Classification and Image Processing on MNSIT Data Set

Author: Andrew Halsaver, Brian Becker, & Farshad Chitchian

Math 503AB: Mathematical Modeling I & II

Course Numbers: 18817 & 18839

1 Abstract

To classify handwritten digit characters from the MNIST data set, we describe and utilize several popular implementations of four classification algorithms: K-nearest neighbors, random forests, polynomial-kernel support vector machines, and non-convolutional, deep neural networks. We analyze the accuracy and time spent training these algorithms at varying amounts of training observations to obtain an empirical evaluation of the classification methods' effectiveness. We also explore the effects that result from using principle components and several image-processing techniques.

2 Introduction

The problem of classification is perhaps one of the most widely studied in the data mining and machine learning communities. This problem has been studied by researchers from several disciplines over the last three decades. Applications of classification include a wide variety of problem domains such as text [7], multimedia [26], social networks [28], and biological data [6]. In particular, we focus on digital handwriting which has applications in postal mail sorting, bank check processing, and more.

In this project we will develop methods to automatically recognize handwriting. We will apply various classification algorithms to the MNIST data set of handwritten digit images [31]. We suspect that the relatively high dimension of our data (the 784 features or pixels) will have a significant impact on the training time of our classification methods. We will evaluate our models using various machine learning algorithm performance metrics to better understand how the classification algorithms perform under the "curse of high dimensionality" [16]. Afterwards, we plan to investigate what image pre-processing techniques and how principle component analysis (PCA) might improve our results.

3 Literature Review

Two of the perhaps well known empirical studies to examine the performance of different machine learning methods on the MNIST data set (amongst other data sets) appear to be STATLOG [22] and "An empirical Comparison of Supervised Learning Algorithms" [11]. These papers investigate the performance of different supervised learning algorithms. While both of these papers measure the performance of different classification algorithms such as SVMs, neural networks, logistic regression, naive Bayes, random forests, boosted trees, etc., they do not attempt to better understand the problems that large feature spaces present in classification problems. In [10], the same authors also explore these issues by evaluating each classification algorithm on data sets of increasing dimensionality. Further, in [15], Li Deng reports that the highest recorded classification accuracy of the MNIST data set is 99.73%, and is achieved by a *committee* [30] of of convolutional neural networks. The highest classification accuracy obtained by a *single* algorithm is obtained by a very large and deep convolutional neural nets with an overall testing accuracy of 99.65%.

To better handle the "curse of high dimensionality" [16], we look to image pre-processing techniques. In [24], many techniques are introduced that are very popular in unsupervised

learning methods. We will investigate the popular principle components analysis [3, 9, 24] methods, as well as hough-deskewing image preprocessing techniques [18].

4 Data Description

We use a version of the MNIST data set available at [4]. This includes two sets of data with one labeled the training set and the other being the official testing set. We immediately set aside the official, unlabeled testing set for later and mostly use it as a novel data set with which to make and submit predictions to Kaggle.com.

To summarize, the training data contains 42,000 gray-scale images of hand-drawn digits from zero through nine. Each image is twenty-eight by twenty-eight pixels for a total of 784 pixels. Thus, the training set csv file consists of 42,000 rows (images) and 785 (784 pixels and a one label) columns. For each image, every pixel has a single intensity value between 0 and 255, inclusive, with a higher number indicating a lighter pixel. The full description of the training and testing data can be found at [4]. The official testing set is nearly identical in format except that it is unlabeled, and so only contains 784 columns instead of 785.

5 Algorithm Descriptions

We choose to use to implement the K-nearest neighbors, random forests, support vector machines, and neural networks models to perform our digit classification task. We now present a high-level description of each of the models.

5.1 K-Nearest Neighbors

In pattern recognition, the K-nearest neighbors is a non-parametric classification method that defers most of the computational time until the actual prediction. It is typically regarded as one of the simpler classification and regression methods in machine learning.

Given a positive integer K and test observation x_0, the K-nearest neighbors classifier first identifies the K training observations that are *closest* to x_0 (in Euclidean distance), represented by N_0. It then estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j:

$$\Pr\left(Y = j | X = x_0\right) = \frac{1}{K} \sum_{i \in N_0} I\left(y_i = j\right).$$

K-NN then classifies the test observation x_0 to the class with the largest probability, utilizing Baye's rule [21]. Note that a very small value of K tends to produce an overly flexible decision boundary (which may cause over-fitting) while a larger value of K generates a more linear and less flexible decision boundary. Note that our implementation [1] opts to break classification ties at random and if there are ties for the Kth nearest neighbor then all candidates are included in the vote. It also automatically centers and scales the features.

5.2 Random Forests

Random forests models are built up from *tree-based* methods [21]. Tree-based methods for classification involve making a series of splitting rules to partition the feature space into J regions, $R_1, R_2, ..., R_J$. Then, for a test observation x_0, we predict that each observation belongs the most commonly occurring class of training observations in the region to which it belongs.

To decide where to make the binary tree splits, a natural metric would be the classification error rate. The classification error rate E is the fraction of training observations in each terminal node region that do not belong to the most common class. We can then express this as

$$E = 1 - \max_k \left(\hat{p}_{mk}\right),$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However, two better alternative metrics for assessing *node purity* for tree-growing turn out to be the *Gini index* G, and *cross-entropy* D [21]:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}\left(1 - \hat{p}_{mk}\right),$$

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log\, \hat{p}_{mk}.$$

Both of these metrics are numerically similar in that since $0 \leq \hat{p}_{mk} \leq 1$, both D and G will be near zero when all of the $\hat{p}_{mk}$ are close to zero or one.

Decision trees are generally strong in that they are easy to explain, can form non-linear decision boundaries, and can be displayed graphically [33]. However, they generally do not hold the same level of predictive accuracy as other algorithms. Two common approaches to strengthening the predictive power of decision tree algorithms involve *bagging* and *random forests* [17, 21].

The issue with decision trees is that they suffer from high variance; two tree models based on a two-way split of one training dataset would likely yield very different predictions. To counteract this, *bagging* (or aggregate bootstrapping) is a procedure that involves forming B separate prediction models on B different bootstrapped training data sets [21]. Each Bth bootstrapped training set produces a different Bth tree, which we then aggregate all the trees together to make a majority vote for each test observation. The overall prediction is the most commonly occurring class among the B predictions.

While bagging reduces variance, the *random forests* model further decreases variance by decorrelating the trees. When building each tree in a random forests model, each time a split is considered, only a random sample of m predictors are chosen as split candidates from the full set of p predictors. This process results in very different looking trees, however, averaging or voting across all trees serve to form a less variable and more reliable prediction.

5.3 Support Vector Machines

We first introduce the *maximal margin classifier*, which fits a hyperplane through two classes in the way that the separating hyperplane is furthest from the training observations [21]. With two classes of output $y_i \in \{-1, +1\}$ and n training observations $x_1, ..., x_n \in \mathbb{R}^p$, the maximal margin hyperplane expression $\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}$ satisfies

$$y_i \left(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}\right) > 0$$

for all $i = 1, ..., n$. To maximize the margin M is to really solve the optimization problem

$$\max_{\beta_0, \beta_1, ..., \beta_p} M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1, \text{and}$$

$$y_i \left(\beta_0 + \beta_1 x_{i1} ... + \beta_p x_{ip}\right) \geq M$$

for all $i = 1, ..., n$. We can then classify the test observation x^* based on the sign of our prediction function

$$f(x^*) = \beta_0 + \beta_1 x_1^* + ... + \beta_p x_p^*.$$

This method usually fails, however, as it requires the existence of a separating hyperplane to exactly separate the two training classes.

The *support vector classifier* is an augmented version of the *maximal margin classifier*. Instead of simply maximizing the distance of training observations to the separating hyperplane between two classes, the *support vector classifier* (also known as the *soft margin classifier*), opts to forgo a wide margin in favor of a greater robustness to individual observations. It allows for some training observations to fall on the wrong side of the margin or even on the wrong side of the hyperplane. We then wish to maximize the margin M, albeit with a few new conditions. Hence, the *support vector classifier* solves the optimization problem

$$\max_{\beta_0, ..., \beta_p, \epsilon_1, ..., \epsilon_n} M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i \left(\beta_0 + \beta_1 x_{i1} ... + \beta_p x_{ip}\right) \geq M \left(1 - \epsilon_i\right),$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

where C is a nonnegative tuning parameter and ϵ_i are the slack variables that allow individual observations to be on the wrong side of the hyperplane.

The *support vector machine* is then an extension of the *support vector classifier*. It attempts to fit non-linear decision boundaries by enlarging the feature space using *kernels*,

or generalizations of inner products of the form $K(x_i, x_i')$ [21]. One such type of support vector machine utilizes a *polynomial kernel* of degree d where

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d.$$

The resulting non-linear prediction function then has the form

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i),$$

with n parameters α_i (for each training observation) and S being the collection of support vector indices. Now we are able to construct non-linear decision boundaries between two classes utilizing a polynomial-based support vector machine.

To augment the limited binary-classification support vector machines model, we utilize the *one-vs-one approach* to solve multi-class classification tasks [29]. This means to solve a K-class classification problem it will be necessary to actually implement $\binom{K}{2}$ binary models that output class probabilities for each of the K classes. The class with the highest predicted probability will then be chosen as the prediction for that observation.

5.4 Deep Neural Networks

Neural Networks is known as a powerful classifier in the image-recognition and classification fields [12, 13, 19, 32]. The general neural networks architecture resembles the biological connections of brain neurons, hence the name [20].

The most basic type of neural network is the *perceptron*. The perceptron model is limited in that it is strictly a binary linear classifier [20]. This means that it is only able to classify between two classes and only if the data can be separated by a hyperplane. Unlike the *support vector classifier*, if the data are not exactly separable by hyperplane, the perceptron model will fail to converge to some meaningful decision boundary [20].

The perceptron is composed of an input layer and an output layer. With an p dimensional feature space, the perceptron will have p many input *neurons* that will collectively either activate the output neuron or not for each test observation. If we have $x_1, x_2, ..., x_N$ training observations, than we train the perceptron by feeding one observation through the network at a time, adjusting each neuron's weight w_j to better classify each training observation. Each weight $w_1, ..., w_p$ can be thought of as expressing the importance of each respective input to the output [20]. For a binary output $y_i \in \{0, 1\}$, we may express the output y_i in terms of the neurons and their weights:

$$y_i = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases},$$

where b, the bias, is a real number which acts as a threshold for the output. To build stronger classifiers we can then build networks of perceptrons, however these larger perceptron networks suffer from training difficulty; small changes in weights typically yield big

output differences. *Sigmoid neural networks* help to alleviate this issue by using the sigmoid function to create a continuous output for each sigmoid neurdon defined by

$$y_i = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}.$$

Next to train the sigmoid neural network model, we adjust and choose our weights w_i and bias b to minimize the cost function

$$C(w, b) = \frac{1}{2n} \sum_x \|y - f(x)\|^2,$$

where $f(x)$ is our model's output and y the true class [20]. We then utilize the *gradient descent* method to minimize C. For this method, we iteratively update our weights w and b according to the formulas

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k},$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Here, η is known as the learning rate [20] where a small value of η results in a longer computational time to converge, but an η that is too large may not converge at all [6].

