Automatic Classification of Image Data Through Application of RandomForest

MATH 6350 - Statistical Learning and Data Mining

Ravik Chand
Lucas Smith

MATH 6350 - Statistical Learning and Data Mining Ravik Chand
Project 4 Report Lucas Smith

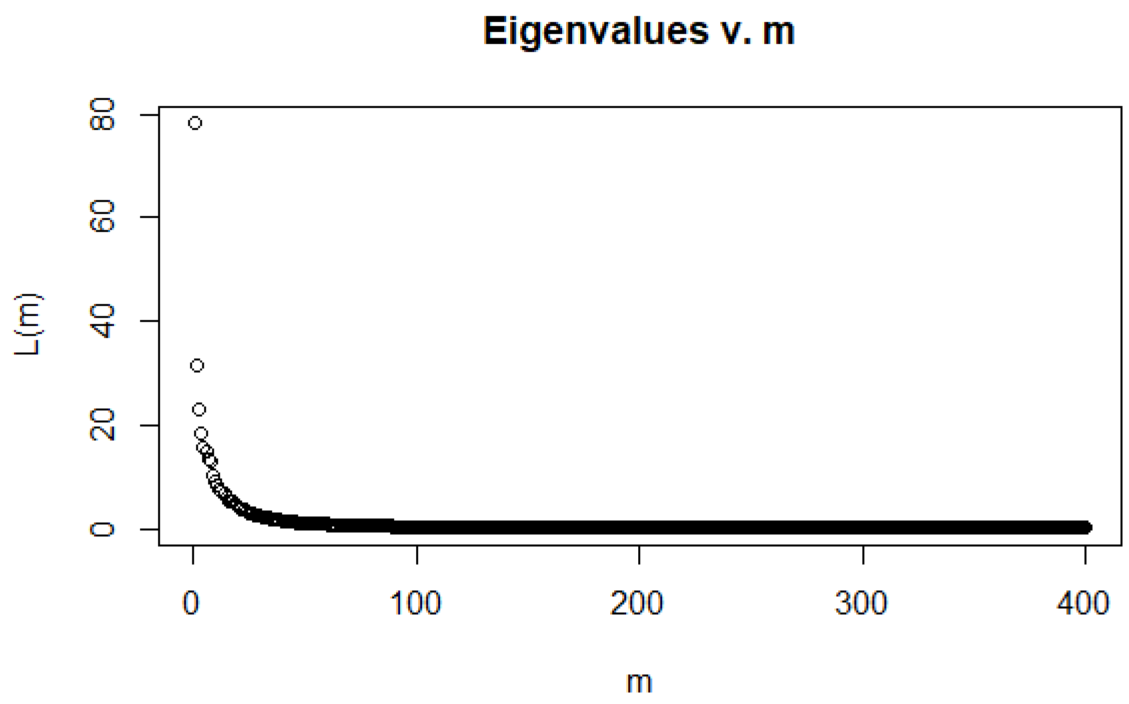
The objective of Project 4 was to create a model that could classify fonts through the application of Random Forest. The data used was sourced from the University of California Irvine's Machine Learning Database archive. Six .csv were selected containing character data for the fonts, "Bauhaus," "Jokerman," "Magneto," "Agency," "French," and "Palace." Each file contained nearly 1000 character observations divided into 412 feature columns. The first three columns included the font identity, its strength which defined whether the character was bold, and a binary identifier of italicization. The final four hundred dimensions described the digitized image of the character by the gray value of each pixel contained within a 20x20 array.

After initially importing into RStudio as *font1*, *font2*, *font3, font4, font5,* and *font6*, these datasets were trimmed down to the 403 columns relevant to this exercise using the subset function. Cells containing missing values were replaced with NULL, then all observations with NULL values were filtered out. Six classes *cl1*, *cl2*, *cl3, cl4, cl5,* and *cl6* were created by again using the subset function to isolate character data that was bold with a strength value of 0.7 and not italicized with a italic value of 0. These classes contained about 245 observations each and would contribute roughly 16.6% of the total entries to the exercise thus mitigating any skewness towards a particular font. After this, the columns which identified boldness and italicization were also removed. The six class datasets were unified into a singular dataset, *font.data*, containing 1472 observation rows and 401 feature columns.

