**Introduction**

Handwriting recognition is the ability of a computer or a mobile device to read handwriting as actual text from sources such as printed physical documents, pictures and other devices, or to use handwriting as a direct input to a touchscreen and then interpret this as text. The input is usually in the form of an image such as a picture of handwritten text that is fed to a pattern-recognition software, or by using a camera for optical scanning. The human brain effortlessly recognizes handwritten digits through complex image processing. But it isn’t as easy as it looks. The difficulty of visual pattern recognition becomes apparent if you attempt to write a computer program to recognize digits like those above. What seems easy when we do it ourselves suddenly becomes extremely difficult. Machine learning approaches the problem in a different way by taking several handwritten digits known as training examples and develops a model which can learn from these training examples. This paper aims to identify the accuracy of correctly identifying handwritten digits from a dataset of tens of thousands of handwritten images. This is accomplished by using a couple of classification algorithms like the decision tree and the naïve Bayes classifiers.

**Analysis and Models**

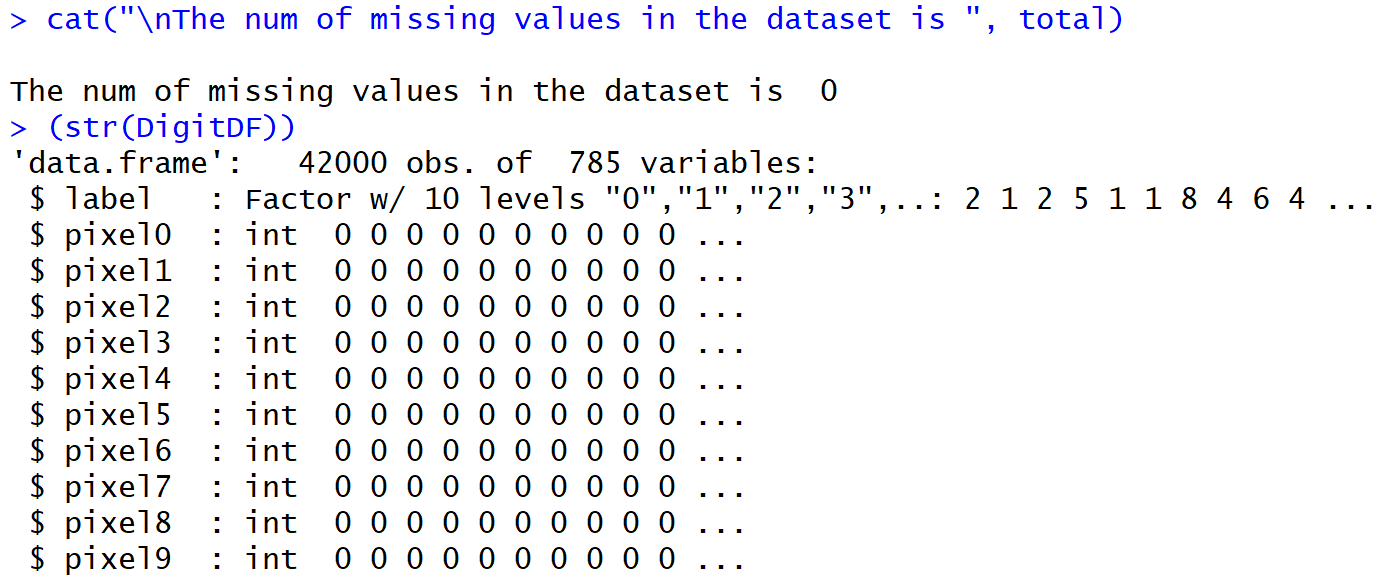
**About the Data**

MNIST ("Modified National Institute of Standards and Technology") is a database of handwritten digits that serves as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike. The data files Kaggle-digit-train.csv and Kaggle-digit-test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

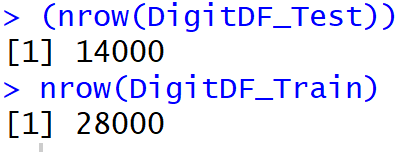
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (Kaggle-digit-train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

As always, data must be cleaned and preprocessed. Once the training dataset is loaded into R, the same is checked for null values. Since there are no missing values, the next step is to check fi the variables are of the right data type. An important thing to note is the fact that the label of a dataset should always be of factor datatype. Hence, the label is converted into a factor. The structure of the transformed data is as follows:



In order to test the accuracy of the model that is going to be built, the training dataset is divided further into a train and test set. Using cross-validation techniques, every 3rd row from the training dataset is pulled out and formed as a test dataset. The rest of the rows are kept in the training dataset. Now, the new training set has 28,000 rows and the new test set has 14,000 rows of data.



An important task to take care of in the test dataset is to make sure the labels are removed since this is the column that is going to be predicted using the model. The next step is to apply the classification techniques to the training dataset to build a trained model.

