# HAND-WRITTEN NUMBER CLASSIFICATION PROBLEM

Date 29-04-2018

Teresa Riedl, University of Cape Town

## A) Introduction

In this report we evaluate the best method to build a prediction model for hand-written number recognition. The idea is to use several different models and identify the best one based on various evaluation methods.

I want to start of with an overview of the given data set and some transformations that I used to simplify the data and make it compatible for most of the modelling methods. Then after dividing the "train" data into a test and a training set I step into several different methods. As we are dealing with a classification problem I first use logistic regression, evaluate whether to use Linear and Quadratic Discriminant Analysis and use K-nearest neighbors. Then I move on to Tree-based methods. To start I build a classification tree and improve its performance with the more complex tree building models Bagging, Random Forests and Boosting. Lastly I build models using Support Vector Machines and Neural Networks.

In the end I compare different performance indicators of all models to choose the best performing one and predict the output of the unseen test data set which will be submitted along with the report.

Upfront I expect Neural Networks to be one of the better performing models because it's commonly used for image recognition tasks. Support Vector Machines are especially popular for non-linear cases which is the case here.

Then I want to state some limitations. My goal was to use the AUC (area under the curve) metric to compare the different models along with the Accuracy and the Error rates. Unfortunately, I didn't succeed to get results for all models.

## B) Data set and transformation of variables

Variables can be classified as either quantitative or qualitative. Whilst in other reports we dealt with a quantitative and therefore regression problems, this report is referring to a classification problem with qualitative outcome variables. We are given a training and a test with each 2500 observations of 785 variables. The goal is to predict whether a hand-written number that consists of 28x28 pixels is even or odd.

1. Convert output variable into binary values with 0 for even and 1 for odd

The data sets include two different types of variables: the output variable "Digit" from type integer is the result that we want to predict correctly. As the goal is to predict either even or odd I will convert the variable into a binary variable with 0 for even and 1 for odd. I will use the same binary outcome for the test set once I predict values with our model at the end.

2. Take out variables with only 0's in them

The 784 predictor variables are each one pixel of the created image with the hand-written number that we are trying to predict. Looking at the data it is noticeable that some columns only consist of zeros. To reduce complexity, I take out all columns that do not change across all observations or are very close to 0. Otherwise some of the methods I'm planning to use will not work as well. I thereby reduce the dimension of the data to 247 instead of 784 predictors.

3. Scaling the data

I'm planning to use several methods such as KNN and SVM that require (or recommend) to scale the data. To choose an appropriate scale we calculate the maximum pixel number and divide all pixels by that value. The maximum is 255. After scaling the pixel variables, the maximum value is 1.

Before we start building our model we subset the training data into a training (train.x) and a test (test.x) set and keep the unseen test set aside for final validation.

## C) Outline of steps used for model building

1. Logistic Regression

   Logistic Regression is a regression method for classification problems with binary response variables. Instead of predicting the outcome it helps us to estimate the probabilities for each category. To estimate the Logistic Regression, it's uncommon to use the least squares approach. The more general method it the maximum likelihood function

$$l(\beta_0, \beta_1) = \prod_{i:y_{i\prime}=1} p(x_i) \prod_{i':y_{i\prime}=0} (1 - p(x_{i'})$$

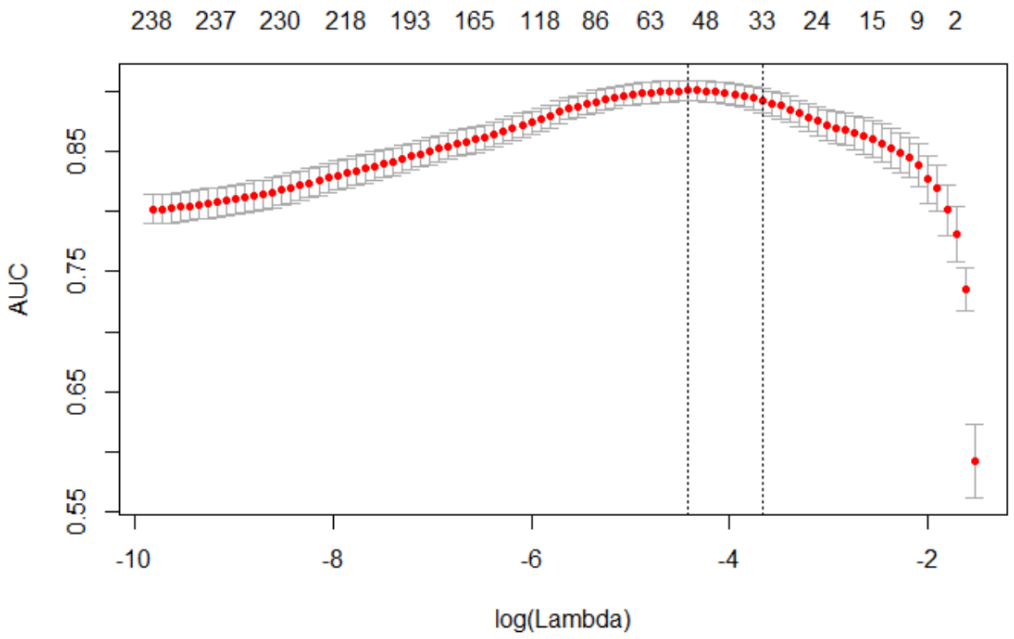
   Where the estimates $\beta_0$and $\beta_1$are chosen to maximize the likelihood function. As we have more than 200 variables I won't outline the individual $\beta$ values here but move on to the mdel outcome. In our case we are estimating the probabilities for even (0) and odd (1) values where probabilities above 0.5 are classified as odd and below as even.