In *Back Propagation Neural Networks* we define an error term δ_j^l in j^{th} neurons and l^{th} layer to find better $\frac{\partial C}{\partial w_k}$ and $\frac{\partial C}{\partial b_l}$ which helps us in minimizing the cost function [20]. The errors propagate backward through the layers so the gradients of the previous layers are a function of the errors and weights in the layer ahead of it [6].

5.5 Implementation Details

We use the `caret` package in R to implement our aforementioned K-nearest neighbors, random forests,support vector machines, and deep neural networks to solve our multinomial classification digit-recognition problem. We expect a properly trained (non-convolutional) deep neural network to obtain the highest accuracy and to achieve near state of the art accuracy of the convolutional deep neural networks [12, 13, 32].

6 Data Preprocessing

We can observe the first sixteen training examples in Figure 1. We may quickly observe some important details from these examples. First, we can see that a lot of the bordering pixels are always black. Next, a few of the depicted digits are skewed. That is, they are not aligned vertically as they ought to be (such as the top left image of the digit 1). We will attempt to use the hough transformation to hopefully correctly vertically align the digits correctly.

6.1 Feature Extraction

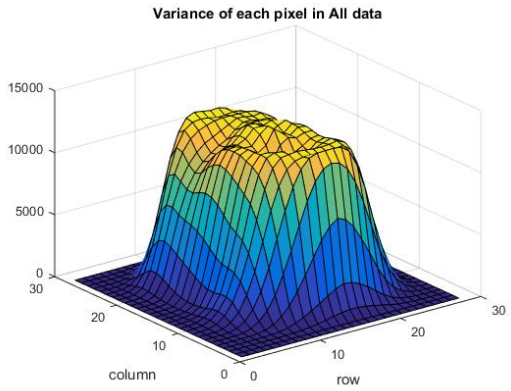


Figure 2: Variance of pixels in the entire training set

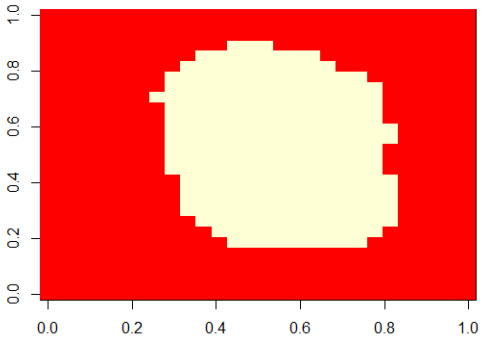


Figure 3: Extracted (high-variance) 252 pixels

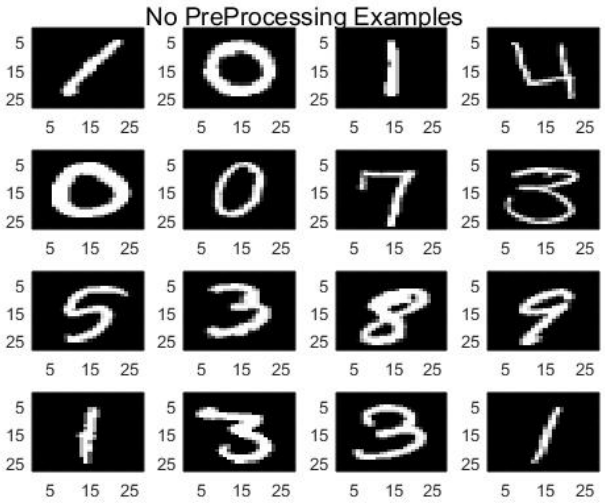


Figure 1: 28×28, centered & scaled training images.

Across all classes, most of the pixels near the image border are always black and thus do not vary much in pixel intensity. In fact, we measure the variance in the pixel intensity over the official training set and plot the results.

As we expect, the bordering pixels show little to no variance in intensity values. Since there is little variation across all classes, this means that these pixels hold very little predictive power [14]. To filter out zero-variance pixels and also remove pixels that have relatively very little variance (this second set of pixels are also likely to cause issues with scaling and centering our data later when calculating principle components), we will calculate two metrics that will serve as thresholds for filtering: the *frequency ratio* of the most prevalent value over the second most frequent value and *percent of unique values* [2] of the total number of samples. We search over all of the training data and chose to remove pixels that have both a higher *frequency ratio* than 95/5 and a lower percentage of *unique values* than 10%. This leaves us with a much smaller data set of only 252 pixels.

Next, we begin partitioning our (now 252-pixel) training data set into two smaller, disjoint subsets: a `TRAINbig` set and a `TESTing` set. The advantage of partitioning our labeled training set into a smaller `TRAINing` set and `TESTing` set is two-fold. First, we can easily

evaluate our models' performance by predicting on the labeled (with the correct answers) `TESTing` set. By comparing our predictions to the actual labels we can better understand what digits our models tend to label incorrectly. Secondly, our computing resources dictate that we cannot easily train any classification model on *all* of the official training set of forty-two thousand observations. We might as well use some of the official and labeled training set to better understand our algorithms' performance. We will perform tests on the official, unlabeled testing set and submit our predictions to the Kaggle website mostly as a novelty.

Now, by partitioning our `data` using an eighty-twenty percent split, we have a `TRAINbig` set of 33,604 observations and a `TESTing` set of 8,396 observations. Because we wish to quickly train our models and compare their performance against one another, we randomly sample five thousand observations from `TRAINbig` to form a `TRAINing` set with which we can more easily train different models and perform cross-validation within a reasonable amount of time.

6.2 Principle Components

We also wish to investigate how our models perform using principle components pre-processing. We perform principle components analysis on our 252 features which requires 91 principle components to capture ninety-five percent of the variability in the data. Plotting the first two principle components against each other helps illustrate how the different classes may become more separable in a smaller dimensional space. Comparing the two plots of figure 4 demonstrates the differences in class separation between the first two components and the last two.

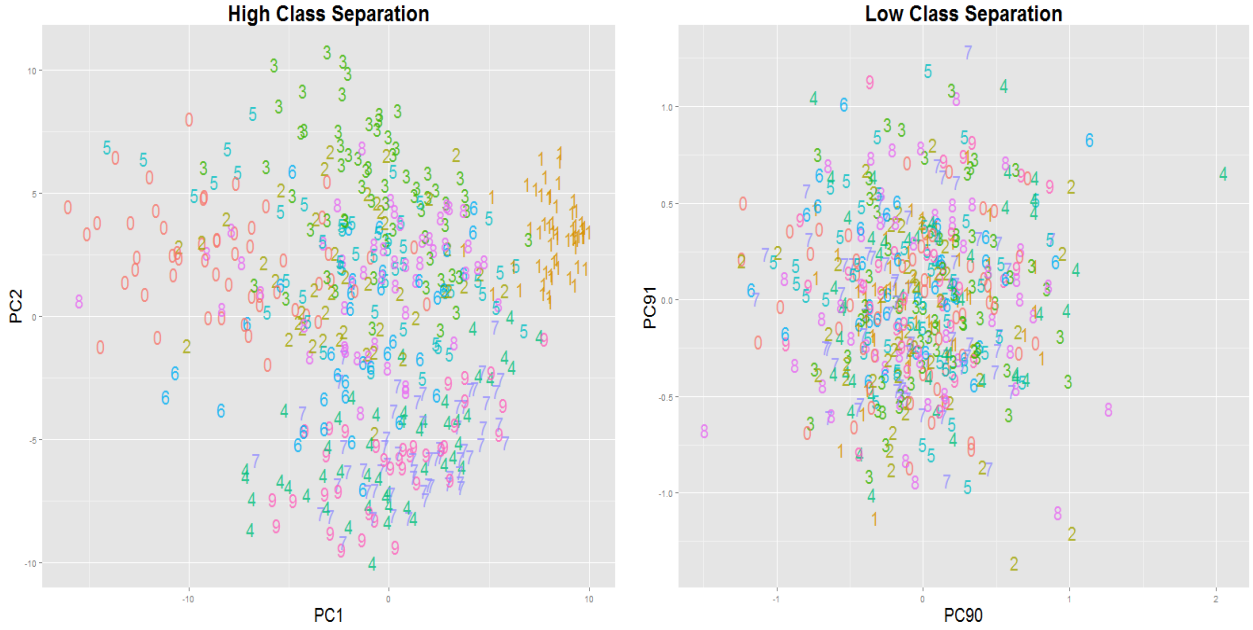


Figure 4: Principle Components Visualization

Using principle components analysis, we build the data sets `TRAINpca` and `TESTpca` of 5,000 and 8,396 observations, respectively. We anticipate that with the same amount of

observations as the `TRAINing` data set, the models trained with principle components should perform nearly as well as the 252-pixel data set, but much faster.

6.3 Image Processing

Other image pre-processing attempts have been useful in machine learning such as image segmentation to aid in the process of detecting cancer in breast tissue [27]. However, the images we have are very simple, and segmentation is not necessarily needed. Most MNIST data sets found online have been size-normalized and centered in a fixed-size image to provide a more efficient machine learning process, but there is always room for improvement. Our methods previously discussed are quite sufficient in the machine learning process. This does not mean we should not try other methods. We will discuss other approaches and, test whether or not these approaches will prove to be useful.

Thresholding and an edge detection method known as the Hough transform has been used in [18] (which are also byproducts of segmentation [25]). The features used in the classification process considers the geometric characteristics of the digits, such as the lines and circles seen in one of the digits, and the Hough transform is designed to find particular shapes in binary images [18, 25].

To detect lines, we consider a point (pixel) in a binary image, (x, y). Our goal to find every pair (a, b) that satisfies the equation of a line, $y = ax + b$. There are infinitely many lines to pass through (x, y), but only one will be found for particular pairs of (a, b). We plot the pairs of (a, b) in the "parameter space" or "accumulator array" [18, 25]. For example, let's consider the pixel $(x, y) = (1, 1)$. So the equation of a line is expressed as $1 = a \cdot 1 + b$, or simply $b = -a + 1$. Hence, the line $b = 1 - a$ has pairs of (a, b) relating to the single point $(1, 1)$ as shown in figure 5(a) [18, 25].