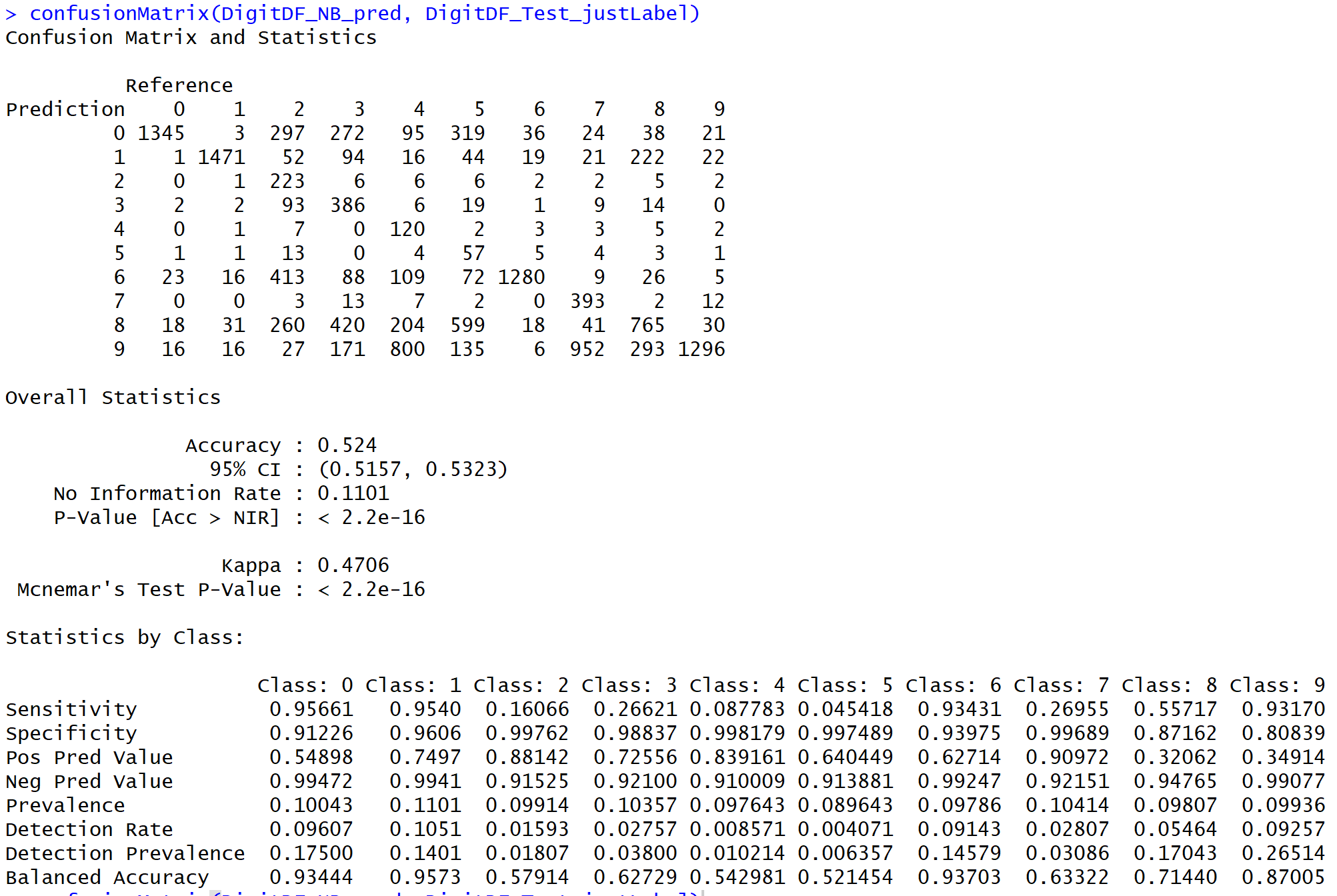
**Classification Techniques**

As per the definition provided in the book ‘Introduction to Data Mining’, classification is the task of learning a target function *f* that maps each attribute set *x* to one of the predefined class labels *y*. Classification techniques are most suited for predicting or describing datasets with binary or nominal categories. Different classification models can be built from an input data set using classification techniques. Some of the examples include decision-tree classifiers, rule-based classifiers, neural networks, support vector machines and naïve Bayes classifiers. In this paper, decision tree algorithm and naïve bayes classifier is used to classify the handwritten digits. A general approach for solving classification problems is to divide the records of a data set into a training set and a testing set. The training set is used to build the classification model, which is subsequently applied to the test set that consists of the records with unknown class labels.

