Brandon Croarkin

# HW7: SVMs, kNN, and Random Forest for handwriting recognition

## Section 1: Introduction

Optical character recognition (OCR) is a topic that has been at the forefront of machine learning in recent years. OCR is the process of converting images of typed, handwritten or printed text into machine-encoded text. This can come from scanned documents, photos of documents, or even images with superimposed text.

Today, OCR has become capable of a high degree of recognition accuracy from most fonts, with a support for multiple languages as well. It is widely used to assist with information entry from printed paper data records. When successful, it can drastically cut down manual data entry time. As the sophistication of these OCR algorithms have improved, its applications have spread. OCR has almost become ubiquitous and can found in tasks such as passport recognition, converting electronic images of printed documents to searchable versions (Google Books), automatic number plat recognition, and even extract business card information into a contact list.

The goal of this report is to correctly identify digits from a dataset of tens of thousands of handwritten images using the MNIST dataset. To correctly classify images there are many steps that need to be done first. One, a large dataset of data is needed. Without a large collection of data, it is hard if not impossible to be able to account for all the variations in how a digit can be written. Next, these images need to be of uniform dimensions. The algorithms used are unable to compare a 28x28 pixel image to a 50x50 pixel image unless it resizes the images within the algorithm. Furthermore, the images’ pixels need to be on the same color scale. Color images are 3-dimensional and contain data for the level of red, green, and blue in a pixel. This is very different than 1-dimensional black and white images that has a single integer value between 0 and 255, inclusive, with higher numbers meaning darker. Again, it is impossible to compare color images to black and white images using these algorithms without pre-processing to put them on the same scale.

### The Data

MNIST (“Modified National Institute of Standards and Technology”) is the de facto “hello world” dataset of computer vision. It was originally released in 1999 and has since served as the basis for benchmarking classification algorithms. It consists of a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-set image. These images are 28x28 pixel grayscale.

For testing purposes, the 60,000 example training set was divided into two datasets of 42,000 and 28,000 examples for training and testing of models. Due to the large size of the dataset, the 42,000 example training set was further cut down into just 4,000 random examples for training and 400 random examples for testing.

The data pre-processing steps were the same for each of the models used. The number label was transformed into a factor variable to work with the algorithms. Additionally, the pixel values were all scaled to a 0-1 scale using max-min normalization. There were no missing values found in the dataset.

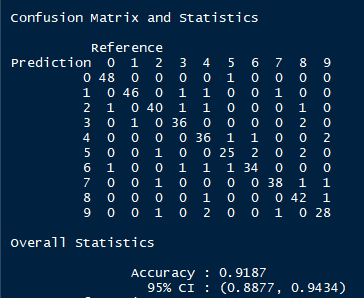
## Section 2: SVMs

A Support Vector Machine (SVM) is a supervised learning model that applies the statistics of support vectors to categorize unlabeled data. SVMs are very powerful algorithms that can be used for nearly any type of learning task, including both classification and numeric prediction.

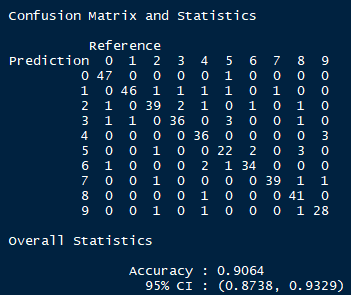
An SVM can be imagined as a surface that creates a boundary between points of data plotted in a multidimensional space that represent the examples and their feature values. The goal of SVM is to create a flat boundary, called a hyperplane, that divides the space to create fairly homogenous partitions on either side.

Kernels are a key feature of SVMs due to their ability to map a problem into a higher dimensional space. Essentially, kernels involve a process of constructing new features that express mathematical relationships between measured characteristics. There are various types of kernel functions used within SVM, such as linear, polynomial, radial, and sigmoid. The linear kernel is the simplest and actually does not transform the data at all. It is expressed as the dot product between the features. The polynomial kernel of degree *d* adds a nonlinear transformation of the data. The radial basis kernel uses the squared Euclidean distance between features to help map the dimensions. The sigmoid model uses a sigmoid activation function, similar to a neural network model.