   We use glm() and predict() in order to retrieve the correctly and incorrectly classified values. The following table shows the result for 1000 observations in the test set (test.x):

| **Predictions** | **0 = EVEN** | **1 = ODD** |
|---|---|---|
| **0 = EVEN** | 385 | 96 |
| **1 = ODD** | 117 | 402 |

We observe that 213 observations out of 1000 were misclassified and 787 were correctly classified. The accuracy of the logistic regression model is 78.70 %.

ROC curves compare the True and False positive rates of predicted values and are therefore a very good indicator for the performance of a classifier. The area under the curve (AUC) returns the overall classifier performance by summarizing the over all possible thresholds. The AUC is supposed to be as large as possible. To identify the best model performance I used cross validation and then calculated a mean of 85.13% for the AUC value. The following graph shows though that the AUC ranges between 59.25% and 90.06% throughout the cross validation subsets.




## 2. Discriminant Analysis

### 2.1 Linear Discriminant Analysis (LDA)

LDA uses a different approach to Logistic Regression. Instead of modelling the conditional distribution of the response Y (in our case Digit) given the predictors X (Pixels), we model the distribution of the predictors X separately in each of the response classes. Afterwards we use the Bayes theorem for classification.

Reasons to use a different approach are that if

- classes are well-separated, the parameter estimates for the LR model are unstable
- n is small and the distribution of the predictors is approximately normal in each of the classes, the LDA is more stable
- we have more than two response classes, LDA is more popular. But this doesn't apply in this case.

To pursue the LDA we use the lda() function. We then use the Bayes theorem for classification into K = 2 classes. The following formula describes the Bayes theorem where $\pi_k$ represents the prior probability that a randomly chosen observation comes from the kth class; and $f_k(x)$ is the density function for an observation in the kth class.

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

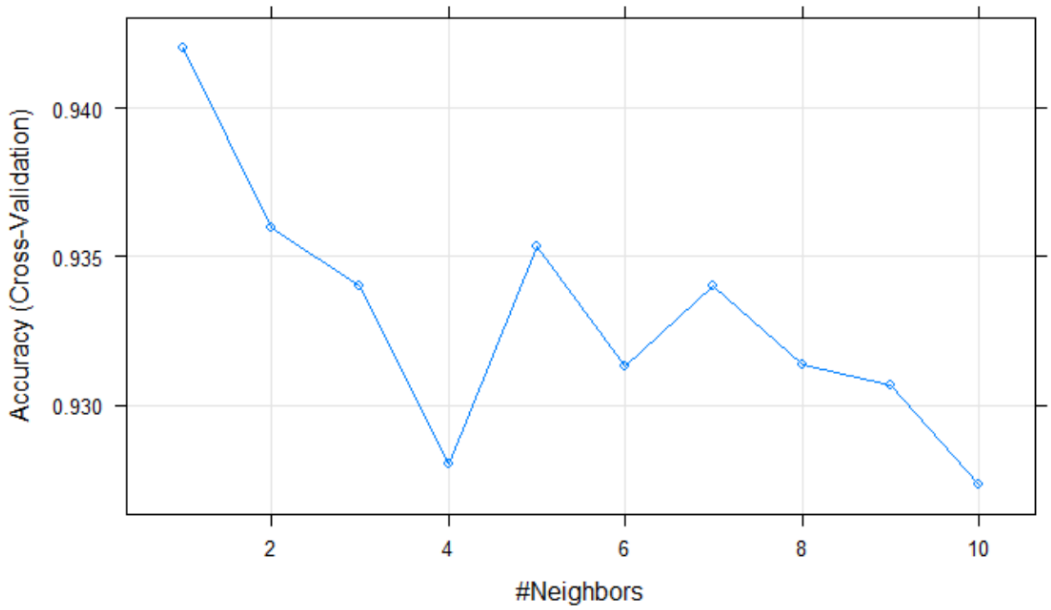
The prior probability that a randomly chosen observation is an even number is $\pi_0$ = 49.80% and the prior probability to randomly select an even one is $\pi_1$ = 50,20%. The model accuracy reaches 79,70%.