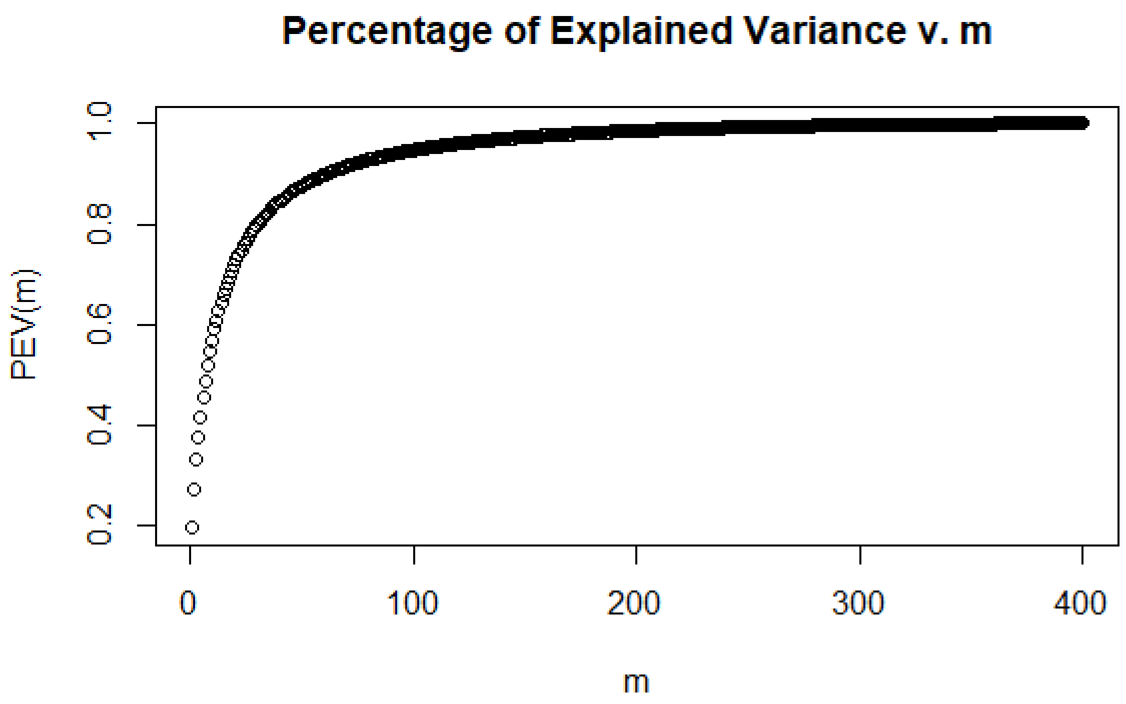
The mean and standard deviation was computed for each numerical feature column by application of a for-loop and deposited into respective *mean.x* and *sd.x* vectors, each of length 400. Then the rescale function was applied to normalize *font.data*. The column identifying the font of each column was rendered NULL and removed as well. The resulting 1472x400 matrix of rescaled features became *RESF.*

A 400x400 matrix *RESF.cor* was created using the cor function on the *RESF* matrix. From *RESF.cor*, the eigen function was applied to create the *RESF.ev* matrix from which vectors *L* and *W* were derived, containing eigenvalues and eigenvectors respectively. A plot of eigenvalues against the count of values in vector *L* was generated as below.

Ravik Chand
Lucas Smith



The percentage of explained variance (PEV) was calculated for every value in vector *L* as the cumulative sum of values divided by the length of *L* against the count of values in *L* which can be observed in the plot below. Of the 400 values in *L*, the minimum value, *threshold*, which captured 97.5% of variance was determined to be 157. The matrix *redW* was created by taking the first 157 columns from the eigenvector matrix *W*.



The matrix multiplication of matrices *RESF* and *redW* results with the 1472x157 matrix *Z*. Matrix *Z* contains new features vectors created from linear combinations of the rescaled features and eigenvectors.