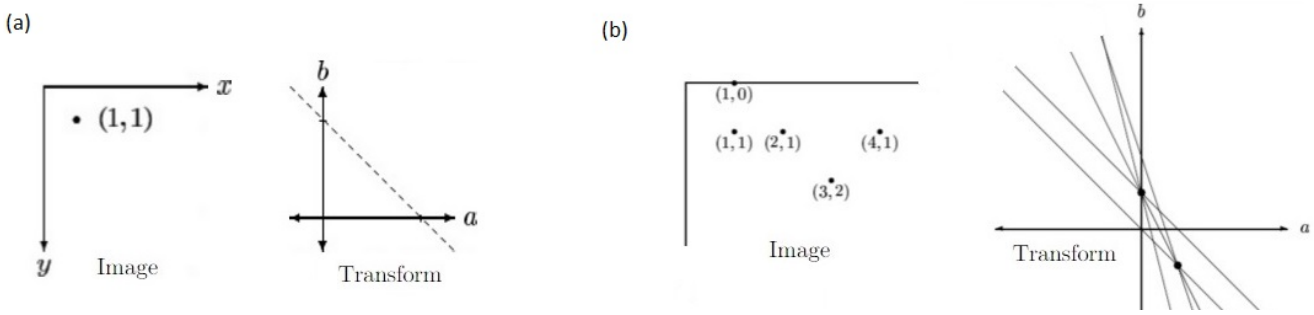


Figure 5: (a) A pixel $(1, 1)$ in the left coordinate space, and corresponding line in the parameter space to the right. (b) Five pixels in the left coordinate space, and corresponding lines in the parameter space to the right. In the transform, the dots tell us where there are the most intersections of lines (three in this case) [25].

When we consider several pixels in an image, we find the line corresponding to each pixel. Each pixel in the image is mapped onto a line in the parameter space indicating the pairs of (a, b). In this space, we look for the points with the greatest number of intersections (see figure 5(b)). The points in the parameter space correspond to the strongest line in the image. [25].

One of the issues encountered here is that $y = ax + b$ cannot be used to detect vertical lines (as that would mean the derivative is infinite). To resolve this issue, we turn to parametrization by mathematically describing lines by radius and angle relative to an origin (see figure 6) [18, 25].

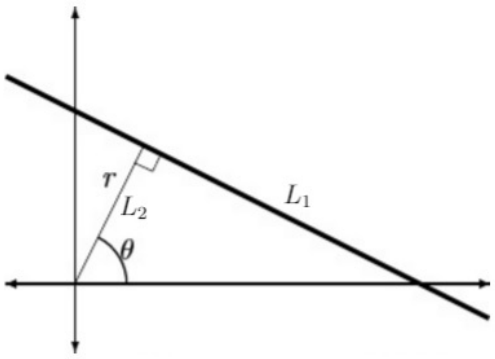


Figure 6: A line and its parameters [25].

Vertical lines are parametrized such that $\theta = 0$. Allowing r to have negative values will restrict θ such that $-\pi/2 < \theta \leq \pi/2$. Parametrizing yields the slope for any (x, y) point of a line as

$$\frac{y - r\sin\theta}{x - r\cos\theta}.$$

In figure 6, the line L_1 is perpendicular to line L_2 extending from the origin. In this, we can correctly define any line (such as L_1) by defining L_2 as a line protruding from the origin with radius r and angle θ relative to the x-axis. Since the slope of L_2 is now $\tan\theta$, the slope of L_1 is therefore

$$-(\tan\theta)^{-1} = -\frac{\cos\theta}{\sin\theta}.$$

That is, we now have

$$\frac{y - r\sin\theta}{x - r\cos\theta} = -\frac{\cos\theta}{\sin\theta}.$$

Using algebra, we acquire the equation for the line as

$$x\cos\theta + y\sin\theta = r.$$

By this we implement the Hough transform by choosing discrete r and θ values. Then, for (x, y), we compute $x\cos\theta + y\sin\theta$, for each θ and r (much like we did with the previous example). The values of (r, θ) with the highest values in the array will correspond to the strongest lines in our image [25]. Shown in figure 7(a) is an example output of what the detection looks like graphically, but the actual Hough transform looks like wavy lines as shown in figure 7(b) [18].

What is seen in figure 7(b) is the plot of what is called the accumulator array. The brighter areas are peaks, or high counts, indicating the number of points lying along the line with parameters (θ, r). In the accumulator array, the rows are designated by θ and the columns by r. As mentioned before, we seek the point which contains the most lines intersecting that point, and that is seen in the brighter regions of (b) [18, 25].

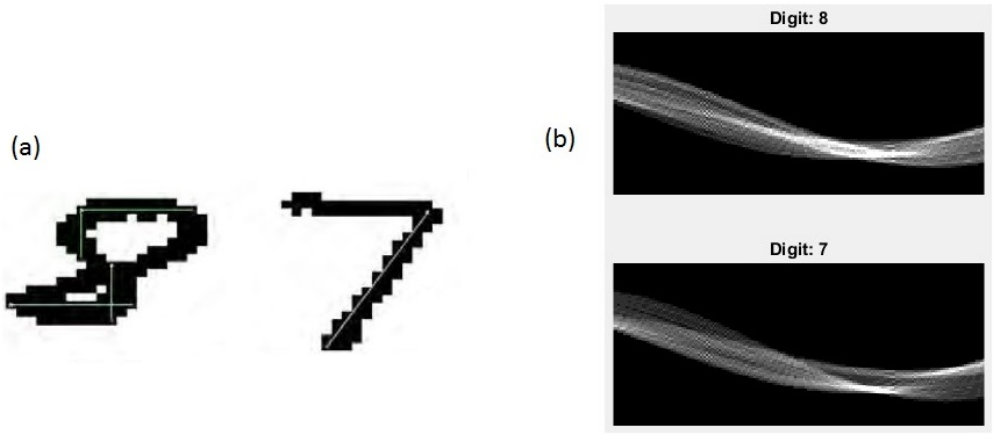


Figure 7

Now, the Hough transform can also be applied for detecting circles. Instead of using the equation for a line, we use the generic equation for a circle,

$$(x-a)^2 + (y-b)^2 = r^2.$$

Using polar coordinates, we can express a and b as

$$a = r\cos\theta - x$$
$$b = r\sin\theta - y.$$

The accumulator array in this case is 3-dimensional since we are now concerned with parameters a, b, and r. Unfortunately, the existence of a peak does not always mean a circle is present because we may not have a full circle. To remedy this, for every radius, we specify the amount of pixels required to obtain a circle, and this is done by using the circumference of a circle, $c = 2\pi r$ [18]. In [18], a circle is "detected" provided that at least 80% of the points composing the circumference are lying on the pixels.

Applying the Hough deskewing process to some of our training examples illustrates that some of the digits in Figures 6.3 have been properly aligned vertically (such as the ones digits) while some have not (such as the five digit). Contrast these images with those found in Figure 1.

6.4 Data Sets Summary

Utilizing our feature extraction along with principle components analysis, Hough transformations, and two different data set sizes leads us to a total of four different training data sets. The data sets and their relevant attributes are listed below in table 1. Note that separate testing sets were formed from `TESTing` for the `TRAINpca` and `TRAINh` data sets to assess their quality.

Note that we have only used our principle components and hough transformations of our data on the smaller data set of 5000 observations to keep our computational time low. In contrast, the `TRAINbig` data set contains eighty percent of the official training observations which we use to train more accurate models at the expense of very long computational times.

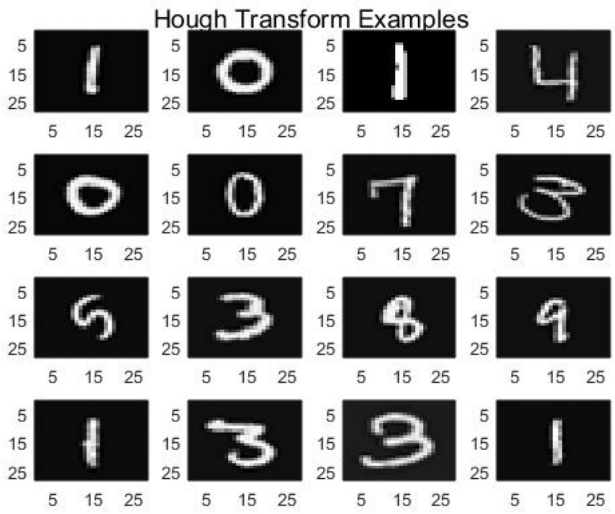


Figure 8: First 16 Hough-deskewed Examples

Table 1: Training Data Sets and Attributes

Data Set	**Features**	**Observations**	**Preprocess**
`TRAINing`	252	5000	none
`TRAINpca`	91	5000	PCA
`TRAINh`	453	5000	Hough
`TRAINbig`	252	33604	none

7 Model Building Methodology

To perform our digit-classification task, we make use of `caret`'s [5] implementation of the following supervised-learning algorithms: K-nearest neighbors, polynomial-kernel support vector machines, random forests, and neural networks. We then perform 5-fold cross validation on our `TRAINing` set with different combinations of each model's unique parameter values. We choose the set of parameters for each algorithm type that results in the highest cross-validation classification accuracy as our best model for each algorithm type. Finally, to obtain a real out of sample performance evaluation, we use the best model from each algorithm to to predict on the `TESTing` set.

Utilizing our best models from each algorithm type, we construct and plot a *confusion matrix* [23] and analyze the `TESTing` set accuracy and corresponding ninety-five percent accuracy confidence interval. The plotted confusion matrices allow us to quickly infer the `TESTing` set accuracy of our models. Here, a perfectly predicting model would produce a confusion matrix with all zeros outside the diagonal.

Next, we combine the prediction outputs from the above larger training set models to attempt to form a "stacked ensemble" support vector machine model [8], not unlike the committee of neural networks model which still has the highest recorded accuracy on the MNIST data set [30]. Using other, diverse models' prediction outputs as features for a new

model is known to usually increase accuracy [30].

8 Results

Our work results in implementing five different types of algorithms (four base algorithm types and one stacked ensemble) with five different data sets (four base data sets and one stacked data set). This yields a total of seventeen different model results where we measure the training time in minutes, the five-fold cross validation and testing set accuracies, and the testing set accuracy 95% confidence interval. The results are summarized in table 2.

Table 2: Model Results Summary

Alg	**Obs.**	**PreProcess**	**Train**	**Accuracy**		**95% C.I.**	
			time (m)	**5-Fold CV**	**Test Set**	**Low**	**High**
K-NN	5000	Sc&Cntr	0.31	0.9294	0.929	0.9233	0.9344
RF	5000	none	11.36	0.933	0.9366	0.9312	0.9418
SVM	5000	none	3.93	0.9526	0.9616	0.9573	0.9657
NNets	5000	none	35.34	0.9174	0.951	0.9462	0.9556
K-NN	5000	PCA	0.1742	0.9312	0.9341	0.9286	0.9393
RF	5000	PCA	4.42	0.9156	0.9115	0.9052	0.9175
SVM	5000	PCA	3.74	0.9362	0.9396	0.9343	0.9446
NNets	5000	PCA	50.03	0.9058	0.9393	0.9339	0.9443
K-NN	5000	Hough	2.53	0.906	0.9097	0.9034	0.9158
RF	5000	Hough	39.45	0.8666	0.888	0.8811	0.8947
SVM	5000	Hough	60.29	0.9228	0.9222	0.9163	0.9279
NNets	5000	Hough	37.89	0.8788	0.9269	0.9211	0.9323
K-NN	33604	Sc&Cntr	10.14	0.9635	0.9661	0.962	0.9698
RF	33604	none	143.66	0.9579	0.9611	0.9567	0.9651
SVM	33604	none	65.67	0.9782	0.9812	0.978	0.984
NNets	33604	none	182.26	0.9521	0.9758	0.9723	0.979
Ensemble	33604	none	415.51	0.9794	0.9877	0.9851	0.99

8.1 Data Set Analysis

It appears that our feature extraction method of only working with the 252 highest variance pixels does not appear to greatly hinder our testing accuracy. We are still achieving quite high accuracies with our different models using less than a third of the total amount of pixels while decreasing our algorithm training time greatly. Another important lesson is that while principle components analysis helped to "compress" our number of features from 252 to only 91, our algorithms did noticeably lose a few percentage points of accuracy. Also, while it appears that some researchers were able to achieve improved results with deskewing [15], our Hough deskewing transformations did not. Finally, not surprisingly, the largest increase of accuracy results from increasing our number of training images from 5000 to 33,604.