**Naïve Bayes Classifier Algorithm**

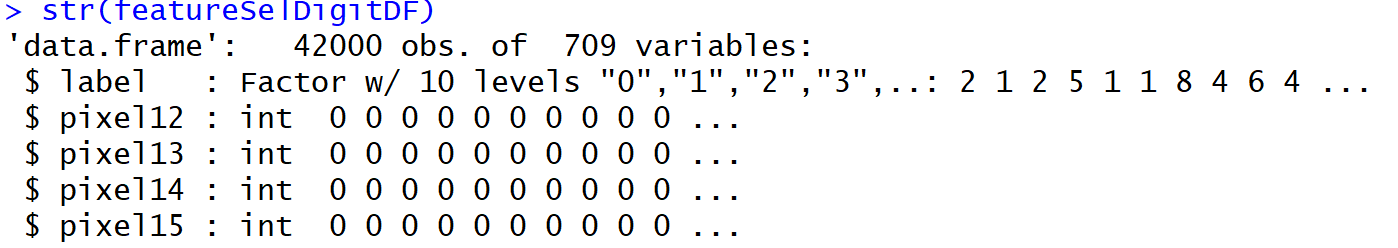
Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values. It is called naive Bayes because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value, they are assumed to be conditionally independent given the target value. This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

In order to classify the handwritten digits in the training dataset, the naïve Bayes model is applied by selecting *label* as the Y variable and the rest of the columns as the X variables. The trained model is then applied to the test dataset that doesn’t have any of the labels. A confusion matrix was used to find out the accuracy of this model. The below snapshot shows an accuracy of 52% which is not that great.

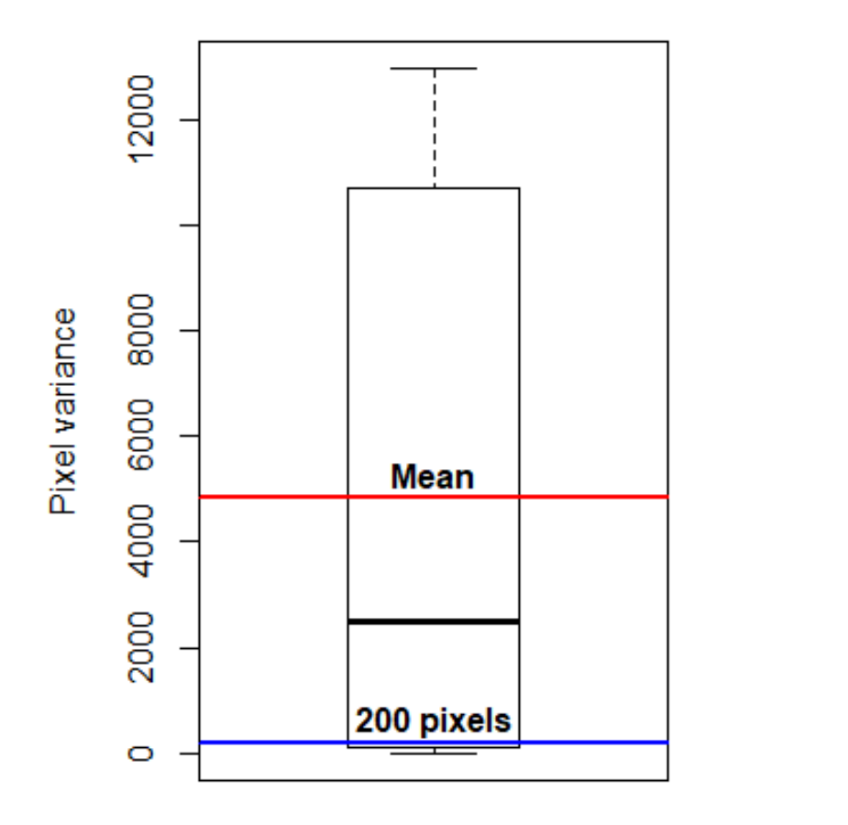


To improve the accuracy, a different approach was implemented by cleaning the dataset more.

