Looking at a printout of the tolerance values from iteration to iteration revealed what was happening.

```
Iteration    Tolerance
    1      1000.0000000
    2       286.1494980
    3        77.6847922
    4        23.9519856
    5         8.1626274
    6         3.1243638
    7         1.4055197
    8         0.9071106
    9         0.7153577
   10         0.6816969
    ⋮             ⋮
   25         0.6718298
   26         0.6718275
   27         0.6718297
   28         0.6718275
   29         0.6718297
   30         0.6718275
```

After 25 iterations the algorithm was falling into an oscillating pattern where at each iteration it would switch back and forth between values, thus prohibiting it from converging. It seems the algorithm had encountered some kind of bi-modality in the data. Investigating this phenomenon further, two outliers were detected. These were observations 6,887 and

8,633. After removing these two observations and re-running SVMI on the full data, the algorithm was able to converge in 22 iterations. The summary statistics of the full data matrix, including the imputed values, are reported below in Table 4.9.

Table 4.9: Summary Statistics of NLSY97 Post-Imputation

	Min	Max	Mean	Mode	Missing
`absent`	0	200	4.84	0	0
`tchrgood`	1	4	1.98	2	0
`gender`	0	1	0.51	1	0
`ethnicity`	0	1	0.21	0	0
`race`	1	5	1.82	1	0
`PTA`	1	3	2.10	2	0
`PVolClass`	1	3	2.43	3	0
`age`	12	18	14.35	15	0
`Region`	1	4	2.64	3	0
`momeduc`	1	20	12.39	12	0
`RaceEthnicity`	1	4	2.79	4.00	0
`asvab`	0	100.8	43.35	–	0
`hs20`	0	1	0.88	1	0
`FamStructure97`	1	4	3.06	4	0
`avginc10k`	0	42.56	5.06	–	0

The values reported in Table 4.9 can be compared to those found in Table 4.2 in Section 4.3. There are a couple of things to note looking at these two tables of summary statistics. The first is that the means change very slightly from pre- to post-imputation, and the second is that `asvab` now has a maximum value that is marginally outside the range of possible values. These two issues will be addressed in turn.

Dealing with the slight change in means, in this situation (survey data) the missing values are assuredly *not* missing at random. Thus, one would expect the means to change a little after imputation. One thing to note is that the changes in the means seem to make sense. Logically, one would expect a greater number of missing values to come from respondents who have negative answers to the questions. For example, if a respondent's mother did not graduate high school the respondent is more likely to not answer a question about their mother's education out of embarrassment. Another example would be that someone who has an unusually high number of absences might also be difficult to track down for the survey. After imputation of the NLSY97 data, the means of all of the imputed variables shift slightly in the direction that makes sense based on a logical assessment of respondent psyche. The slight shift in means exhibited here seems to indicate heuristically that the imputation is working well.

With regards to the out of bounds imputation on `asvab`, it turns out that there is only one imputed value that falls above 100. The first thing to remember is that the method of imputation for continuous variables in SVMI is multivariate regression. Multivariate regression assumes that the responses are normally distributed and, thus, defined on the entire real line. Looking at the observation in question the values of the other covariates seem to explain why it was given such an unusually high imputed value. For that observation, the respondent is a white female from a two biological parent household, whose mother has a PhD level of education and whose parents' income is about $200,000 a year. All of these values are correlated with higher scores on the armed services aptitude battery, hence, it makes sense that her imputed value was very high. In practice this value can be rounded back to 100 in order to keep it within the range of acceptable values. This high value is another instance indicating that SVMI is yielding imputed values that make sense in the context of the data.

CHAPTER 5

CONCLUSION

In this paper the history and technical details surrounding support vector machines have been presented. Using the U.S. zip code data, a classic data set from the machine learning literature, it has been demonstrated that SVMs perform very well relative to competing methods in terms of out-of-sample prediction error rates. Finally, an new algorithm for imputing categorical as well as continuous values has been proposed and constructed. This algorithm has been shown to perform as well or better than the EM algorithm under three different scenarios of missingness.

BIBLIOGRAPHY

Abe, S. (2005), *Support Vector Machines*, Advances in Pattern Recognition, New York, NY: Springer.

Aizerman, M., Braverman, E., and Rozonoer, L. (1964), "Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning," *Automation and Remote Control*, 25, 821–837.

Aronszajn, N. (1950), "Theory of Reproducing Kernels," *Transactions of the American Mathematical Society*, 68, 337–404.

Bennett, K., and Mangasarian, O. (1992), "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets," *Optimization Methods and Software*, 1, 23–34.

Boser, B., Guyon, I., and Vapnik, V. (1992), "A Training Algorithm for Optimal Margin Classifiers," ACM, Pittsburgh: ACM Press, no. 5 in Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, pp. 144–152.