8.2 Model Analysis

While we expected a deep neural networks to perform best, instead of second best, we find the support vector machine model taking the top position in single algorithm. Surprisingly, the K-nearest neighbors model performs very well for how quickly it is trained (although it takes just as long to perform predictions on new data) compared to the other models. This suggests that the K-nearest neighbors model is likely a very good alternative to the support vector machine classifier when training time needs to be minimized. Finally, the stacked ensemble algorithm achieves our highest testing accuracy of 98.77% (which is quite good for only utilizing a third of the available pixels), albeit at the expense of a total of nearly seven hours of training time.

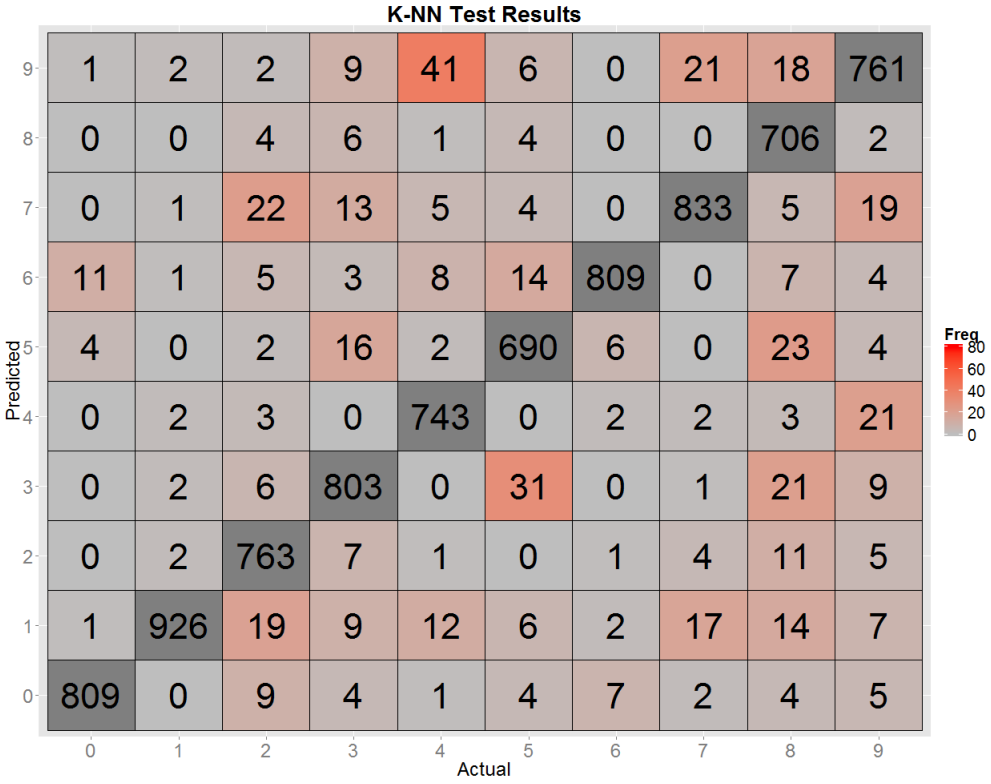
9 Conclusion

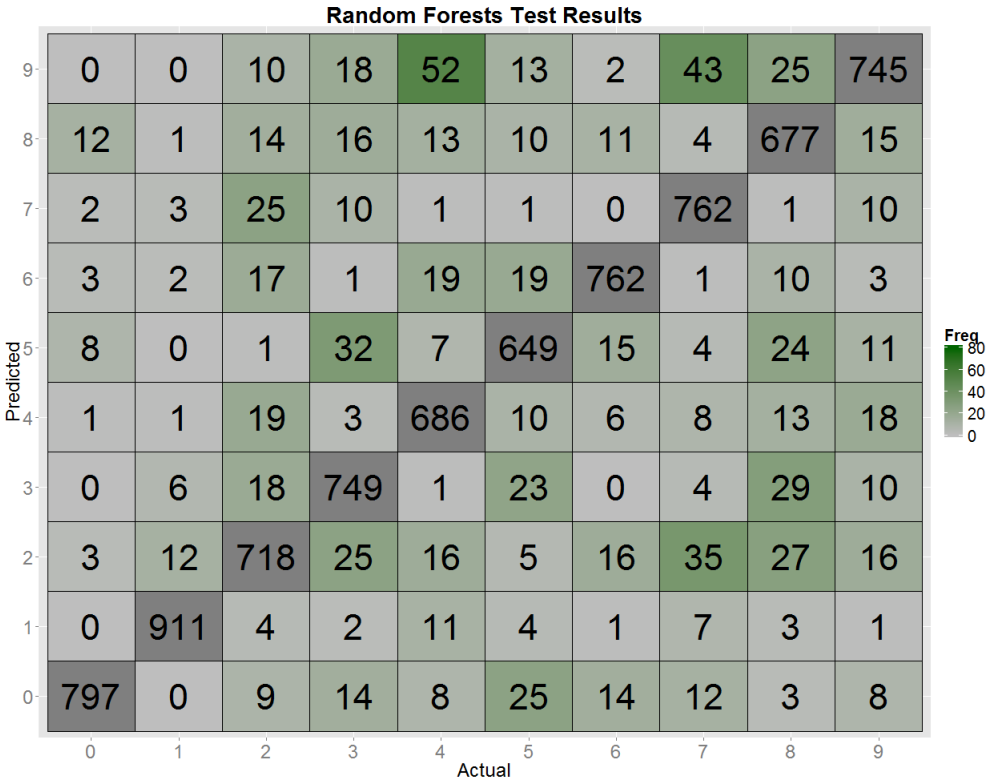
With these results, we are reminded of a few fundamental lessons in machine learning. The first is that increasing the number of training observations is usually the simplest and best way to increase classification accuracy. Using PCA, image preprocessing transformations, and other complex data compression techniques may not perform as well as a simple, intuitive feature extraction of the higher variance pixels. Finally, while our neural networks algorithm performed well, neural networks are typically more difficult to train due to their complex neural architecture [19] and the support vector machine appears to work as a better "out-of-the-box" classification algorithm. Finally, our stacked ensemble model is able to capture a slightly higher testing accuracy than the other models.

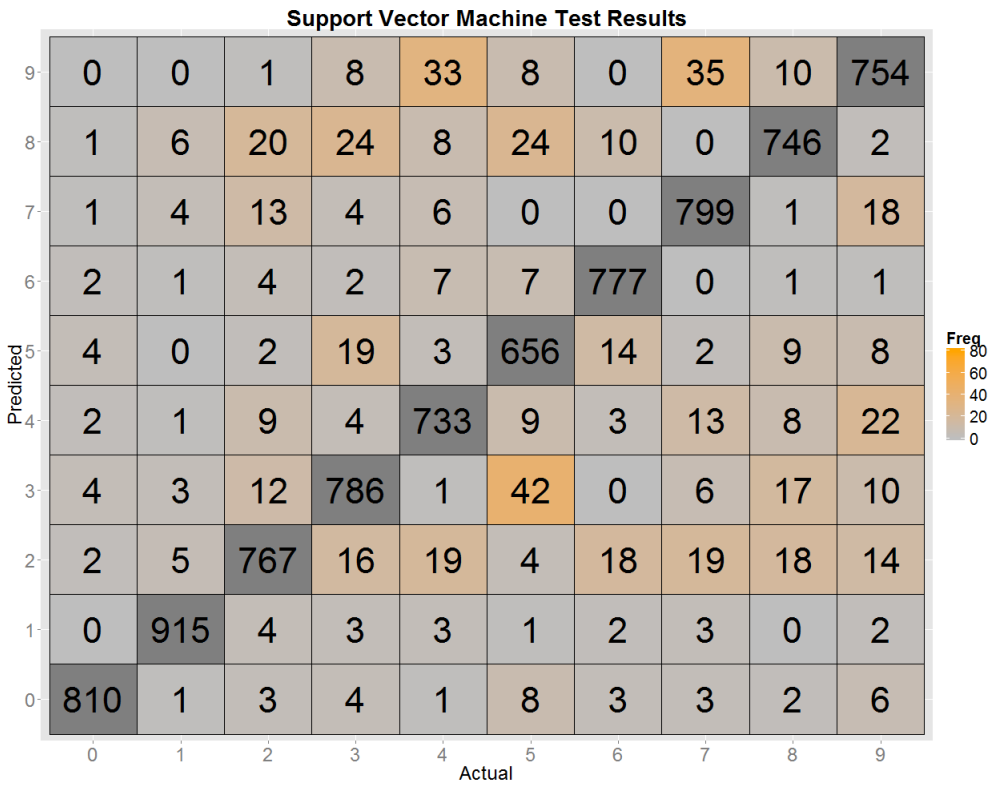
10 Appendix

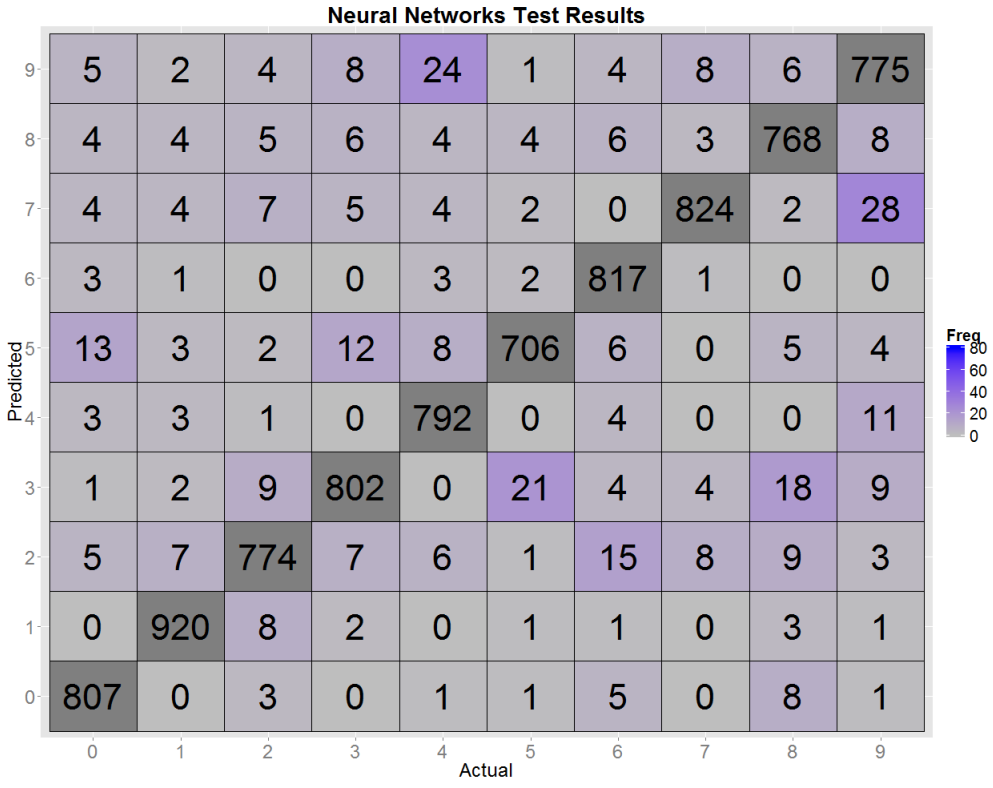
Here, we list the confusion matrix results for all 17 different models. The confusion matrices depict the **frequency** of predicted digits (vertical axis) vs actual digits (horizontal axis) in each tile. Better prediction results are indicated by smaller (ideally zero) numbers for the non diagonal tiles.