2.2 Quadratic Discriminant Analysis (QDA)

There is no indication that there is a normal gaussian relationship between the data and the data has a very high dimensionality. The model does result in 86,60% accuracy but the arguments are pointing against it for the given data set.

## 3. K-nearest Neighbors (KNN)

We use the training (train.x) and test (test.x) set we created from the original training (train) set before to apply the knn() function. I use cross validation to decide which k value builds the best model. The following graph shows which number of neighbors K allows for the highest accuracy.




K=1 results in a 94,40% accuracy (or 94.20% as a cross validation mean) in predicting whether a number is even or odd. The accuracy is already outstanding.

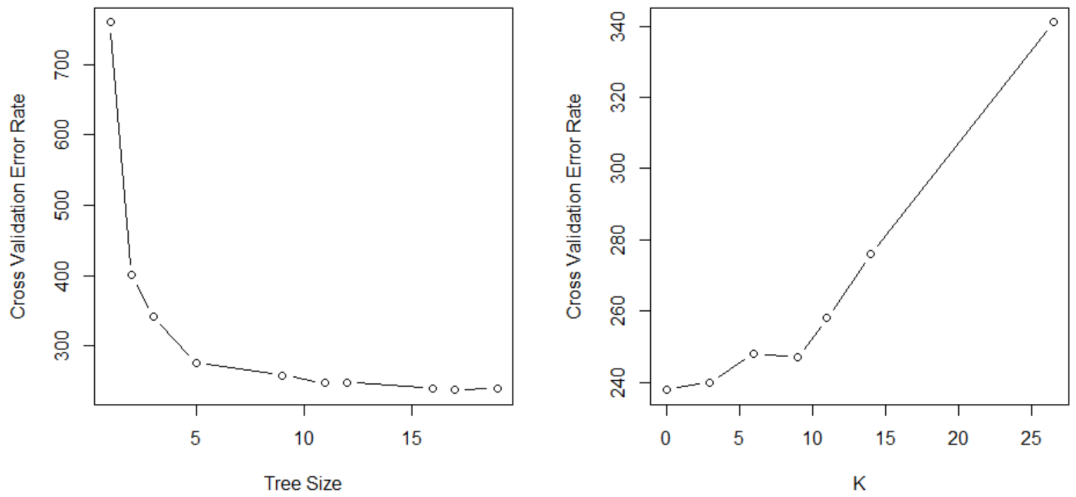
## 4. Tree-based Models

We use tree-based models to segment the predictor space into a number of simple regions. Their advantage is the useful and simple interpretation, but they usually don't deliver the best model. Decision trees can be applied to both regression and classification problems.

4.1 Classification Tree

The predicted response for an observation in a classification tree will be estimated by the most commonly occurring class of the training observations in the region to which it belongs. Besides the class prediction we will have a look at the class proportions among the training observations that fall into that region.

As the RSS cannot be used as a criterion for the binary splits as in a regression tree, we use the classification error rate which is the fraction of the training observations in that region that do not belong to the most common class.
The tree() function uses 16 variables to determine 18 terminal nodes. The total residual mean deviance is 56,08%. The misclassification error rate for the training set of 12,13 % lies below the performance of other models we were using before.
We then use the classification tree model to predict values for our test set. The model predicts 83,20% of the images correctly in terms of whether their number is even or odd.




As a next step we use Cross Validation to improve the results of the classification tree. The two graphs above show the CV error rates in relation to tree size and K (corresponding to $\alpha$). We observe than the CV error rate is strongly decreasing when we increase the tree size up until 5 terminal nodes. Afterwards the improvement slows down. Considering the graphs, we choose to prune the tree to best=9 terminal nodes instead of 18 which the original tree suggested. In fact, in my model the tree pruning decreased the model accuracy to 80,80% instead of increasing it which should not be the case, also tried tree sizes 11 and 13 but both didn't result in a better result than the original classification tree.