Box, G., and Cox, D. (1964), "An Analysis of Transformations," *Journal of the Royal Statistical Society*, 26(2), 211–252.

Chang, C.-C., and Lin, C.-J. (2011), "LIBSVM: a Library for Support Vector Machines," Found at <www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>.

Christianini, N., and Shawe-Taylor, J. (2000), *An Introduction to Support Vector Machines*, Cambridge, UK: Cambridge University Press, pp. 156–157.

—— (2004), *Kernel Methods for Pattern Analysis*, Cambridge, UK: Cambridge University Press.

Cortes, C., and Vapnik, V. (1995), "Support Vector Networks," *Machine Learning*, 20, 273–297.

Cover, T. (1965), "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Transactions on Electronic Computers*, 14, 326–334.

Dempster, A., Laird, M., and Rubin, D. (1977), "Maximum Likelihood From Incomplete Data Via the EM Algorithm," *Journal of the Royal Statistical Society*, B-39, 1–38.

Duan, K.-B., and Keerthi, S. (2005), "Which is the Best Multiclass SVM Method? An Empirical Study," .

Fisher, R. (1936), "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, 7, 111–132.

Hamel, L. (2009), *Knowledge Discovery With Support Vector Machines*, Wiley Series on Methods and Applications in Data Mining, Hoboken, NJ: John Wiley and Sons Inc.

Hassoun, M. (1986), "Logical Signal Processing with Optically Connected Threshold Gates," Phd dissertation, Wayne State University, Department of Electrical and Computer Engineering, Detroit, MI.

Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2003), "A Practical Guide to Support Vector Classification," Technical report, National Taiwan University, Department of Computer Science, Taipei, Taiwan.

Hsu, C.-W., and Lin, C.-J. (2002), "A Comparison of Methods for Multiclass Support Vector Machines," *IEEE Transactions on Neural Networks*, 13, 425–425.

Jiang, B., Zhang, X., and Cai, T. (2008), "Estimating the Confidence Interval for Prediction Errors of Support Vector Machine Classifiers," *Journal of Machine Learning Research*, 9, 521–540.

Mangasarian, O. (1965), “Linear and Nonlinear Separation of Patterns by Linear Programming,” *Operations Research*, 13, 444–452.

—— (2000), “Generalized Support Vector Machines,” in *Advances in Large Margin Classifiers*, eds. Smola, A., Bartlett, P., Schölkpf, B., and Schuurmans, D., Cambridge, MA: MIT Press, pp. 135–146.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (2007), *Numerical Recipes: The Art of Scientific Computing*, New York, NY: Cambridge University Press, section16.5, 3rd ed., pp. 883–898.

Rosenblatt, F. (1962), *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanics*, Washington D.C.: Spartan Books.

Schölkpf, B., and Smola, A. (2002), *Learning with Kernels*, Adaptive Compuation and Machine Learning, Cambridge, MA: MIT Press.

Smith, F. (1968), “Pattern Classifier Design by Linear Programming,” *IEEE Transactions on Computers*, C-17, 367–372.

Steinwart, I., and Christmann, A. (2008), *Support Vector Machines*, Information Science and Statistics, New York, NY: Springer.

Vapnik, V., and Chervonenkis, A. (1974), *Theory of Pattern Recognition: Statistical Problems of Learning*, Moscow, Russia: Nauka.

Vapnik, V., and Lerner, A. (1963), “Pattern Recognition Using Generalized Portraits,” *Automation and Remote Control*, 24, 709–715.

Wu, T.-F., Lin, C.-J., and Weng, R. C. (2004), “Probability Estimates for Multi-class Classification by Pairwise Coupling,” *Journal of Machine Learning Research*, 5, 975–1005.

Xie, Y. (2007), “An Introduction to Support Vector Machines and Implementation in R,” `<http://yihui.name/cv/images/SVM_Report_Yihui.pdf>`.