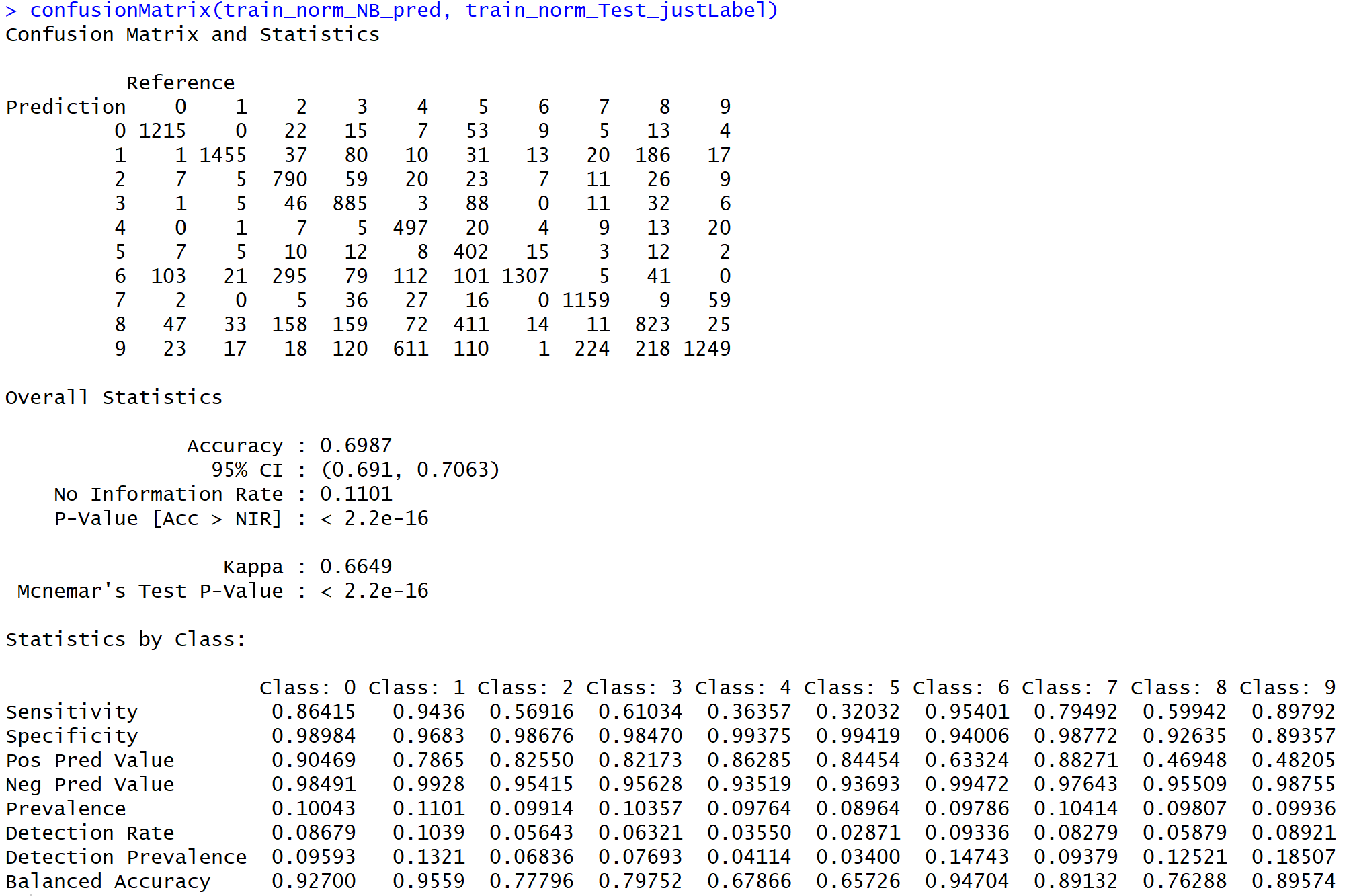
In the second model, feature selection was implemented by removing all the pixel columns that had an average of zero since this is of no use to our model. Through feature selection, the number of attributes reduced to 709 from 785.



To reduce the dimensionality even further, more pixel columns should be removed. Pixels with little variance will not contribute to the digit images. Hence, by eliminating those pixels with less variance, the pixel variables were brought further down by 509. The below plot helps to find out about how many pixels have a low variance and close to which quantile.



Next, the data is normalized using the normalize function. The dataset is once again divided into train and test subsets and the naïve Bayes model is applied on the trainset to get a trained model. Once the model is built, the model was applied to the test dataset to predict the label. A confusion matrix showed the accuracy rates to be 70% which is so much better than the previous model.