The underlying principle of the Random Forest algorithm is rooted within the tree classifiers that compose the forest. Within each tree classifier, there are multiple levels of decision nodes. Each decision node-k performs a binary split into a node-k1 and node-k0 based on a feature such that observations categorized within node-k1 are similar to one another yet dissimilar from those categorized into node-k0, and likewise for node-k0. The process continues for each subsequent level of decision nodes and moves towards class purity after each binary split. Random Forest generates **n** trees, through bootstrap aggregation in which each tree samples observations randomly (with replacement) from the given data set. Additionally, each decision node-k categorizes observations based on a subset **s** of features, which has a count no greater than the square root of the original count of **p** features. In doing so, the Random Forest algorithm mitigates correlation between trees. The **n** count of trees can be optimized for accuracy however higher tree counts will significantly impact compute time. The forest calculates its ultimate classifications by accepting final decisions as votes from all trees. In doing so, the algorithm leverages the multiplicity of trees to compensate for the errors that may occur in individual trees.

The out of bag (OOB) accuracy refers to the rate at which a specific tree within a Random Forest successfully classifies an observation that was not part of its original sample.

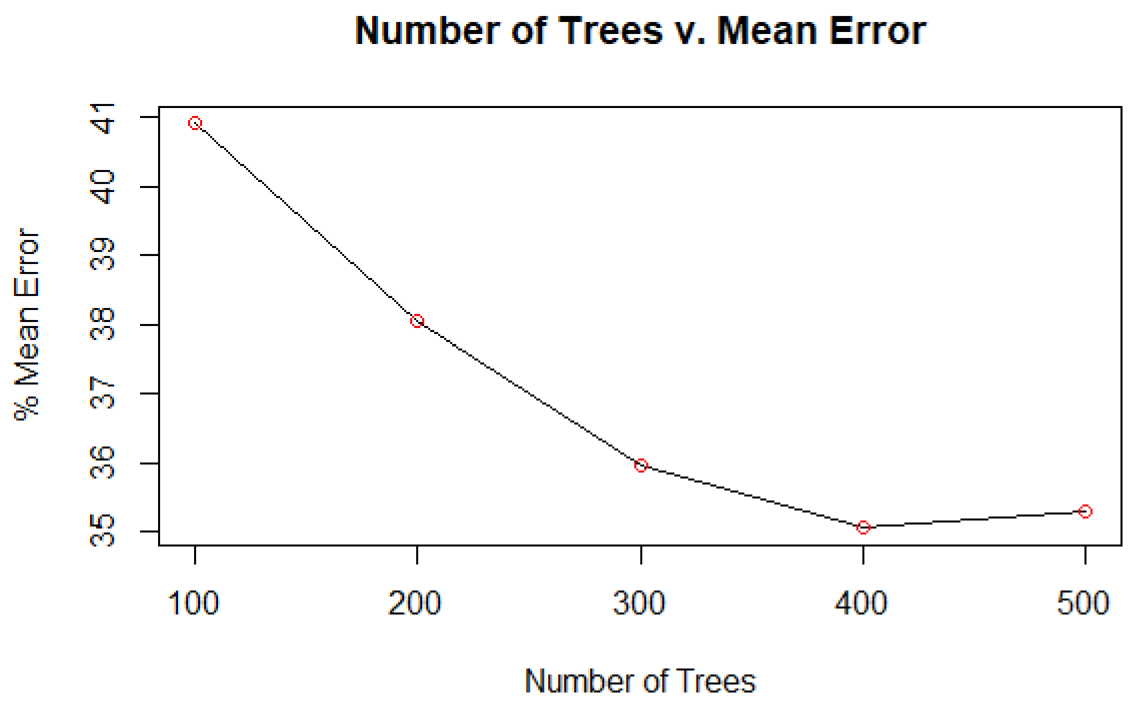
The first input required by the randomForest function in R is a formula for the model being fitted. This is composed of column vectors corresponding to a response variable followed by a predictor variable or series of predictor variables. Alternatively, the function can be provided with a data set **x** and a response vector **y**. The next input is the data set containing the variables that will be applied in the model, which would default to NULL if already provided. The third input is the data set containing the test predictors, followed by the data set containing the test response vector. Both these may be NULL by default. The final vital input is the count of trees that will be generated by the function.