APPENDICES

APPENDIX A

R CODE

A.1 R Code for Zip Code Example

```
rm(list=ls())
library('e1071')

traindat<- read.table("E:/Masters Project/zip.train.txt",sep="",header=F);
dig0<- traindat[which(traindat[,1]==0),-1]
dig1<- traindat[which(traindat[,1]==1),-1]
dig2<- traindat[which(traindat[,1]==2),-1]
dig3<- traindat[which(traindat[,1]==3),-1]
dig4<- traindat[which(traindat[,1]==4),-1]
dig5<- traindat[which(traindat[,1]==5),-1]
dig6<- traindat[which(traindat[,1]==6),-1]
dig7<- traindat[which(traindat[,1]==7),-1]
dig8<- traindat[which(traindat[,1]==8),-1]
dig9<- traindat[which(traindat[,1]==9),-1]


##### Digit Plotter Function ###########
digit.plot<- function(vec256,num,numson=F){
    tranvec<- (vec256+1)/(max(vec256)+1)
    tranvec<- -1*(tranvec-1)
    xpts<- matrix(nrow=16,ncol=16)
    ypts<- matrix(nrow=16,ncol=16)
    for(i in 1:16){
        xpts[,i]<- rep(i,16)
        ypts[i,]<- rep(17-i,16)
    }
    plot(as.vector(t(xpts)),as.vector(t(ypts)),type="p",cex=4.4,pch=15,
        col=gray(tranvec),xlab="",ylab="",main=paste("Digit",num))
    abline(v=seq(from=-.5,to=16.5,by=1))
    abline(h=seq(from=-.5,to=16.5,by=1))
    if(numson){
        nums <- matrix(1:256,nrow=16,ncol=16,byrow=T)
        for(j in 16:1) {
            for(i in 1:16){
                text(i,j,labels=paste(nums[17-j,i]))
            }
        }
    }
}
```

```
digit.plot(traindat[1,-1],6,numson=T)

par(mfrow=c(2,5))
for(i in 0:9){
    digit.plot(apply(traindat[which(traindat[,1]==i),-1],2,mean),num=i)
}
mtext("Plot of Average Values for Each Digit",line=-1.4, outer=T)
par(mfrow=c(1,1))

############ Classification of 9s and 4s ###################
y49<-as.factor(c(rep(4,nrow(dig4)),rep(9,nrow(dig9))))

digdat49<-rbind(dig4,dig9)

out.tuner2<- tune.svm(x=digdat49,y=y49,gamma=1/(256*(seq(.1,4,by=.1))^2),cost=10^2,scale=F)
out.tuner2b<- tune.svm(y=y49,x=digdat49,kernel="polynomial",gamma=1/256,
    coef0=c(0,5,10,50,100,1000),degree=seq(1,6,by=1),cost=10,scale=F)

out2<- svm(y=y49,x=digdat49,gamma=0.003,cost=100,scale=F)
out2b<- svm(y=y49,x=digdat49,kernel="polynomial",gamma=1/256,coef0=0,degree=4,cost=10,scale=F)

## In sample prediction ##
mean(predict(out2b)==y49)

## Out of sample prediction ###
vals4<- predict(out2b,newdata=test4)
vals9<- predict(out2,newdata=test9)
mean(predict(out2,newdata=rbind(test4,test9))==c(rep(4,nrow(test4)),rep(9,nrow(test9))))
table(vals4)
table(vals9)

####### Classification of All of the Digits using SVM ##############
traindat<- read.table("E:/666/Final Project/zip.train.txt",sep="",header=F);
digmat<- traindat[,-1]
resp<- as.factor(traindat[,1])

ziptuner<- tune.svm(x=digmat,y=resp,gamma=seq(.005,.01,by=.001),cost=100,scale=F)
ziptuner2<- tune.svm(x=digmat,y=resp,kernel="polynomial",degree=seq(1,6,by=1),
    gamma=1/256,cost=c(1,10,100),scale=F)
zipmod<- svm(y=resp,x=digmat,gamma=.009,cost=100,scale=F)
zipmod2<- svm(y=resp,x=digmat,kernel="polynomial",degree=4,gamma=1/256,cost=100,scale=F)

## In sample prediction ##
1-mean(predict(zipmod)==resp)
1-mean(predict(zipmod2)==resp)

## Out of sample prediction ##
1-mean(predict(zipmod,newdata=testdat[,-1])==as.factor(testdat[,1]))
1-mean(predict(zipmod2,newdata=testdat[,-1])==as.factor(testdat[,1]))
```