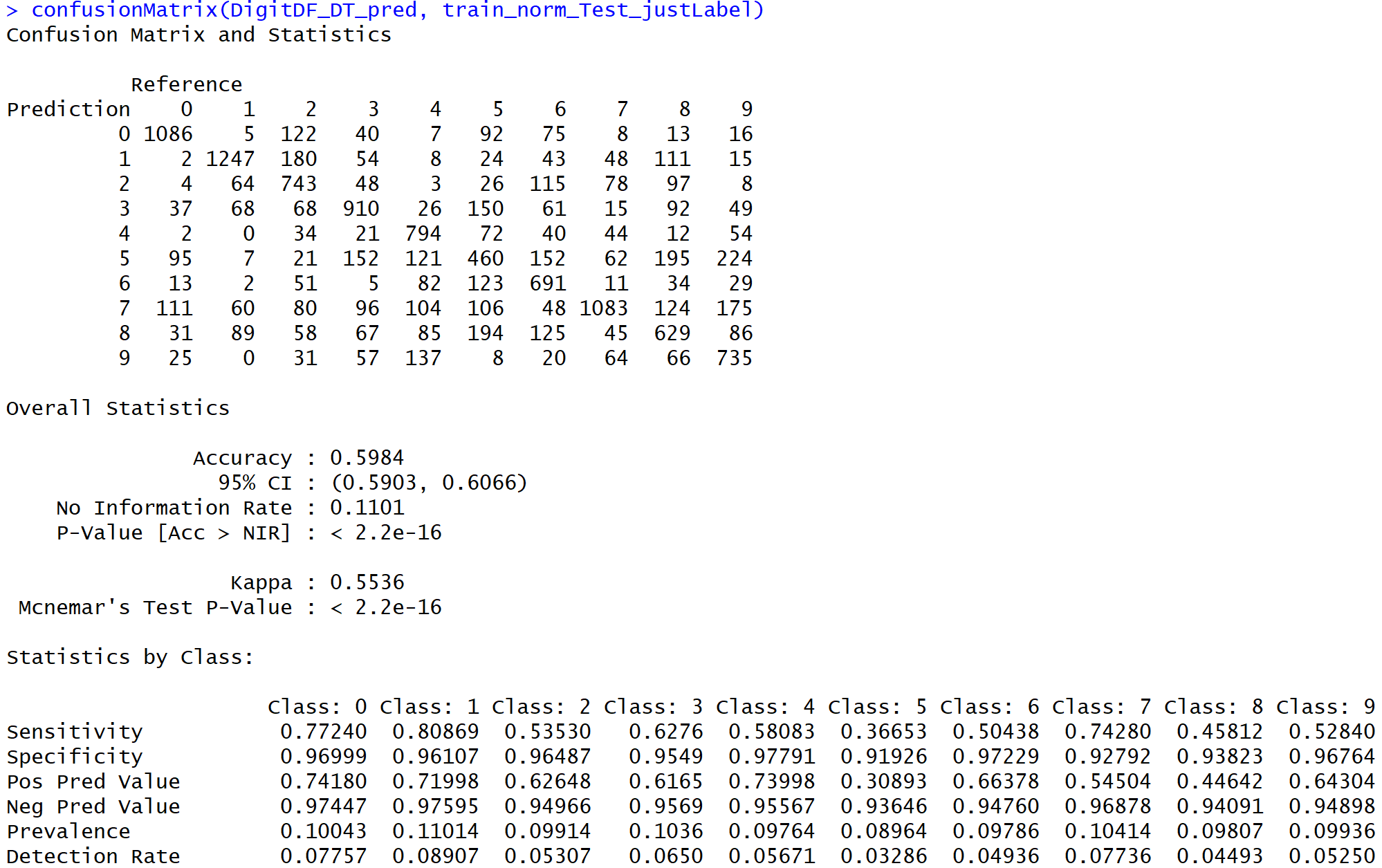
To see if better accuracy rates can be achieved, a different classification technique such as the decision tree algorithm is applied to the dataset to see if there are any changes.