The vector *Zlabel* was created from the first column of *font.data* to hold the correct classifications of all observations. The object *model1* was then built using the randomForest function to automatically classify the 1472 observations in matrix *Z*. The model was provided with a formula specifying *Zlabel* as a response variable, and all column vectors of matrix *Z* as predictor variables. The tree count was set to 100. The value *n100err* was calculated as the difference between 1 and the average error rate of all trees in *model1*, multiplied by 100 to yield the percentage out of bag accuracy of the model.

The tree count parameter of the randomForest function was modified in 100 tree increments from 200 to 500 trees, with each increase adding about 1.6 seconds to compute time. The out of bag accuracy was calculated for the varying tree counts and deposited into the following table.

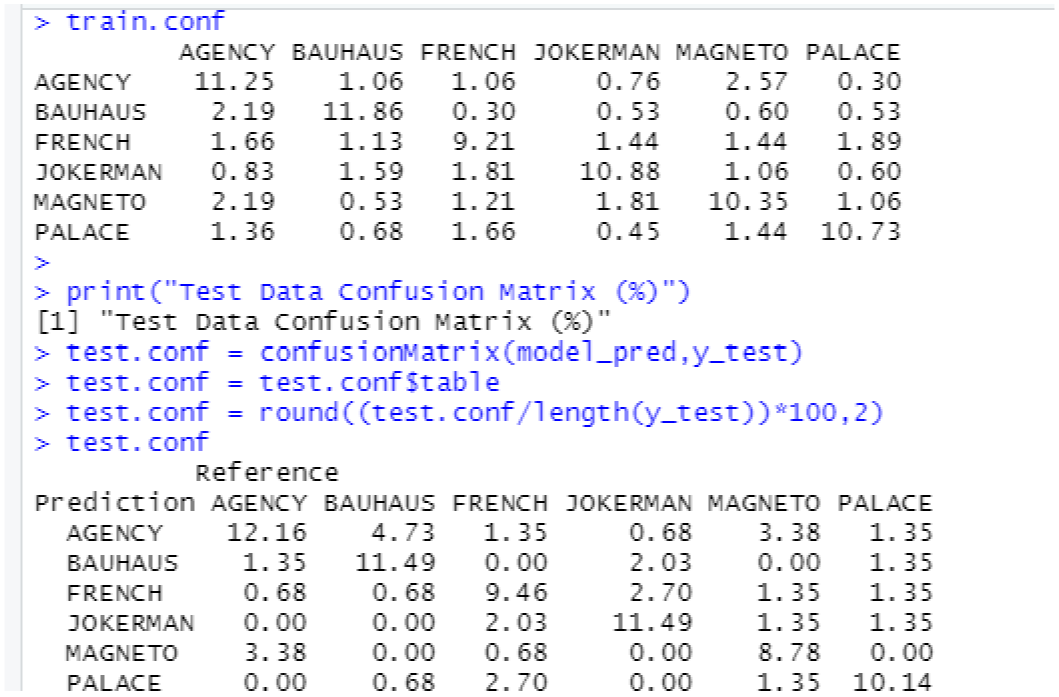
Trees	Accuracy(%)	Compute Time (sec)
100	59.088	1.61
200	61.639	3.24
300	63.799	4.77
400	64.081	6.42
500	64.011	8.00

Based on the below graph, the optimal tree count was determined to be 400. The n=300 tree classifier could also be used, but it was determined that the additional accuracy at the cost of less than 2 seconds of computation time was worth the selection of n=400.