4.2 Bagging

To improve the performance of tree-based models, bagging, random forests and boosting are very powerful tools. Bagging in particular helps us to reduce the variance of statistical methods. The idea is to take many training sets from the population, building a separate model using each training set and averaging the resulting predictions. The different training sets get generated using bootstrap. In the case of a classification tree bagging uses a majority vote to predict the most likely class value.
We are calling the randomForest() function using 1000 as the number of trees and 247 variables at each split for. I chose the number of trees based on observation. As a result, we retrieve an estimated OOB error rate of 8,14% as well as a Test set error rate of 8,4%. Compared to the error rates in the initial classification tree, bagging improved the model significantly.
We also have a look at the variable importance based on the Mean Decrease Gini Index. The plot shows that Pixel_271 is the most important in the data set with a value of close to 100, then there are four pixels (242, 213, 299 and 270) which are also sticking out with values

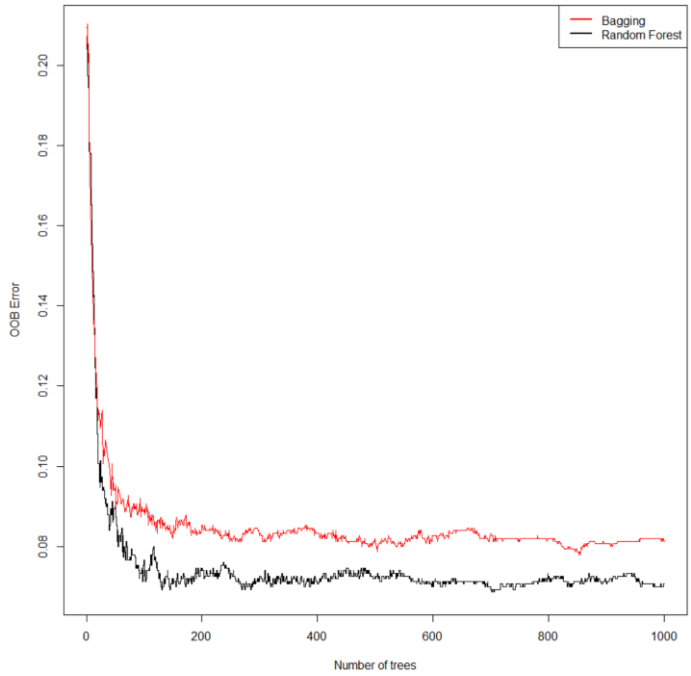
between 25 and 45 before all other displayed variables have values close to each other between 2 and 20. This makes sense because Bagging uses predictors at each split in the same order whereas Random Forests chose a random set of predictors at each split.

### 4.3 Random Forests

Random Forests follows the same idea as Bagging just that it decorrelates the trees which means that each time when a split in a random tree is considered we use a random set of predictors chosen from all predictors.

Again, we use 1000 trees and the function tried 15 variables at each split. As opposed to Bagging the variable importance values (Mean Decrease Gini Index) vary between 5 and 20 now which makes sense because the variable selection at each split happens randomly.

The results show an improvement of the model. The model accuracy increased to 93%. The following graph shows a comparison between the two methods Bagging and Random Forests according to the OOB Error based on the number of trees. We can tell that regardless of the number of trees Random Forests outperforms Bagging and reduces the OOB Error by around 1%.




### 4.4 Boosting

While Bagging builds each tree on a bootstrap data set, independent from the other trees Boosting grows its trees sequentially upon the information from the previously grown tree. Instead of using bootstrap to sample from the data, each tree is fit on a modified version of the original data set.

To create the Boosting model, we chose the following tuning parameter: B = 100 is the number of trees which we try to keep reasonably small to avoid overfitting, for the shrinkage parameter $\lambda$ we would use 0.01 which is a typical value – smaller values would require a very large number of trees. The third variable would be d the number of splits in each tree which

controls the complexity of our model. We would choose d=1 which often works well according to literature and creates "stumps" – trees which consist of a single split.