**Decision Tree Algorithm**

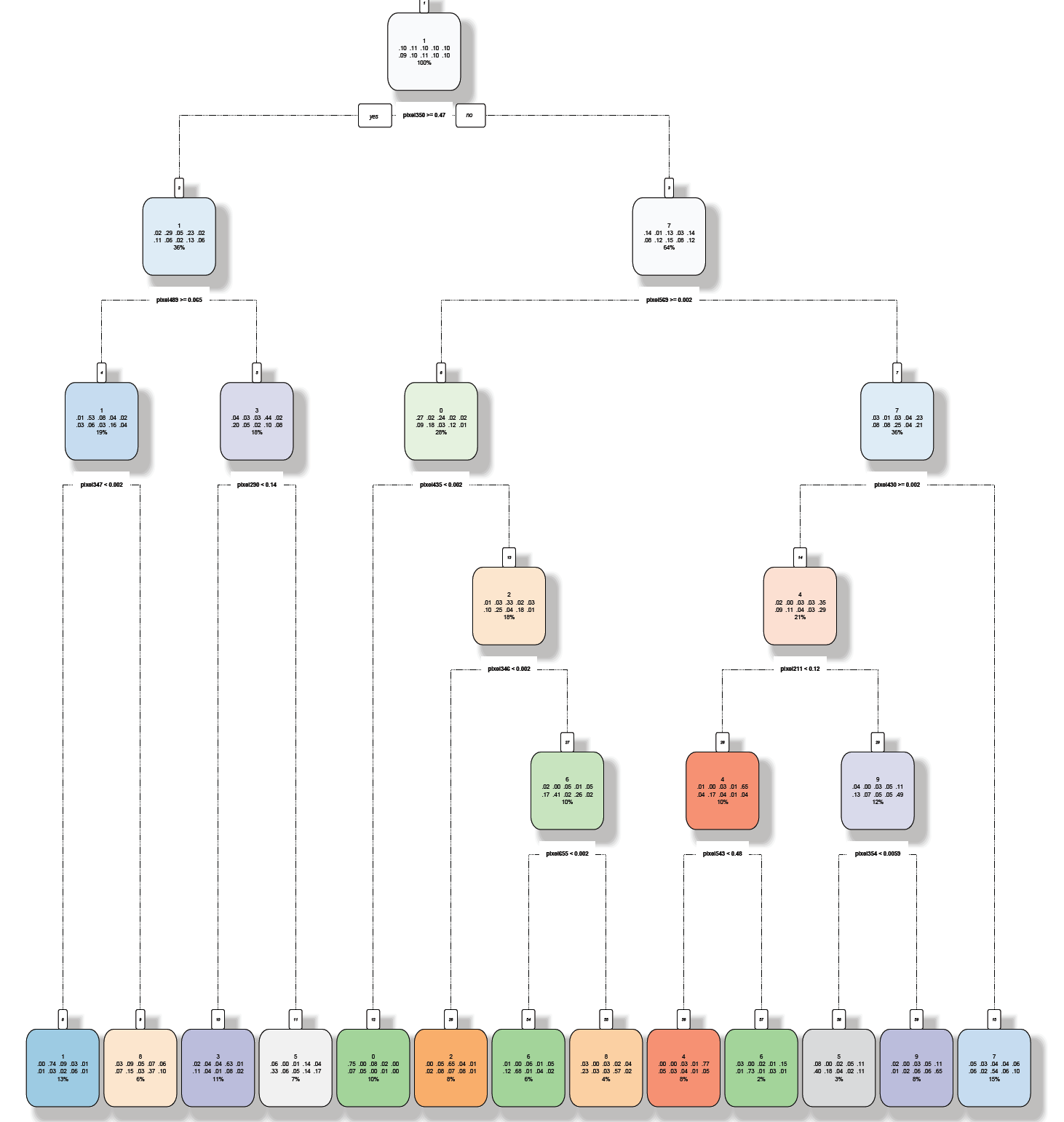
Decision tree algorithm is a simple and widely used classification technique that has several advantages. After the model has found the patterns in the data, it shows what decisions will be made for unseen data predictions. Decision trees are intuitive and can be read by people with little experience in data mining. Basically, the algorithm starts with all the data at the root node and scans all the variables for the best one to split on. The leaf nodes or the terminal nodes are the ones with the class labels or the column name that needs to be predicted. Classifying a test dataset is straightforward once a decision tree model has been built. Starting from the root node, the test condition is applied to the test records and the appropriate branches are followed based on the outcome. This will either lead to another internal node with another variable or to a leaf node with the class label.

For the analysis, the *rpart* function is used which works by splitting the dataset recursively, which means that the subsets that arise from a split are further split until a predetermined termination criterion is reached. As parameters, the training dataset is provided which specifies to use the *label* column as the *y* variable and use the rest of the columns in the train dataset as the *x* attributes. The *method* parameter is set to *“class”,*which simply tells the algorithm that the predicted variable is discrete. Some other control parameters are also set to govern how many files\rows should sit in a bucket before looking for a split (minsplit) and the *cp* parameter is set to stop splits that aren’t deemed important enough.

Once again, the model was built and the same was tested against the test dataset to predict the class labels. Note that the model was applied to the previously curated normalized and reduced dataset. A confusion matrix showed that 60% of the results were accurate. The result accuracy is not as great as the naïve Bayes model.