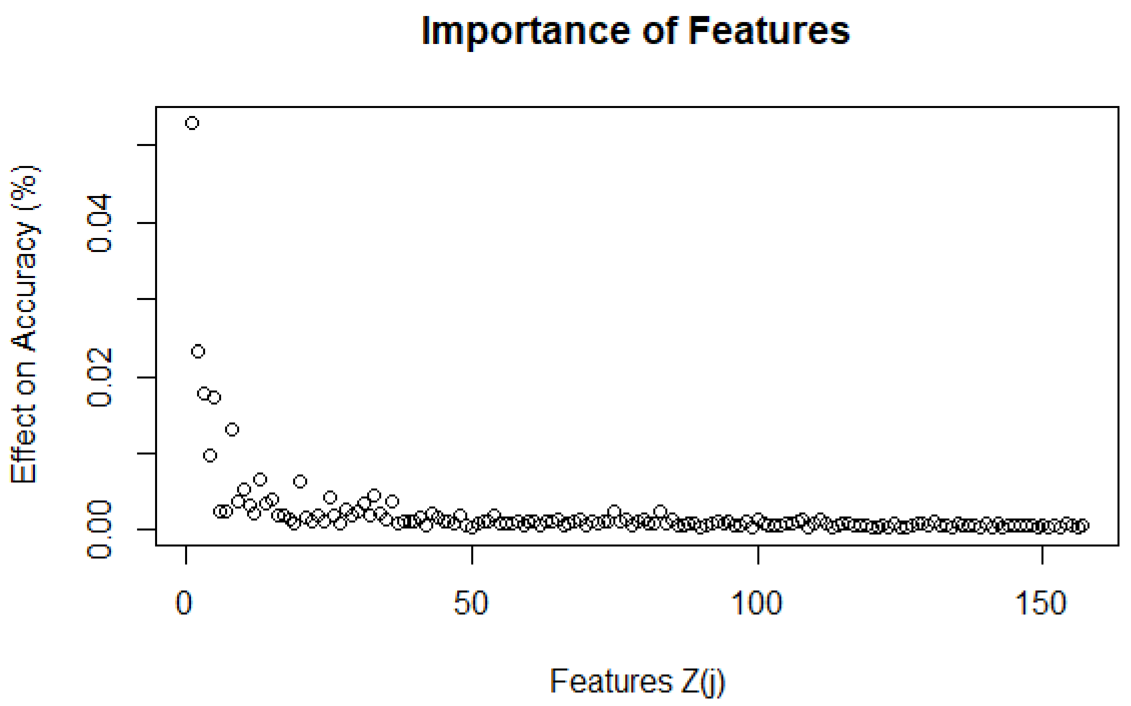
A series of integers equal to 90% of the length of matrix *Z* was generated and used to sample corresponding observations into the index *idx*. This index was used to pull observations from matrix *Z* into matrix *x_train* which would serve as a training set of data. The remaining 10% was allocated to matrix *x_test* to serve as the test set. The index *idx* was similarly applied to the vector *Zlabel* to create the vectors *y_train* and *y_test*. The object *model.opt* was built using the previously determined optimal tree count, with *y_train* as the response variable and the column vectors of *x_train* as the predictor variables.

The confusion matrices of RF** were computed on both the training set and the test set. For the training set, class CL2 “Bauhaus” has the highest rate of successful categorization, while class CL3 “French” has the lowest level of successful categorization. For the test set, class CL1 “Agency” has the highest rate of successful categorization, while class CL5 “Magneto” has the lowest level of successful categorization.