10.1 Small Training Set Confusion Matrices

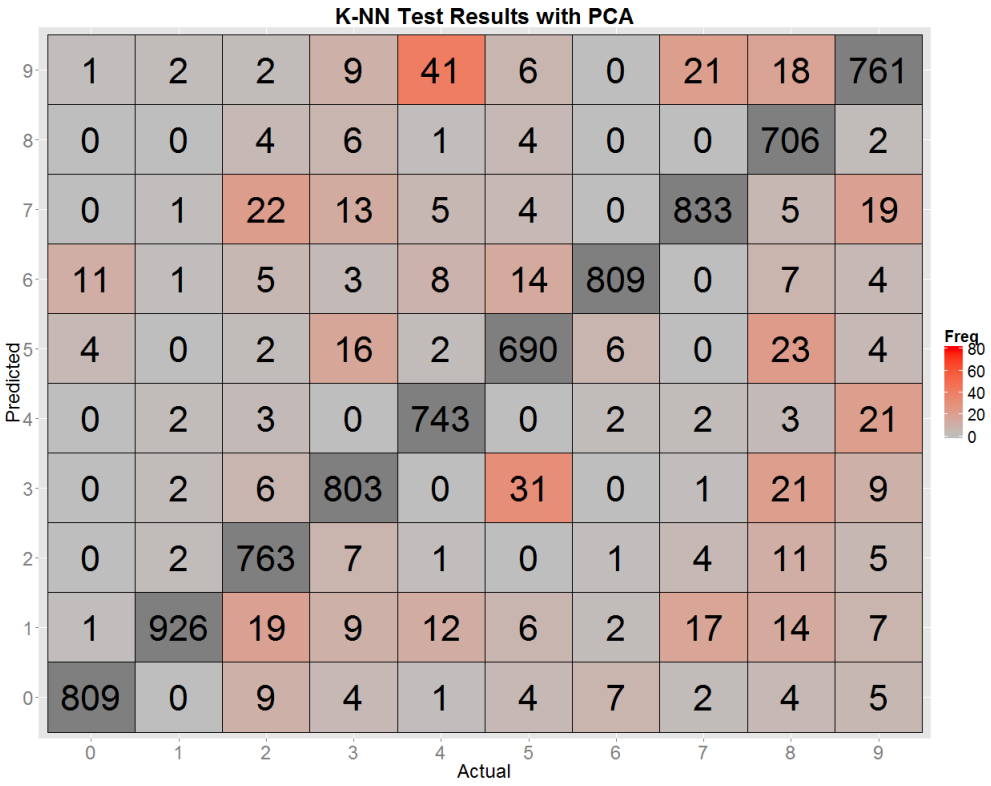


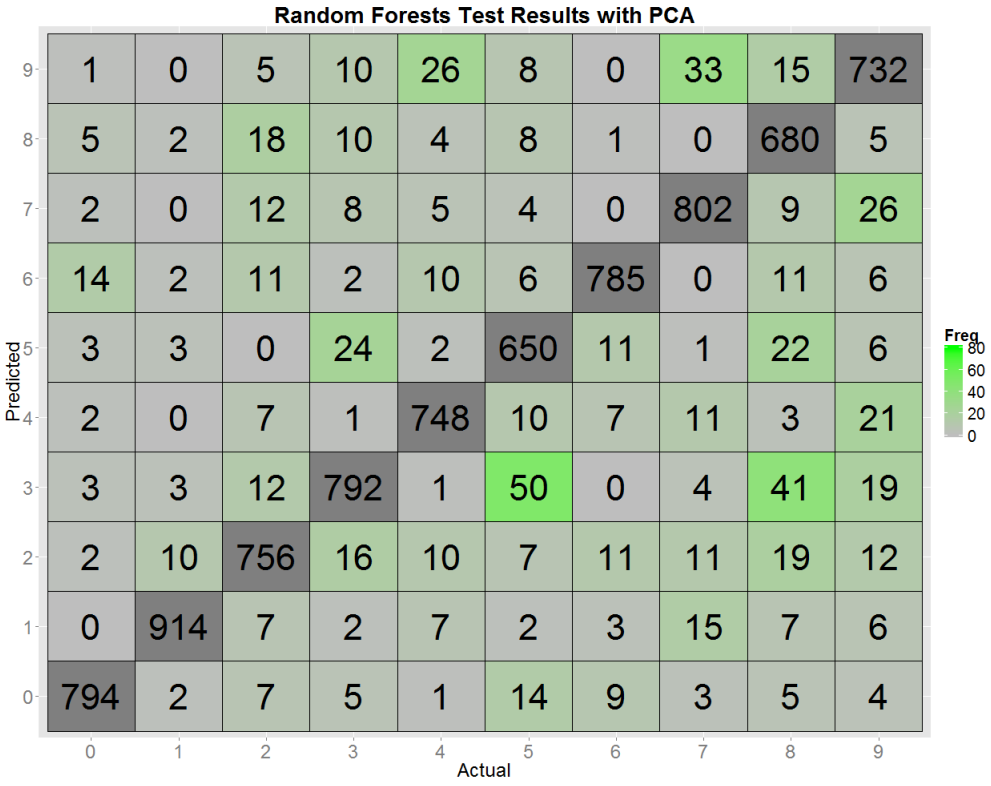


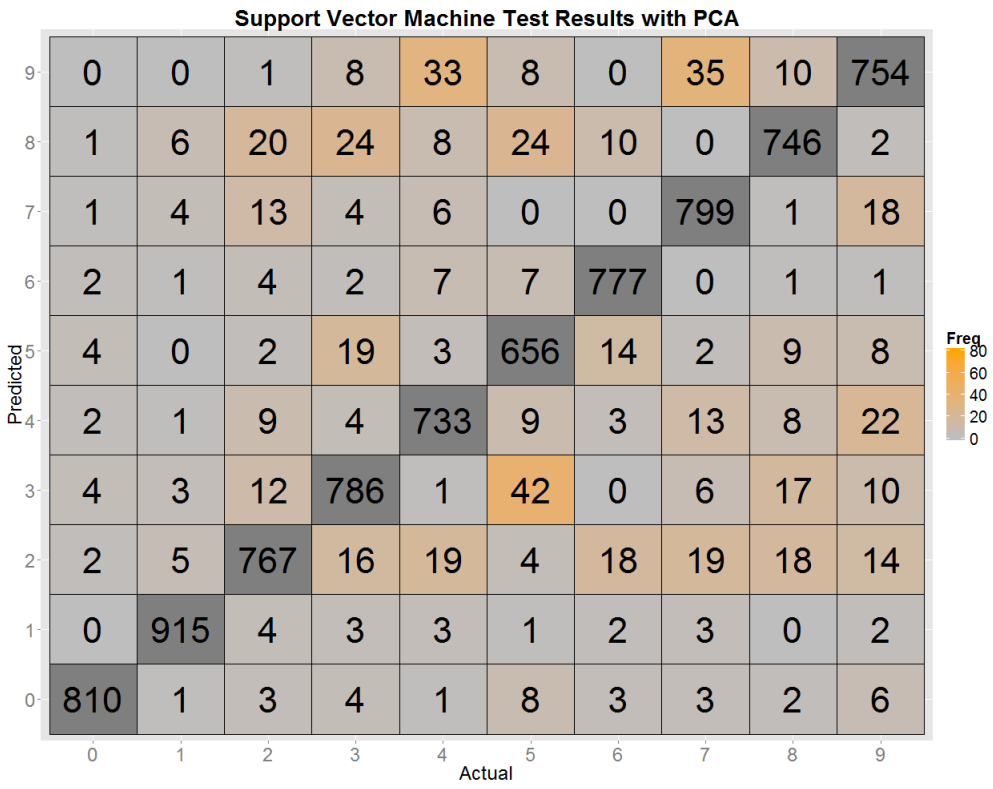

Support Vector Machine Test Results
Predicted
Actual
Freq
80
60
40
20
0
0 0 1 8 33 8 0 35 10 754
1 6 20 24 8 24 10 0 746 2
1 4 13 4 6 0 0 799 1 18
2 1 4 2 7 7 777 0 1 1
4 0 2 19 3 656 14 2 9 8
2 1 9 4 733 9 3 13 8 22
4 3 12 786 1 42 0 6 17 10
2 5 767 16 19 4 18 19 18 14
0 915 4 3 3 1 2 3 0 2
810 1 3 4 1 8 3 3 2 6