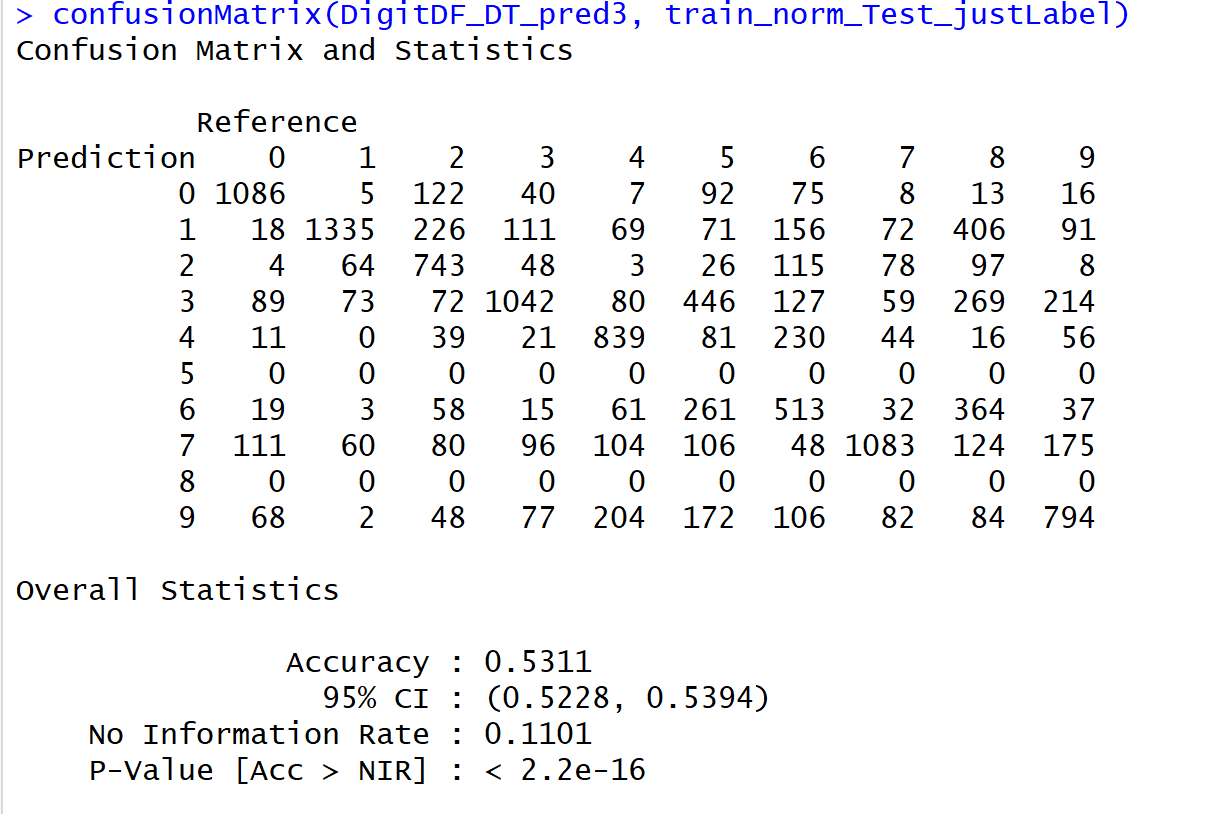
Below is a screenshot of the decision tree that was built from this model. A better view of the tree is attached as a pdf file.



The decision tree shows that the entire model was first classified based on the pixel column 350 and was further divided using various other pixel numbers.

Parameters of the decision tree model were finetuned to see if that gave any different results. For example, changed the cost parameter value to 0.05 to cut down the tree. But this did not help. In fact, it reduced the accuracy level to 45% which is not good.

Again, the cp value was reduced to 0.03 to test the accuracy level. The below image shows that the accuracy level went up to 53%.

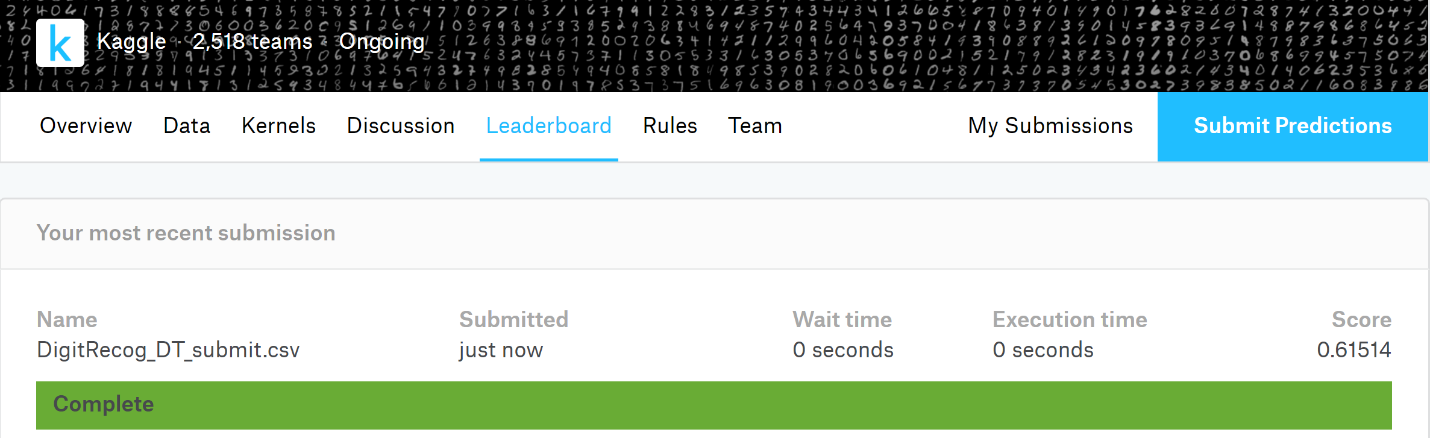


Lesson learnt is that by increasing the cost parameter, the accuracy levels are going down and is not helping to achieve higher accuracy results. Keeping the cp=0.01, the highest accuracy level for a decision tree classifier was achieved which is 60%.

