Jeff Levesque

Professor Gates

IST 565

Homework #7

**Introduction**

Today many algorithms exist to perform classification techniques. However, not all are appropriate for the task at hand. Additionally, choosing the most appropriate algorithm is often not a trivial task. Various factors including accuracy, and computational time, are factors requiring careful consideration. In data science, concepts of aggregating data, visualization, and generation of models for predictive and prescriptive analysis are major areas of focus. However, sometimes concepts of big o are forgotten.

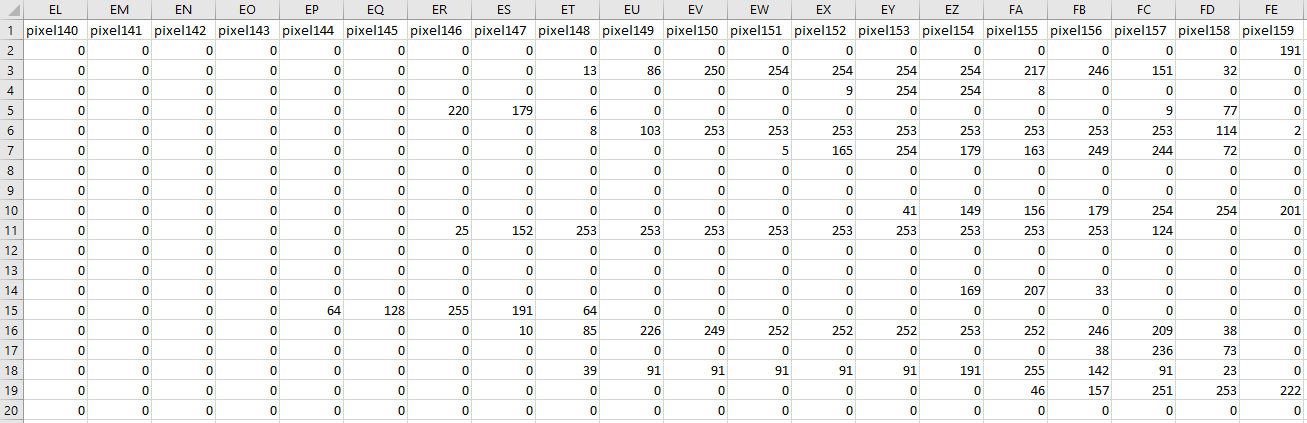
Having prepackaged libraries in python or R, consolidates code, allowing functions to be easily implemented. Examples of these are numerous including the R’s e1071, and python’s sklearn. Since machine learning algorithms can quickly get complicated with calculations and ensembling, sometimes the easiest way to determine performance, is to determine the runtime of a function. This can be done by comparing the runtime of multiple functions by passing the same dataset(s). Therefore, multiple techniques, such as classification algorithms, can be compared for accuracy, as well as runtime performance.

In this study, various models will be fitted using support vector machines (svm), k-nearest neighbors (knn), and random forests. Specifically, these classification algorithms will attempt to predict numeric values, represented as a series of pixels, where numeric values indicate color darkness. Prediction accuracy will be compared between the three algorithms, as well as runtime performance. These results will then be compared to previous findings, regarding predicting numerical text, with respect to naïve bayes and decision trees.

**Analysis**

Data Preparation:

Two csv datasets representing instances of numeric digits ranging from 0-9 are used. More generally, each numeric digit was represented by a sequence of (roughly 780) pixels. Each pixel contains a numeric value associated with the pixel density. Therefore, higher values, are associated with darker pixels. The original train dataset contained 42,000 records, where each row represented a numeric instance. Conversely, each column represented a pixel density. The original test dataset was similarly structured, containing 28,000 numeric records.



Since pixel 0 through pixel 11, and pixel 780 through 783 of the train, and test dataset contained a column sum of zero, they were removed from the corresponding dataframe:

## remove redundant pixels

delete = c('pixel0', …, 'pixel11', 'pixel780', …, 'pixel783')

df.train = df.train[, !(names(df.train) %in% delete)]

df.test = df.test[, !(names(df.test) %in% delete)]

Additionally, certain algorithms performed significantly slower than expected. Therefore, a random\_sample function was created within the custom loadPackage library:

random\_sample = function(df, n) {

return (df[sample(nrow(df), n),])

}

This function was used to decrease the provided train, and test dataset to 10% of the original size. Specifically, the train, and test dataset was randomized set containing 10% of the original corresponding dataset.