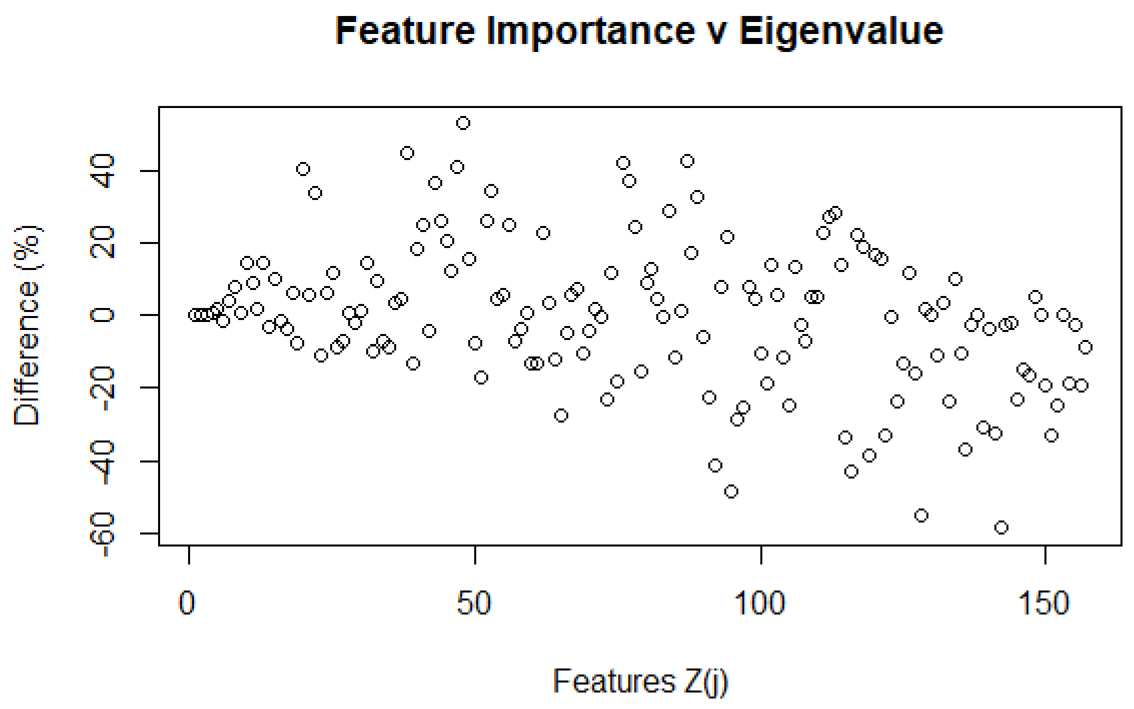


```
> train.conf
         AGENCY BAUHAUS FRENCH JOKERMAN MAGNETO PALACE
AGENCY    11.25    1.06   1.06     0.76    2.57   0.30
BAUHAUS    2.19   11.86   0.30     0.53    0.60   0.53
FRENCH     1.66    1.13   9.21     1.44    1.44   1.89
JOKERMAN   0.83    1.59   1.81    10.88    1.06   0.60
MAGNETO    2.19    0.53   1.21     1.81   10.35   1.06
PALACE     1.36    0.68   1.66     0.45    1.44  10.73
>
> print("Test Data Confusion Matrix (%)")
[1] "Test Data Confusion Matrix (%)"
> test.conf = confusionMatrix(model_pred,y_test)
> test.conf = test.conf$table
> test.conf = round((test.conf/length(y_test))*100,2)
> test.conf
          Reference
Prediction AGENCY BAUHAUS FRENCH JOKERMAN MAGNETO PALACE
  AGENCY    12.16    4.73   1.35     0.68    3.38   1.35
  BAUHAUS    1.35   11.49   0.00     2.03    0.00   1.35
  FRENCH     0.68    0.68   9.46     2.70    1.35   1.35
  JOKERMAN   0.00    0.00   2.03    11.49    1.35   1.35
  MAGNETO    3.38    0.00   0.68     0.00    8.78   0.00
  PALACE     0.00    0.68   2.70     0.00    1.35  10.14
```

The outputs of the randomForest function included a matrix *importance*, which presented the effects of each feature on classification accuracy. The seventh column of this matrix displays the average difference between the accuracy of a prediction in relation to a specific feature, and the prediction accuracy when that feature is randomly altered. This was used to generate the vector *model4.imp* which demonstrated the importance of each feature for classification. The below graph demonstrates the percentage of variance that can be explained by a particular feature. When sorted in descending order of importance, variations appear.



Feature importance as determined by the Random Forest algorithm varies in order from PCA-importance. The below graph demonstrates the percent difference in ranking of importance as determined by Random Forest and PCA. The greatest difference noted was with feature Z50. In contrast, there was no difference in ranking for features Z1, Z2, Z3, and Z153.


Feature Importance v Eigenvalue
Difference (%)
Features Z(j)
-60
-40
-20
0
20
40
0
50
100
150