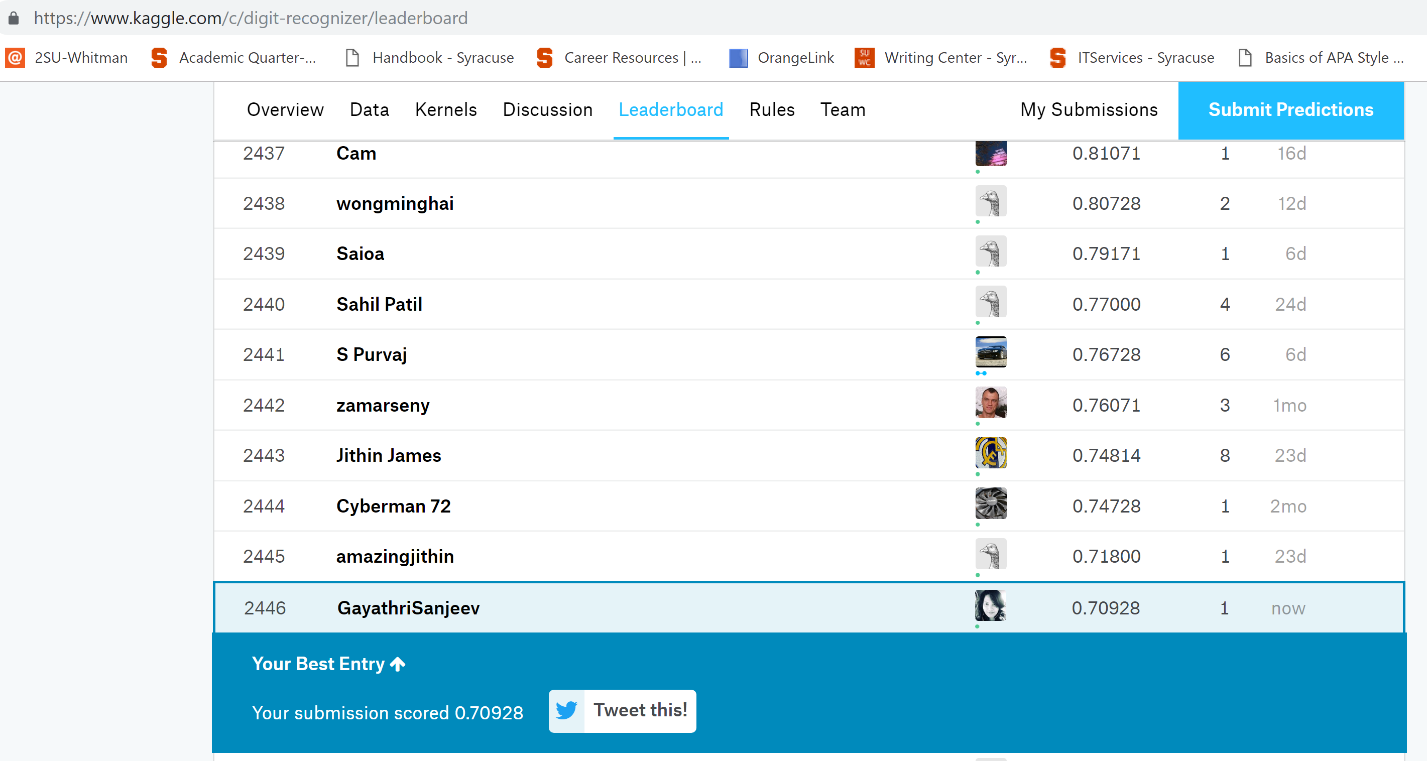
**Results**

Finally, the model was applied to the original test dataset which doesn’t have any class labels.

When the trained decision tree model was applied, a score of 62% was achieved which is slightly better than the results achieved on the training dataset by applying the same model.



By applying the naïve Bayes model on the test dataset, the model was able to achieve 71% accuracy which is slightly higher than the accuracy on the training set. Below is a screenshot of the accuracy rate from Kaggle.



**While the naïve Bayes model gives impressive performance, that performance is still not as good as the human brain. It cannot be concluded that this is the best a model can do. Different techniques and models need to be applied on this dataset to find the best fit. Just by comparing the two algorithms that has been used here, naïve Bayes is the best choice for this dataset. But it is not the only one and there are various other techniques like CNN or Random forests that needs to be explored.**

**Conclusion**

**As it was explained in the beginning of this paper, the human brain is incredible in identifying complex structures of images. Transforming this application to the computers and making it as good as the human brain was not simple but doable. The concept of machine learning has opened doors and made impossible feats possible. One day, machines will probably learn to do everything a human brain can do and much more. The limits are endless. But, more than anything, employing these techniques to make this world a better place should be the end goal.**