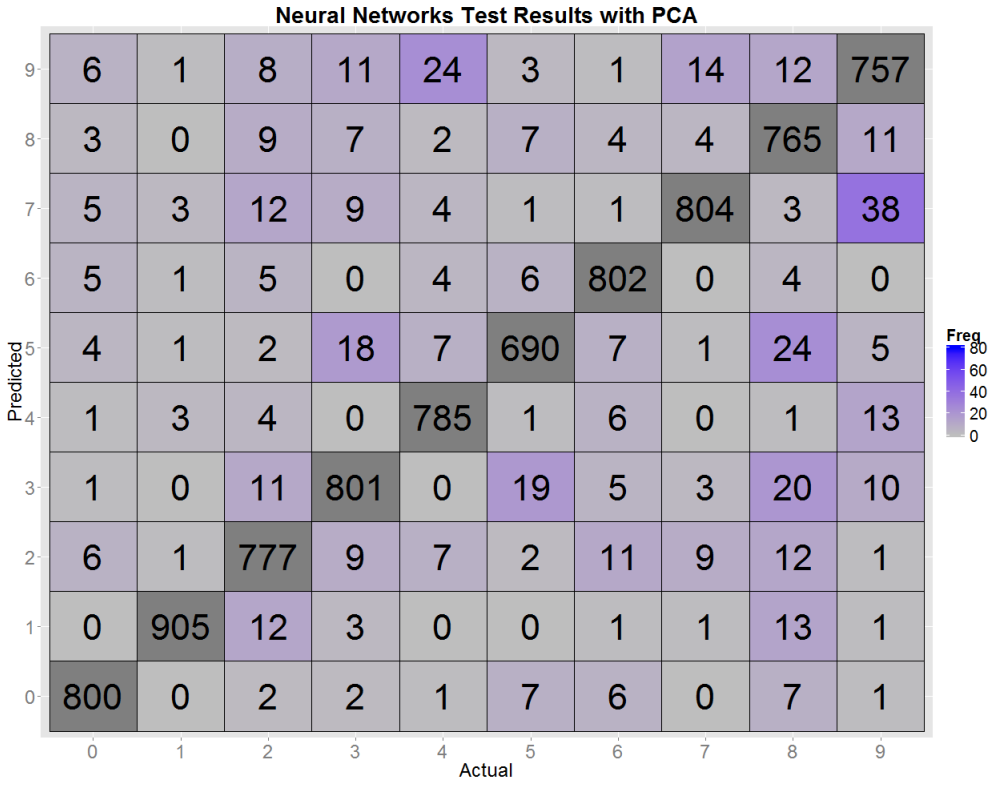

Neural Networks Test Results
Predicted
Actual
Freq
80
60
40
20
0
5 2 4 8 24 1 4 8 6 775
4 4 5 6 4 4 6 3 768 8
4 4 7 5 4 2 0 824 2 28
3 1 0 0 3 2 817 1 0 0
13 3 2 12 8 706 6 0 5 4
3 3 1 0 792 0 4 0 0 11
1 2 9 802 0 21 4 4 18 9
5 7 774 7 6 1 15 8 9 3
0 920 8 2 0 1 1 0 3 1
807 0 3 0 1 1 5 0 8 1

10.2 PCA Confusion Matrices

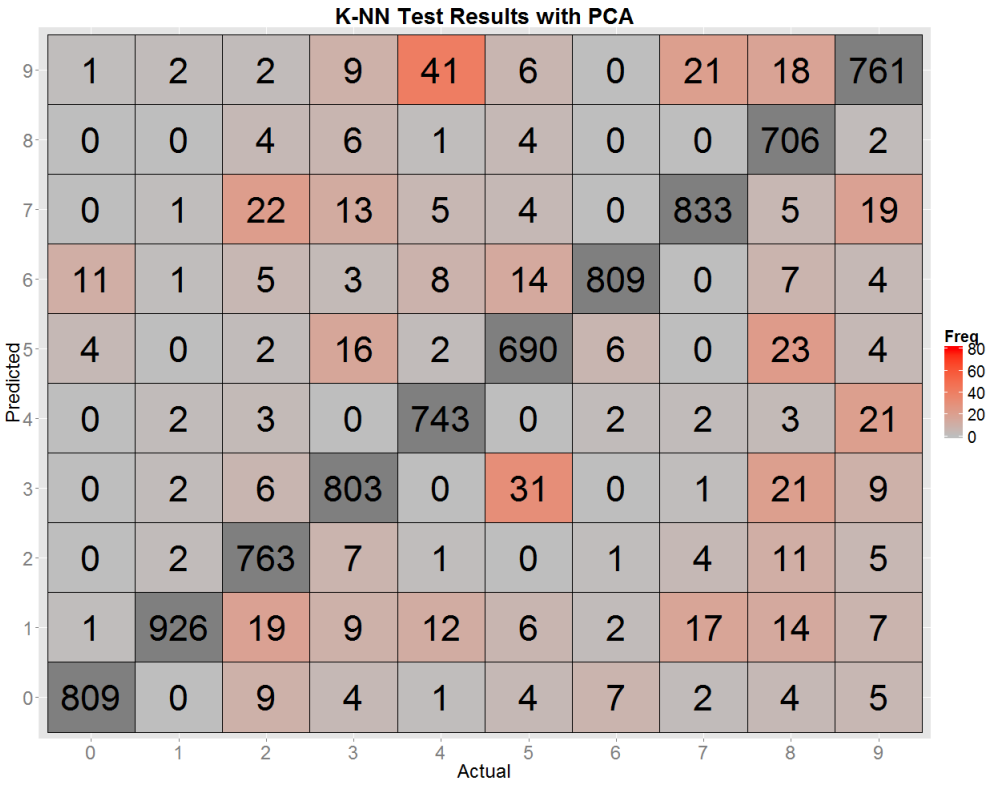


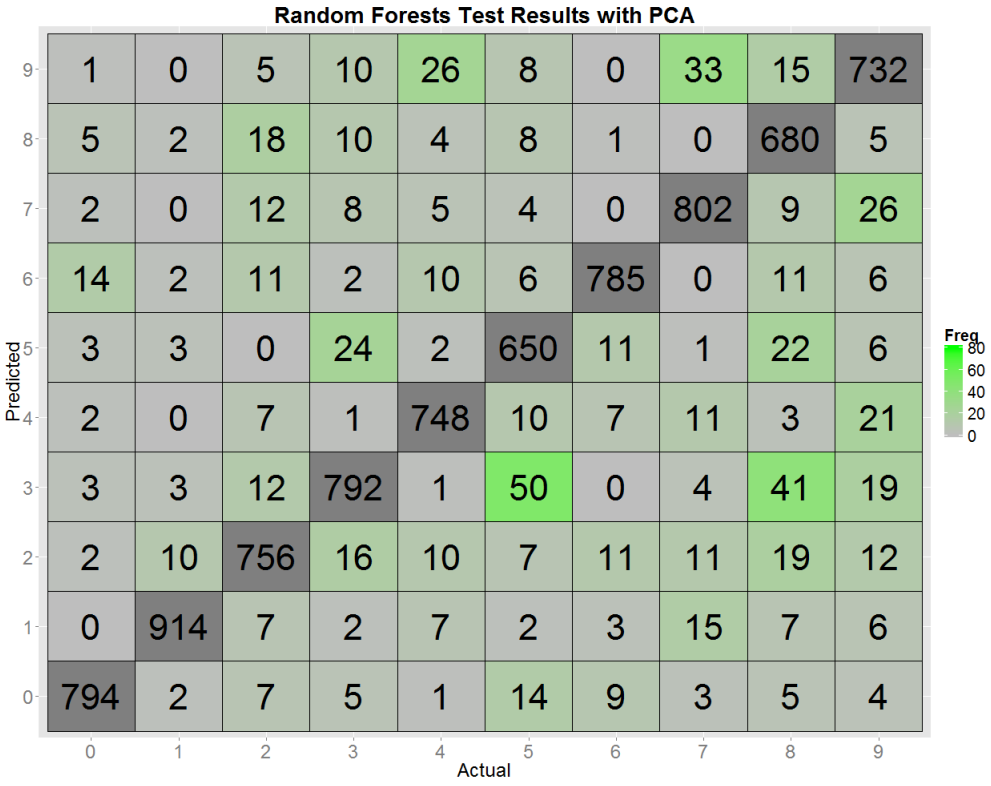


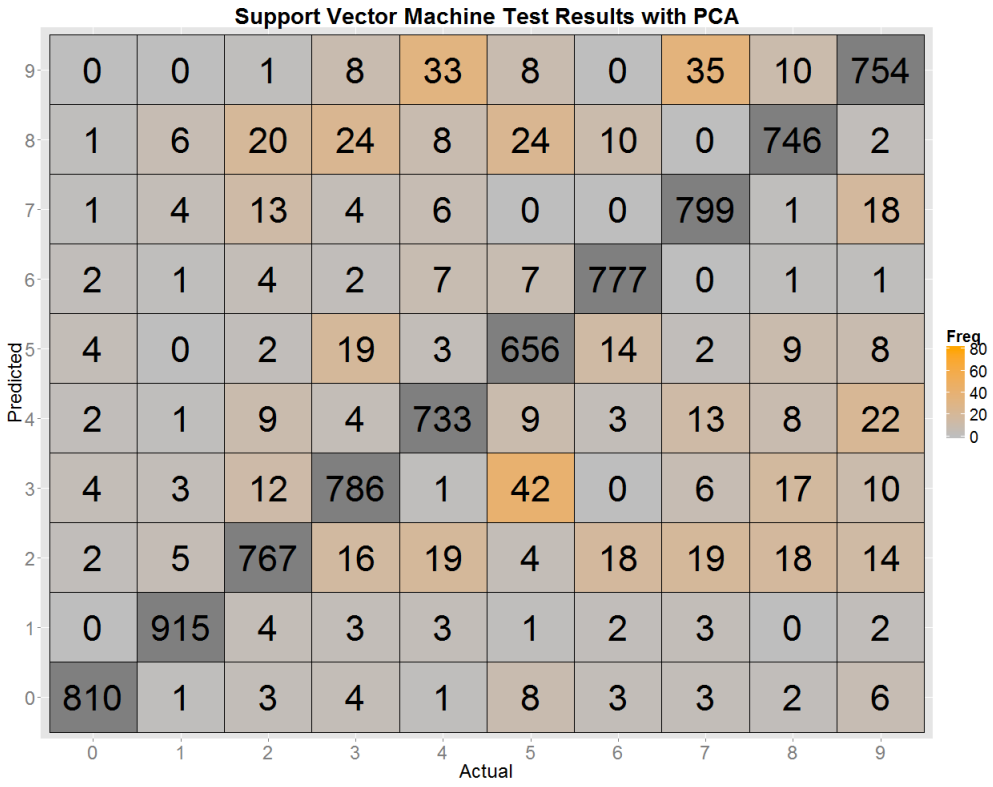

Support Vector Machine Test Results with PCA
Predicted
Actual
Freq
80
60
40
20
0
0 0 1 8 33 8 0 35 10 754
1 6 20 24 8 24 10 0 746 2
1 4 13 4 6 0 0 799 1 18
2 1 4 2 7 7 777 0 1 1
4 0 2 19 3 656 14 2 9 8
2 1 9 4 733 9 3 13 8 22
4 3 12 786 1 42 0 6 17 10
2 5 767 16 19 4 18 19 18 14
0 915 4 3 3 1 2 3 0 2
810 1 3 4 1 8 3 3 2 6