The H2O package includes a boosting function which is easier to use and reveals more information. We use the default values for d and $\lambda$ and just vary the number of trees.

| Number of trees | Training Error | Test Error | Model Accuracy | Gini Index (Test) |
|---|---|---|---|---|
| **B=50** | 7,73% | 6,59% | 93,31% | 95,13% |
| **B=70** | 7,21% | 6,19% | 93,81% | 95,67% |
| **B=100** | 6,73% | 5,69% | 94,31% | 96,02% |
| **B=1000** | 5,67% | 5,00% | 95,00% | 95,80% |

Based on the values in the table above we choose B=100 which has a very high model accuracy and the highest Gini Index or node purity which means that one node contains 96,02% from one class. As a result, we retrieve a 94,30% prediction accuracy for the model – so far - our best result. If we increase the number of trees to B=1000 we can predict with 95,00% accuracy but sacrifice a small percentage of the node purity and need far more computing power.

## 5. Support Vector Machines (SVM)

SVMs are a generalization of a simple and intuitive classifier - the maximal margin classifier – which was extended by the support vector classifier that can be applied in a broader range of cases and the support vector machine that accommodates even non-linear boundaries.

### 5.1 Maximal Margin Classifier & Support Vector Classifier

Based on the given problem we can already state that the predictor pixels are non-linear and this is why there will be no separating hyperplane. We will not use the Maximal Margin Classifier but have a look at Support Vector Machines which have greater robustness when it comes to individual observations and will perform better observations of most of the training observations.

The Support Vector Classifier or Soft Margin Classifier is the generalization of the maximal margin classifier for the non-separable case. It allows for some observations not only on the wrong side of the margin but also of the hyperplane. Observations on the wrong side of the hyperplane are training observations that are misclassified by the Support Vector Classifier. Still the Support Vector Classifier is a method for close to linear problems which is why we directly move on to the Support Vector Machine.

### 5.2 Support Vector Machine (SVM)

In this report we are faced with non-linear class boundaries. The Support Vector Machine allows us to enlarge the feature space to accommodate a non-linear boundary between the classes. Similar to the Support Vector Classifier problem we use the inner products of the observations instead of the observations themselves. We are enlarging the feature space by using polynomials which allows us to use non-linear decision boundaries in the original space but a linear hyperplane in the transformed feature space. Key to the calculation of inner products is the kernel function which quantifies the similarity between two observations. For our data set we use kernel "radial" which takes the form

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

And a constant positive $\gamma$. What it does in essence is that if a given test observation is far from a training observation in terms of Euclidean distance, then the sum in the formula will be large and therefore the result K will be tiny. Therefore, training observations that are far from the test observation will not impact the predicted class label for the test observation. The advantage over the support vector classifier is computational.

It took a few tries to get results from the svm() function that actually made sense and the reason therefore was that the gamma values I chose where too large. I also suspected that the feature space is too large and SVMs would not easily find the structure and suffer from having many dimensions to search over (Curse of Dimensionality).

First, I tried $\gamma$ values of 0.5 which resulted in support vector machines with 1500 support vectors (which is the number of observations in my training set). I varied the values manually until I got 639 support vectors for C = 1 and $\gamma$=0.01. One might ask why I did it manually in the beginning before using the tune function. The reason is that the tuning need very long to compile and I wanted to get a feel for the range of values which would make sense to run through. Literature suggested that usual values for $\gamma$ would be around 0.5 here it didn't return any results that help us understand the predictors better.

To tune the model, I used the following values for the cost parameter: 0.1, 1, 10, 100; and $\gamma$ parameter: 0.01, 0.1, 0.5, 1. The best performing parameters are C = 10 and $\gamma$ = 0.01 with an error rate of 4.20% on the training set and 5.60% on the test set.

## 6. Neural Networks

Neural Networks is a powerful learning method that extracts linear combinations of the inputs as derived features and then models the target as a nonlinear function of these features. We use the h2o package to model the feed-forward neural network for this report. The goal of neural networks is to minimize the cost function J using the Gradient Decent Algorithm. The h2o.deeplearning() function requires several parameters. A parameter we need to choose is the number of hidden layers and the number of neuron within each hidden layer. Literature states that neural networks rarely improve using more than one hidden layer. The size of the hidden layer is defined by the number of neurons. We generally choose a number that lies between the number of input and output variables. One rule of thumb is to use two thirds of the size of input layer plus the size of the output layer. Therefore, we choose a size of 165 neurons in the hidden layer. The activation functions "Tanh" (Sigmoid) and "Rectifier" seem to be the most popular ones in literature. For now, we don't add a dropout rate and use the activation method "Tanh" which we then compare with a model using "Rectifier" and using a dropout rate of 0.5 which is most commonly used. Dropout is a hyperparameter used to prevent overfitting. It randomly sets nodes in the neural network to zero and thereby encourages the network to rely on other features that act as signals. For cross-validation we use 10-folds and we tell the model to stop after 1000 iterations (epochs). The output of h2o is

very insightful. To compare the model with different ones we have a look at the test and training errors as at the cross validation means of the Model Accuracy, the area under the curve (AUC) and the LogLoss metrics.