A.2 R Code for Imputation Algorithm

```r
SVMI<- function(data,categ.vars,modlist,max.iter=100,min.tol=1e-4) {
    #categ.vars is a vector indicating the column numbers of the categorical variables
    #modlist should be a list containing the prefitted SVM models for each of the categ.vars

    nomiss<- which(apply(1*(is.na(data)),2,sum)==0)
    cts.vars<- c(1:ncol(data))[-c(categ.vars,nomiss)]
    cts.mat<- NULL
    for(i in 1:length(cts.vars)){
        cts.mat<- rbind(cts.mat,cbind(t(combn(cts.vars,i)),matrix(0,
            ncol=length(cts.vars)-i,nrow=choose(length(cts.vars),i))))
    }

    # this is a subroutine that will be called later when performing the cts imputation
    hat.fun<- function(datmat,cts.list=cts.mat){
        datmat[,categ.vars]<- apply(as.matrix(datmat[,categ.vars]),2,factor)
        hat.list<- list()
        for(i in 1:nrow(cts.list)){
            X<- suppressWarnings(model.matrix(~.,data=datmat[,-cts.list[i,]]))
            y<- as.matrix(datmat[rownames(X),cts.list[i,]])
            hat.list[[i]]<- X%*%(solve(t(X)%*%X)%*%t(X)%*%y)
        }
        return(hat.list)
    }

    #The following function performs the continuous imputation step on one row
    replacefun<- function(vec,hat.mat){
        # Performs multivariate imputation on a single row of the data matrix
        rownum<- vec[1]
        vec<- vec[-1]
        nposs<- ncol(cts.mat)
        nnmiss<- length(which(is.na(vec)))
        misspot<- c(which(is.na(vec)),rep(0,nposs-nnmiss))
        hatelem<- vecfind(cts.mat,misspot)
        vec[misspot]<- hat.mat[[hatelem]][rownum,]
        return(vec)
    }

    ###### First We impute missing values where only one value on
    ###### an observation is missing and it is categorical

    oneinds<- which(apply(1*(is.na(data)),1,sum)==1)
    onemiss<- data[oneinds,]
    newinds<- as.numeric(rownames(na.omit(onemiss[,-categ.vars])))
    onemiss<- data[newinds,]
    nmiss<- apply(is.na(onemiss)[,categ.vars],2,sum)

    for(i in 1:length(categ.vars)){
        if(nmiss[i]>0){
            missvar<- onemiss[which(is.na(onemiss[,categ.vars[i]])),]
            data[as.numeric(rownames(missvar)),categ.vars[i]]<-
                as.numeric(as.character(predict(modlist[[i]],
                newdata=as.matrix(missvar[,-categ.vars[i]]))))
        }
    }

    ########### Filling in the missing values with an initial guess

    mu0<- apply(na.omit(data),2,mean)
    mu0[categ.vars]<- apply(na.omit(data[,categ.vars]),2,moder)

    misscols<- apply(is.na(data),2,sum)
    misscols<- which(misscols>0)
    pfilled<- data
    for(i in misscols){
        pfilled[which(is.na(pfilled[,i])),i]<- mu0[i]
    }

    for(i in 1:length(categ.vars)){
        pfilled[which(is.na(data[,categ.vars[i]])),categ.vars[i]]<-
            as.numeric(as.character(predict(modlist[[i]],newdata=
            as.matrix(pfilled[which(is.na(data[,categ.vars[i]])),-categ.vars[i]]))))
    }

    prefilled<- pfilled
    pfilled[,-categ.vars]<- data[,-categ.vars]

    ########## Perform the initial imputation #############
    cleanind<- which(apply(1*!is.na(pfilled),1,prod)==1)
    clean<- pfilled[cleanind,] # returns the rows that have no missing values
    dirtind<- which(apply(1*!is.na(pfilled),1,prod)==0)
    dirty<- pfilled[dirtind,] # returns the rows with missing values
    hat.mat0<- hat.fun(datmat=prefilled)

    filled<- t(apply(cbind(as.numeric(rownames(dirty)),dirty),MARGIN=1,
        FUN=replacefun,hat.mat=hat.mat0))
    # recombines the imputed rows with the clean rows
    juntos<- rbind(clean,filled)[order(as.numeric(rownames(rbind(clean,filled)))),]

    iter<- 1
    tol<- 1
    tols<- c(1000)
    ##### Enter the loop where imputation will be performed iteratively ####
    while((iter<max.iter) & (tol>min.tol)) {
        indys<- sample(length(categ.vars),size=length(categ.vars))
        for(i in indys){
            juntos[which(is.na(data[,categ.vars[i]])),categ.vars[i]]<-
                as.numeric(as.character(predict(modlist[[i]],newdata=
                as.matrix(juntos[which(is.na(data[,categ.vars[i]])),-categ.vars[i]]))))
        }
        clean<- juntos[cleanind,]
        dirty[,categ.vars]<- juntos[dirtind,categ.vars]

        hat.mat0<- hat.fun(datmat=juntos)
        fillednew<- t(apply(cbind(as.numeric(rownames(dirty)),dirty),MARGIN=1,
            FUN=replacefun,hat.mat=hat.mat0))
        juntosnew<- rbind(clean,fillednew)[order(as.numeric(rownames(rbind(clean,fillednew)))),]

        sds<- apply(juntos[,-categ.vars],2,sd)
        comp.cts<- t(t(juntos[,-categ.vars]-juntosnew[,-categ.vars])/sds)
        comp.categ<- 1*(juntos[,categ.vars]!=juntosnew[,categ.vars])
        tol<- sum(abs(cbind(comp.cts,comp.categ)))

        filled<- fillednew
        juntosold<- juntos
        juntos<- juntosnew
        iter=iter+1
        tols<- c(tols,tol)
    }
    if(iter==max.iter){
        cat("The algorithm failed to converge in",iter,"iterations\n")
        } else {cat("The algorithm converged in",iter,"iterations\n")}

    out<- list(newdata=juntos,iterations=iter,tolerance=tols)
    return(out)
}
```