**Results**

Once the dataframe refactored, the e1071 package was used to compute the svm model. Using the subsample dataset, with the default radial model, and a 3-fold CV, R console generated a warning after substantial time passed with default scale=True:

WARNING: reaching max number of iterations

Therefore, the scale=False was applied to the svm 3-fold CV model. Since labels were not included in the test dataset, validation was done on the train dataset. Additionally, the computation time for the svm was roughly 16x greater than the previous study using decision tree:

===========================================================

svm, knn, random forest performance (minutes)

===========================================================

[1] "fitting svm: 2.24219071467717"

[1] "predicting svm: 25.8875329494476"

[1] "knn model + prediction: 2.91631125211716"

[1] "random forest model + prediction: 5.38949536482493"

===========================================================

decision tree performance (minutes)

===========================================================

[1] "fitting tree: 2.67286456425985"

[1] "predicting probability: 1.54783892631531"

[1] "predicting class: 1.58008599281311"

===========================================================

naïve bayes performance (minutes)

===========================================================

[1] "fitting tree: 7.53572297096252"

[1] "predicting probability: 53.0671591758728"

[1] "predicting class: 59.0970938205719"

Also, the svm model was built using 10% of the train, then prediction used 10% of the test dataset. It was evident that the prediction was computationally expensive. On a similar note, the resubsitution error for the svm was 90%, indicating the model was only 10% accurate, compared to the decision tree at 64% (and naïve bayes at 53%).

===========================================================

confusion matrix:

===========================================================

fit.svm.class 0 1 2 3 4 5 6 7 8 9

3.43729863692693 402 471 415 426 0 0 0 0 0 0

4.09979863692693 0 0 0 0 295 0 0 0 0 0

4.10057988692693 0 0 0 0 128 0 0 0 0 0

4.89979863692693 0 0 0 0 0 304 0 0 0 0

4.90057988692693 0 0 0 0 0 80 0 0 0 0

5.43729863692693 0 0 0 0 0 0 430 435 369 445

===========================================================

resubstitution error:

===========================================================

[1] 0.9042857

When generating the knn model, the class package was utilized. However, unlike the svm, decision tree, and naïve bayes, validation accuracy was not performed on the train dataset. This was due to the fact that the computed model returned a result with the same dimension as the test dataset. Therefore, the model could not be compared directly with previous ones, unless the train dataset was partitioned for a dedicated validation set. However, this would require the other models to similarly implement. Given these limitations, only the computational performance was directly compared. Specifically, computing the decision tree took roughly four minutes (fitting and prediction), while the knn was roughly 2.88 minutes. Therefore, knn is marginally more performant than decision trees, and therefore more performant than naïve bayes.

Finally, when generating the random forest with 30 trees, the full dataset was used during the fit. This was chosen, since computing a model using the full dataset took roughly five minutes, much tolerable than the svm case, which only used 10% of the original data. Additionally, the validation result was encouraging, since the random forest error of 5.68%, indicates that the model was roughly 94% accurate.

Call:

randomForest(formula = as.factor(label) ~ ., data = df.train.full, ntree = 30)

Type of random forest: classification

Number of trees: 30

No. of variables tried at each split: 28

OOB estimate of error rate: 5.68%

Confusion matrix:

0 1 2 3 4 5 6 7 8 9 class.error

0 4031 0 13 6 8 16 23 3 29 3 0.02444337

1 0 4601 30 13 10 2 3 8 12 5 0.01771990

2 22 15 3940 34 27 7 24 44 50 14 0.05673929

3 10 12 82 3989 4 103 12 44 67 28 0.08319926

4 7 9 14 4 3865 5 22 20 20 106 0.05083497

5 26 10 7 117 19 3494 38 8 48 28 0.07931489

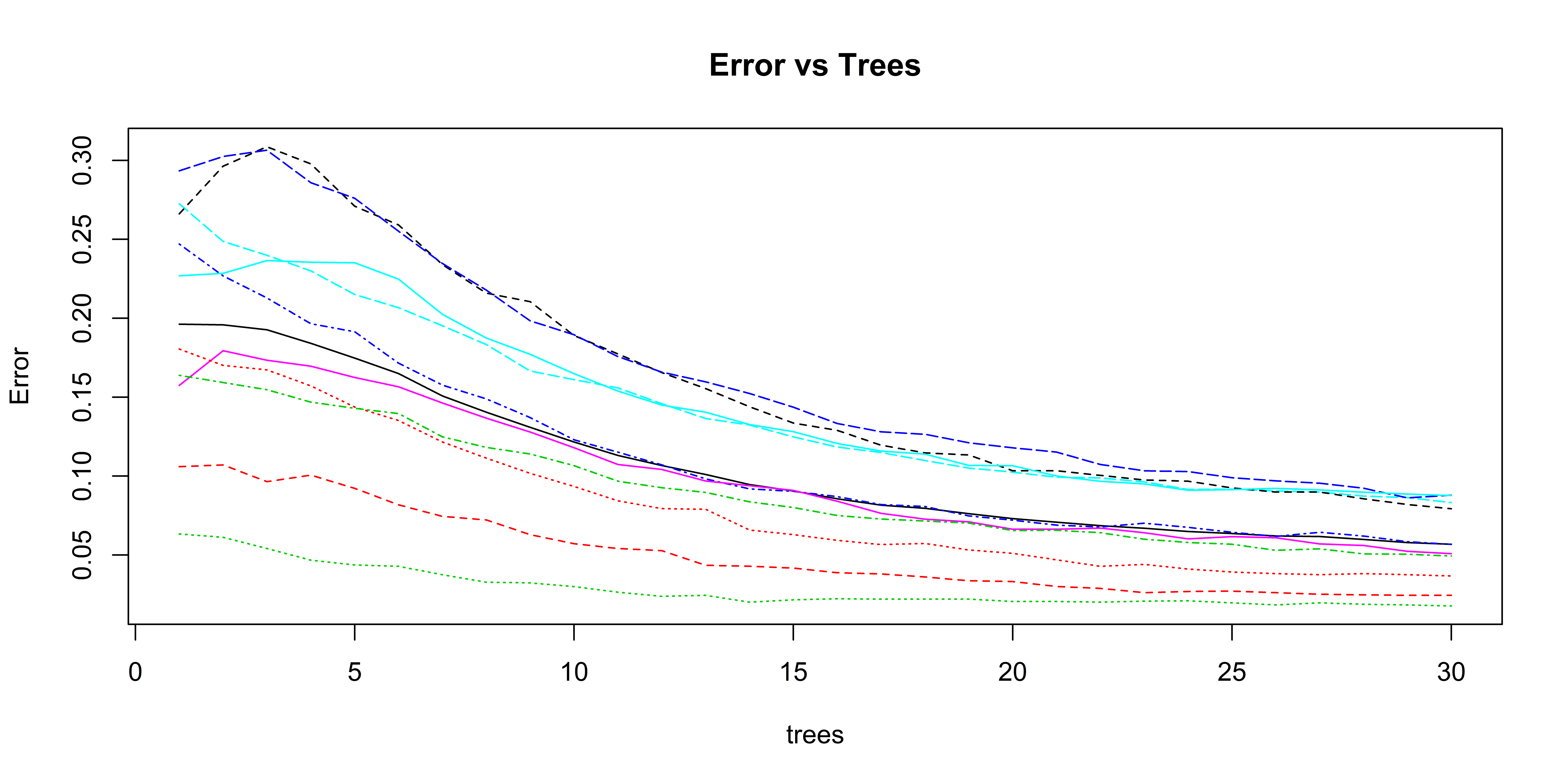
6 29 15 11 2 18 49 3985 1 24 3 0.03674160

7 5 22 62 13 32 6 1 4184 14 62 0.04930698

8 20 25 61 66 36 64 18 15 3706 52 0.08786611

9 19 14 18 62 114 26 8 58 48 3821 0.08763133

As the number of trees grow in the forest, the error rate decreases respectively:



Given the limitation of the knn implementation, it’s not directly possible to compare performance of this model with others. Therefore, it may be short-sided to conclude that random forest is the best performant with 95% accuracy. A more valid approach would be to create a validation partition from the train dataset, then determine the accuracy of each model using this data. However, since this was not implemented with the earlier study using decision tree, and naïve bayes, the approach was omitted in this study.

Many times, comparing one algorithm over another is not entirely straightforward. Various considerations need to be accounted for, including validation accuracy, and runtime performance. Also, the provided dataset is not always enough, and sometimes requiring the need for further partitioning. This was the case with the knn model, since unlike the other algorithms, the fit and predict occur at the same time. This means the returned result, is the set of prediction values, which is not a compatible comparison of the train dataset dimension, used by the other models. Additionally, the svm and knn model required using a subsample, whereas the random forest, decision tree, and naïve bayes were computationally fast enough to use the full original dataset.

**Conclusions**

Today there many different approaches, and packaged tools to allow end users to plug a dataset and determine a prediction result. Some of these off the shelf solutions, require more effort than others. However, these tools ultimately streamline, and make analysis much easier. Therefore, it may become too easy to regularly depend on one technique, or a fixed set of packages. This can be dangerous, as results have shown that choosing one model over the other requires careful choice, and knowledge regarding the shape of the dataset(s).

Comparing models should not be a binary comparison whether one generates a better result, or one has a better runtime. Instead users need to holistically determine an acceptable error tolerance, while understanding their computing requirements. For example, having unrealistic expectation of running a dataset through a more computationally expensive algorithm, will prove counterproductive. Therefore, users would need to consider options of subsampling, or acquiring different computing resources. Combining this knowledge can greatly increase prediction accuracy, while decreasing the prediction times.

In a previous study, it was mentioned that computing either ensemble learners, or random forest would likely perform significantly better than a single decision tree, or a naïve bayes model. This intuitive guess can easily be deduced by the fact that a random forest is a grouping of multiple decision trees. So naturally, the averaging of the trees would smooth out various edge cases. Therefore, this study has successfully provided some initial comparisons, and solidified some base intuition.