| Activation Function | Training Error | Test Error | Accuracy (CV-mean) | AUC (CV-mean) | Logloss (CV-mean) |
|---|---|---|---|---|---|
| **Tanh** | 9.00% | 9.40% | 91.73% | 96.44% | 31.46% |
| **Tanh with Dropout** | 8.67% | 10.00% | 92.40% | 95.57% | 23.80% |
| **Rectifier** | 7.14% | 8.48% | 94.02% | 97.89% | 25.55% |
| **Rectifier with Dropout** | 5.26% | 6.80% | 95.40% | 98.50% | 20.56% |

The activation function ‘‘Rectifier with Dropout’’ returns the best results and we use it for comparison with other models in the next section. Another reason therefore is the standard deviation that is below 1%.

Another way to train neural networks is to feed it teaching patterns and letting it change its weights according to some learning rule. Some error-based learning methods are the Perceptron Rule and the Error Correction Gradient Descent. The Perceptron Rule works for Neural Networks where both the inputs and the outputs are binary. The error Correction Gradient Descent or Backpropagation works for a multilayer network and attempts to minimize the square errors.

H2o does some things for us that we would usually use to improve our model such as choosing the regularization parameter $\lambda$ which would help us to modify bias and variance.

## D) Assessing model accuracy

In order to compare the different models, we selected a number of metrics used to assess model performance. The training and test error are the most straight forward methods and easy to imagine. They are simply calculated as the inverse of the model accuracy of the test and training set. The model accuracy in many cases was retrieved as the cross-validation mean of the accuracy.

The Receiver Operating Characteristic (ROC) curve gives an indication of the performance of a classifier over the entire range of decision rules and thereby helps us to compare different classifiers in a more rigorous way. The way to compare the performance based on the ROC curve is the area under the curve (AUC) metric which typically takes values between 0.5 and 1. Values outside that range indicate that the model might have been set up incorrectly. Lastly the Log Loss metric is sought to be low. Is used to penalise classifiers that are confident about an incorrect classification. This means that if a classifier assigns a very small probability to the correct class for a certain observation then the corresponding contribution to the Log Loss value will be very large. In an ideal model the Log Loss value would be 0.

| Method | Training Error | Test Error | Accuracy (CV-mean) | AUC (CV-mean) | Logloss (CV- mean) |
|---|---|---|---|---|---|
| **Logistic Regression** | - | 21.30% | 78.70% | 85.13% | - |
| **K-nearest Neighbors** | - | 5.60% | 94.20% | | - |
| **Classification Tree** | 12.13% | 16.80% | 83.20% | | - |
| **Bagging** | 8.14% | 8.40% | 91.60% | | - |
| **Random Forests** | 7.07% | 7.00% | 93.00% | | - |
| **Boosting, B=100** | 6.73% | 5.69% | 94.31% | | - |
| **SVM** | 4.20% | 5.60% | 94.40% | | - |
| **Neural Net** | 5.26% | 6.80% | 95.40% | 98.50% | 20.56% |

Overall the best performing model is the Neural Networks algorithm. The accuracy of 95.40% and the AUC value of 98.50% are outperforming all the other models although there are several that reach similar performances like Support Vector Machines, Boosting and K-nearest Neighbors.

After deciding for Neural Networks as the best model I predicted the values for the unseen test data with the h2o.predict() function and saved the values in a .csv file which is attached to the report. The overall prediction was that 1304 numbers are odd (output = 1) and 1196 are even (output=0) which seems to be a realistic ratio.

## E) Conclusion

Building a model is an art, it is an endless process and there will always be more methods to explore and play around with. The goal of this report was to explore the different methods covered in class and compare them amongst each other. I believe that neural networks give a good result in this case and it is also the most commonly used methods for image recognition. Going forward I would like to visually compare the different models using ROC curves or boxplots of the error rates.

## References

*''Introduction to Statistical Learning'' by Gareth James, Daniela Witte, Trevor Hastie and Robert Tibshirani*

*''The Elements of Statistical Learning'' by Trevor Hastie, Robert Tibshirani and Jerome Friedman*