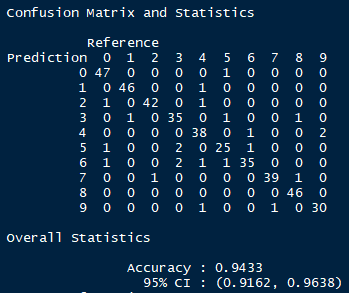
**SVM Model 1 – Caret Package**



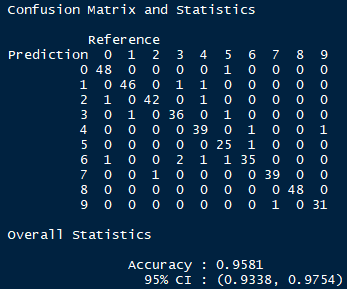
**SVM Model 2 – kernLab Package – Linear kernel**



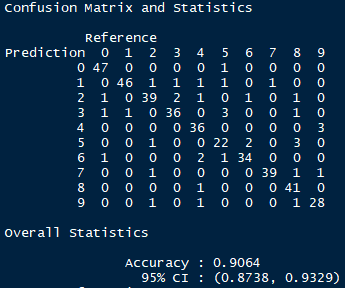
**SVM Model 3 – kernLab Package – Radial Basis**



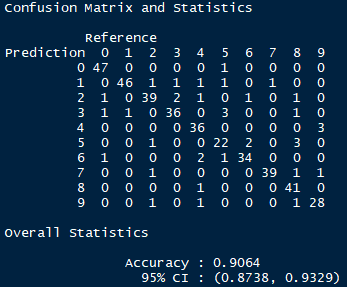
**SVM Model 3.5 – kernLab Package – Radial Basis with C = 5**



**SVM Model 4 – kernLab Package – Polynomial**



**SVM Model 5 – kernLab Package – Sigmoid**



SVM models were run with each of the different kernels detailed above and the default parameters. Based on the default parameters in the kernLab package, the radial kernel function performed best so this algorithm was the model that was tuned further. After trialing with multiple C values, 5 was found to be the best performing with a 95.81% accuracy on the test set.

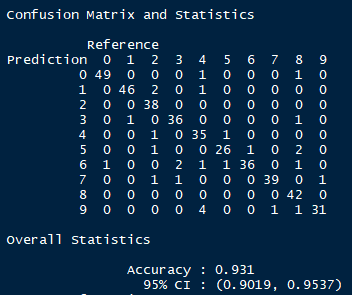
## Section 3: kNN

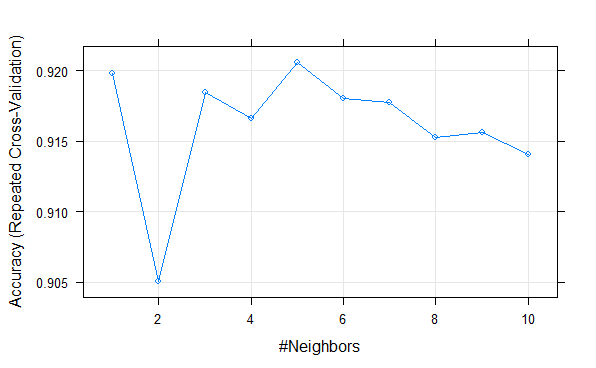
The k Nearest Neighbor (kNN) algorithm is another approach for classification and is one of the simplest, yet also powerful, algorithms around. kNN classifiers are defined by their characteristic of classifying unlabeled examples by assigning them the class of similar labeled examples. The letter *k* in kNN refers to the number of neighbors used to classify unlabeled examples. For each unlabeled record, kNN identifies *k* records in the training data that are the “nearest” in similarity. The unlabeled test instance is assigned the class of the majority of the k nearest neighbors. Due to the voting nature of kNN models, it is best to choose a *k* with an odd value to prevent split votes.

kNN is an example of a “lazy learner” algorithm. The training phase of kNN actually does not involve any work besides just storing the examples of the data. Because of this, kNN has a very fast training phase, but this also means that it does not provide a model to understand how the features are related to the class.

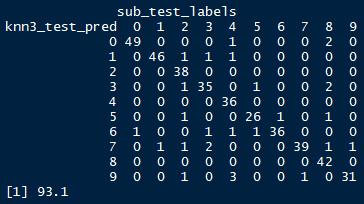
Since the kNN model uses a distance formula (typically Euclidean distance), it is important that the features are transformed to a standard range prior to applying the algorithm. If certain features have a much larger range of values than the others, the distance measurements will be strongly dominated by the features with large ranges. The pixel values in the MNIST data is already on a standard scale of 0-255, but for the kNN, Random Forest, and SVM models these values were converted to a 0-1 scale using min-max normalization.