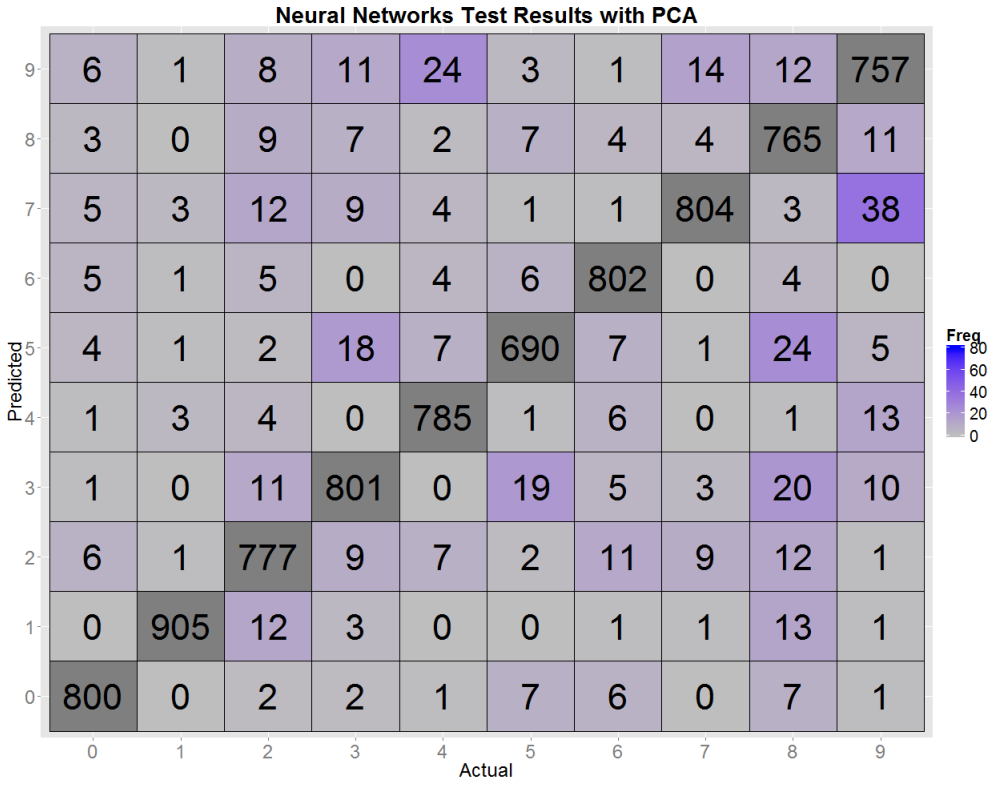

Neural Networks Test Results with PCA
Predicted
Actual
Freq
80
60
40
20
0
6 1 8 11 24 3 1 14 12 757
3 0 9 7 2 7 4 4 765 11
5 3 12 9 4 1 1 804 3 38
5 1 5 0 4 6 802 0 4 0
4 1 2 18 7 690 7 1 24 5
1 3 4 0 785 1 6 0 1 13
1 0 11 801 0 19 5 3 20 10
6 1 777 9 7 2 11 9 12 1
0 905 12 3 0 0 1 1 13 1
800 0 2 2 1 7 6 0 7 1

10.3 Hough-Transformed Confusion Matrices

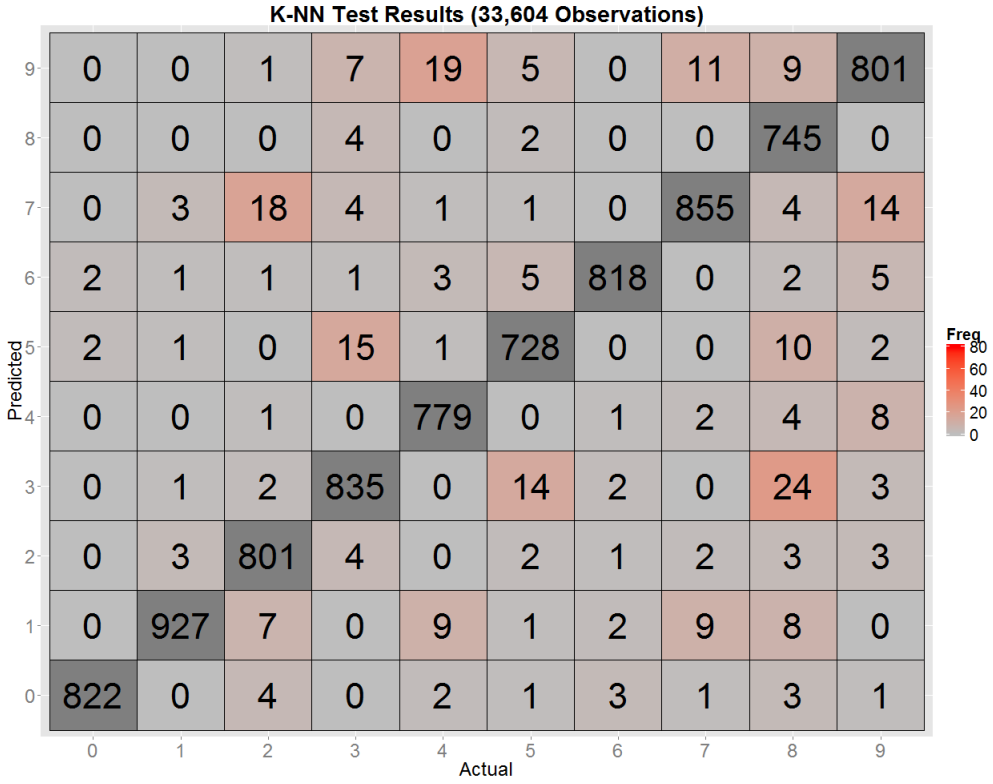


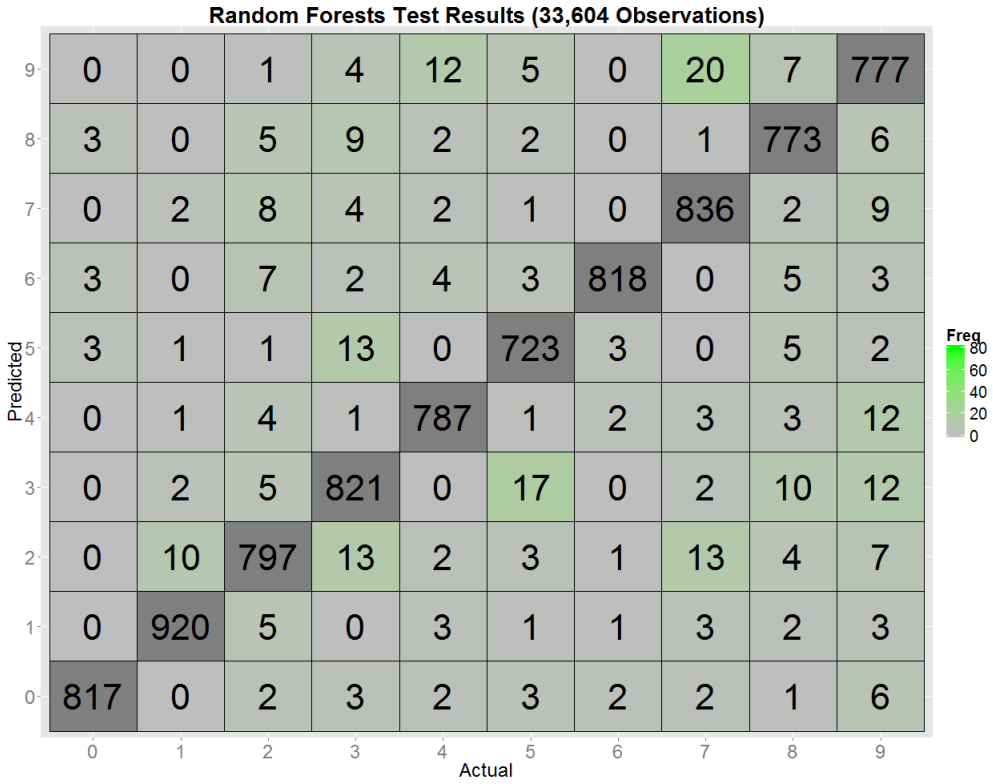


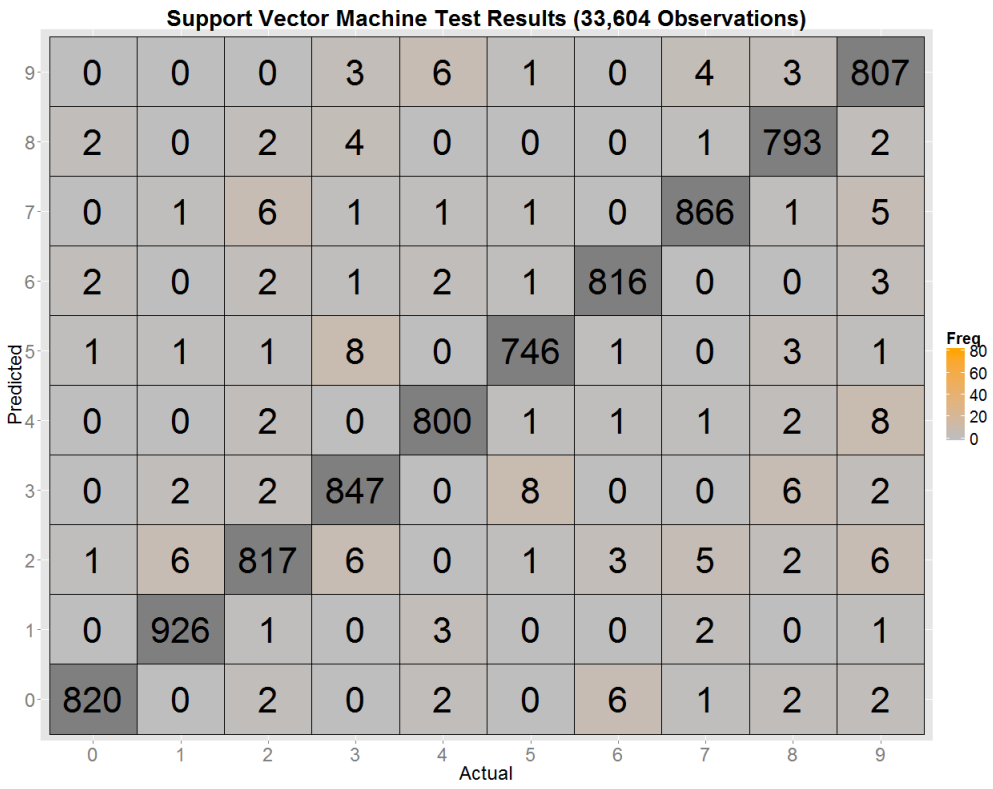

Support Vector Machine Test Results with PCA
Predicted
Actual
Freq
80
60
40
20
0
0 0 1 8 33 8 0 35 10 754
1 6 20 24 8 24 10 0 746 2
1 4 13 4 6 0 0 799 1 18
2 1 4 2 7 7 777 0 1 1
4 0 2 19 3 656 14 2 9 8
2 1 9 4 733 9 3 13 8 22
4 3 12 786 1 42 0 6 17 10
2 5 767 16 19 4 18 19 18 14
0 915 4 3 3 1 2 3 0 2
810 1 3 4 1 8 3 3 2 6

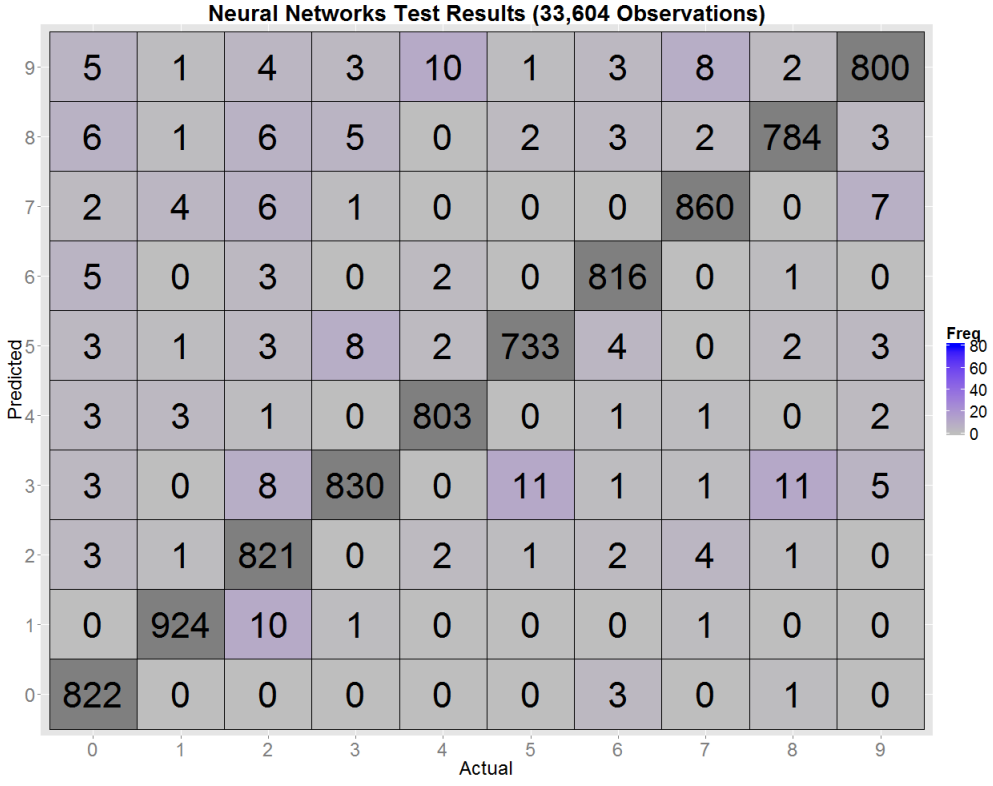

Neural Networks Test Results with PCA
Predicted
Actual
Freq
80
60
40
20
0
6 1 8 11 24 3 1 14 12 757
3 0 9 7 2 7 4 4 765 11
5 3 12 9 4 1 1 804 3 38
5 1 5 0 4 6 802 0 4 0
4 1 2 18 7 690 7 1 24 5
1 3 4 0 785 1 6 0 1 13
1 0 11 801 0 19 5 3 20 10
6 1 777 9 7 2 11 9 12 1
0 905 12 3 0 0 1 1 13 1
800 0 2 2 1 7 6 0 7 1

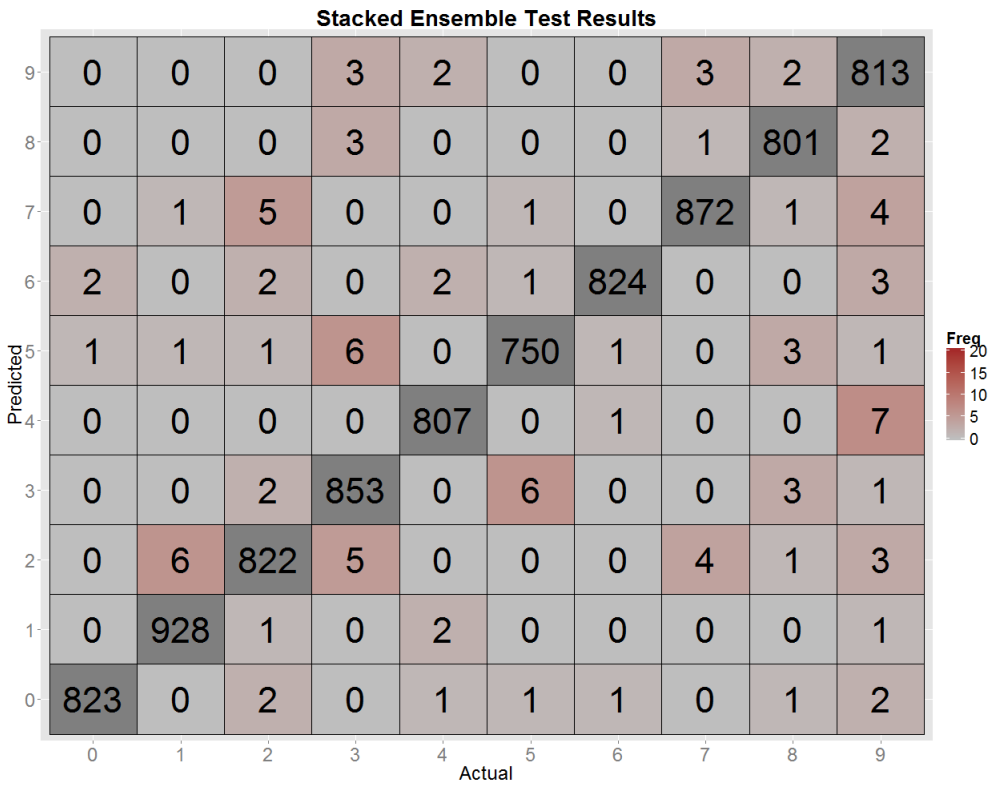
10.4 Large Training Set Confusion Matrices






Support Vector Machine Test Results (33,604 Observations)
Predicted
Actual
Freq
80
60
40
20
0
9 0 0 0 3 6 1 0 4 3 807
8 2 0 2 4 0 0 0 1 793 2
7 0 1 6 1 1 1 0 866 1 5
6 2 0 2 1 2 1 816 0 0 3
5 1 1 1 8 0 746 1 0 3 1
4 0 0 2 0 800 1 1 1 2 8
3 0 2 2 847 0 8 0 0 6 2
2 1 6 817 6 0 1 3 5 2 6
1 0 926 1 0 3 0 0 2 0 1
0 820 0 2 0 2 0 6 1 2 2
0 1 2 3 4 5 6 7 8 9


Neural Networks Test Results (33,604 Observations)
Predicted
Actual
Freq
80
60
40
20
0
9 5 1 4 3 10 1 3 8 2 800
8 6 1 6 5 0 2 3 2 784 3
7 2 4 6 1 0 0 0 860 0 7
6 5 0 3 0 2 0 816 0 1 0
5 3 1 3 8 2 733 4 0 2 3
4 3 3 1 0 803 0 1 1 0 2
3 3 0 8 830 0 11 1 1 11 5
2 3 1 821 0 2 1 2 4 1 0
1 0 924 10 1 0 0 0 1 0 0
0 822 0 0 0 0 0 3 0 1 0
0 1 2 3 4 5 6 7 8 9


Stacked Ensemble Test Results
Predicted
Actual
Freq
20
15
10
5
0
9 0 0 0 3 2 0 0 3 2 813
8 0 0 0 3 0 0 0 1 801 2
7 0 1 5 0 0 1 0 872 1 4
6 2 0 2 0 2 1 824 0 0 3
5 1 1 1 6 0 750 1 0 3 1
4 0 0 0 0 807 0 1 0 0 7
3 0 0 2 853 0 6 0 0 3 1
2 0 6 822 5 0 0 0 4 1 3
1 0 928 1 0 2 0 0 0 0 1
0 823 0 2 0 1 1 1 0 1 2
0 1 2 3 4 5 6 7 8 9

References

[1] knn. http://www.inside-r.org/r-doc/class/knn.

[2] Nearzerovar. http://www.inside-r.org/packages/cran/caret/docs/nearZeroVar. Accessed: 2015-11-25.

[3] Optical Character Recognition: Classification of Handwritten Digits and Computer Fonts. http://cs229.stanford.edu/proj2011/Margulis-OpticalCharacterRecognition.pdf.

[4] Kaggle digit recognizer data. https://www.kaggle.com/c/digit-recognizer/data, 2012. Accessed: 2015-11-25.

[5] The caret package. http://topepo.github.io/caret/index.html, 2015. Accessed: 2015-11-25.

[6] C. C. Aggarwal. *Data Classification: Algorithms and Applications*. CRC Press, 2014.

[7] C. C. Aggarwal and C. Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[8] R. Avnimelech and N. Intrator. Boosted mixture of experts: an ensemble learning scheme. *Neural computation*, 11(2):483–497, 1999.

[9] J. Bobin, J.-L. Starck, and R. Ottensamer. Compressed Sensing in Astronomy. *Selected Topics in Signal Processing, IEEE Journal of*, 2(5):718–726, 2008.

[10] R. Caruana, N. Karampatziakis, and A. Yessenalina. An Empirical Evaluation of Supervised Learning in High Dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 96–103. ACM, 2008.

[11] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[12] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. Pcanet: A simple deep learning baseline for image classification? *arXiv preprint arXiv:1404.3606*, 2014.

[13] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[14] M. Cord and P. Cunningham. *Machine learning techniques for multimedia: case studies on organization and retrieval*. Springer Science & Business Media, 2008.

[15] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[16] D. L. Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 2000.

[17] P. Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[18] A. Giuliodori, R. E. Lillo, and D. Peña. Handwritten digit classification. 2011.

[19] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[20] I. Goodfellow, A. Courville, and Y. Bengio. Deep learning. Book in preparation for MIT Press, 2015.

[21] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R.* Springer Publishing Company, Incorporated, 2014.

[22] R. D. King, C. Feng, and A. Sutherland. Statlog: Comparison of Classification Algorithms on Large Real-World Problems. *Applied Artificial Intelligence an International Journal*, 9(3):289–333, 1995.

[23] R. Kohavi and F. Provost. Glossary of terms. *Machine Learning*, 30(2-3):271–274, 1998.

[24] J. Mairal, F. Bach, and J. Ponce. Sparse Modeling for Image and Vision Processing. *arXiv preprint arXiv:1411.3230*, 2014.

[25] A. McAndrew. An Introduction to Digital Image Processing with MATLAB. *school of computer science and Mathematics, Victoria university of technology*, pages 321–343, 2004.

[26] P. J. Moreno, P. P. Ho, and N. Vasconcelos. A kullback-leibler divergence based kernel for svm classification in multimedia applications. In *Advances in neural information processing systems*, page None, 2003.

[27] J. Nagi. The Application of Image Processing and Machine Learning Techniques for Detection and Classification of Cancerous Tissues in Digital Mammograms. *University of Malaya Kuala Lumpur*, pages 70–73, 2011.

[28] D. J. Watts, P. S. Dodds, and M. E. J. Newman. Identity and search in social networks. *Science*, 296(5571):1302–1305, 2002.

[29] J. Weston, C. Watkins, et al. Support vector machines for multi-class pattern recognition. In *ESANN*, volume 99, pages 219–224, 1999.

[30] D. Wu, G. Ngai, and M. Carpuat. A stacked, voted, stacked model for named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 200–203, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[31] M. Wu and Z. Zhang. Handwritten Digit Classification Using the MNIST Data Set. *Course project CSE802: Pattern Classification & Analysis*, 2010.

[32] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[33] N. Zumel, J. Mount, and J. Porzak. *Practical data science with R.* Manning, 2014.