**kNN Model 1 – Caret package – Training Control**

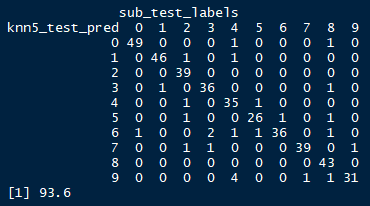




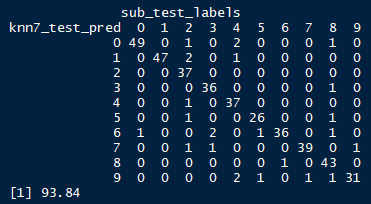
**kNN Model 2 – Class package – k = 3**



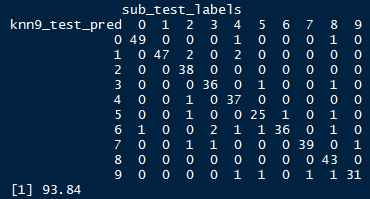
**kNN Model 2 – Class package – k = 5**



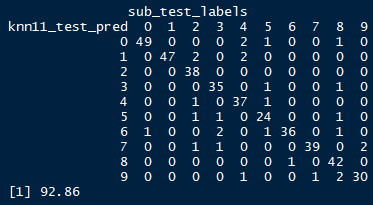
**kNN Model 2 – Class package – k = 7**



**kNN Model 2 – Class package – k = 9**



**kNN Model 2 – Class package – k = 11**

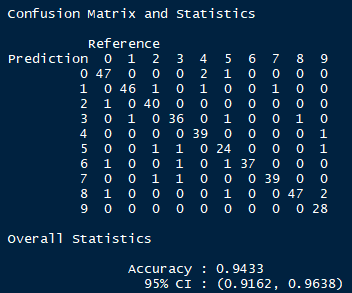


After doing training control with the Caret package, 5 was said to be the number of neighbors with the best accuracy. However, after testing the kNN model from the Class package with numerous *k ­*values, 7 and 9 were found to have the best accuracy with 93.84%.

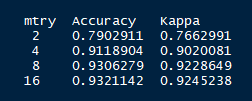
## Section 4: Random Forest

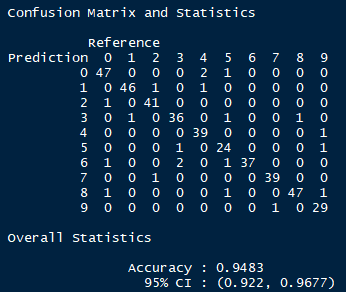
Random forest is an example of an ensemble-based method. Ensemble methods are based on the idea that by combining multiple weaker learners, a stronger learner is created. Random forest focuses only on ensembles of decision trees. It uses the base principles of bagging with random feature selection to add additional diversity to the decision tree models. After the ensemble of trees is generated, the model uses a vote to combine the trees’ predictions. Since the ensemble uses only a small, random portion of the full feature set, random forests can handle extremely large datasets and avoid the “curse of dimensionality” that can cause many other models to fail.

**Random Forest Model 1 – randomForest package – default parameters**



**Random Forest Model 2 – Caret package – Train control (5 folds, 3 repeats)**





After running trainControl from the Caret package, it was found that 16 was the optimal number of variables randomly samples as candidates at each split. Further testing with the number of tree sizes used did not result in any higher accuracy than the default of 500 trees used. In the end, the best Random Forest model achieved an accuracy of 94.83% on the test dataset.

## Section 5: Algorithm Performance Comparison

The SVM tuned model with a radial kernel and cost of 5 yielded the highest accuracy on the testing dataset of the three algorithms used. While it did perform the best, it was also the model that required the most tuning and had very long training times. This large training time could be due to the large number of features that were included in the image dataset. Since SVM needs to convert these features into a high dimensional space, the large number of features likely played an impact in the speed of the algorithm. Its great performance is likely due to SVMs tendency to not be overly influenced by noisy data and thus overfitting. Furthermore, the fact that a lot of tuning was required also could be a reason for why the SVM algorithm ended up performing the best. This tuning allowed the algorithm to use the best possible parameters to reach a solution.

The Random Forest was the next best performing model. It was also the longest to tune when training to find the best number of features to use. This makes sense since it is already an ensemble method, so the training process to find the optimal number of features exponentially increased the computations needed to be ran. The tuning of the model was very easy after this was done though. It also did a great job with the large number of features that were included in the dataset. This is likely due to its ability to select only the most important features.

The kNN model was the worst performing model of the three used, but it was also the fastest and easiest to tune. Additionally, even though it had the lowest accuracy of the three algorithms used, it still had very good performance of around 93%. The simplicity of its algorithm is likely why it was able to perform so fast, since it does not actually do any learning and merely stores the examples.

None of these models were able to give outputs that were interpretable. Because of this it is unclear how they were able to distinguish one number from another.

## Section 6: Kaggle Test Result

