MACHINE LEARNING

Febin Zachariah – 800961027

SUPERVISED LEARNING-K-NEAREST NEIGHBOR & BOOSTING

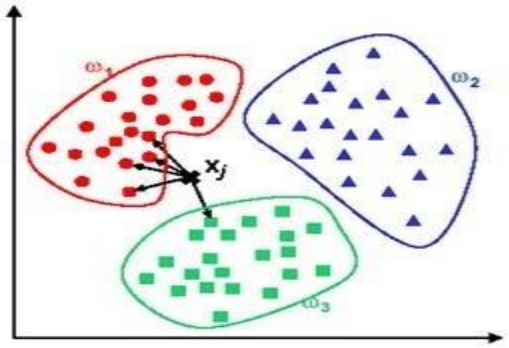
K-Nearest Neighbors Algorithm

K Nearest Neighbor algorithm (knn) is a non-parametric method used for both classification and regression. Knn is widely is used for predictive analytics. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

Main Characteristics of knn are:

1) Knn does not make any assumption on the underlying data distribution.
2) Knn is a lazy algorithm and it does not use training data to make any generalization.
3) There is no explicit training phase in knn.
4) For knn classification: An object is classified by using majority vote of its neighbors.
5) For knn regression: final value is the average of all its neighbors
6) Knn will keep all the training data in memory to perform the testing and it consumes a lot of memory.
7) k-NN assumes that the data is in a feature space.
8) Each training data has a set of vectors and class label associated with each vector. In the simplest case, it will be positive or negative. But k-NN will work well with arbitrary number of classes.
9) The number "k" in this algorithm decides how many neighbors influence the classification. The value of "k" is usually an odd number if the number of classes is 2.

Knn algorithms behaves differently with different values of k. If the value of k is 1, it is called nearest neighbor algorithm. It will just find the nearest neighbor and assigns that that class for our test variable. This scenario holds good only if the number of data points is not very large, else this procedure will lead to a huge error. For k value greater than 1, we do a majority voting and assign to the class which has the majority. Following figure shows how knn is performed in the cases of classification.



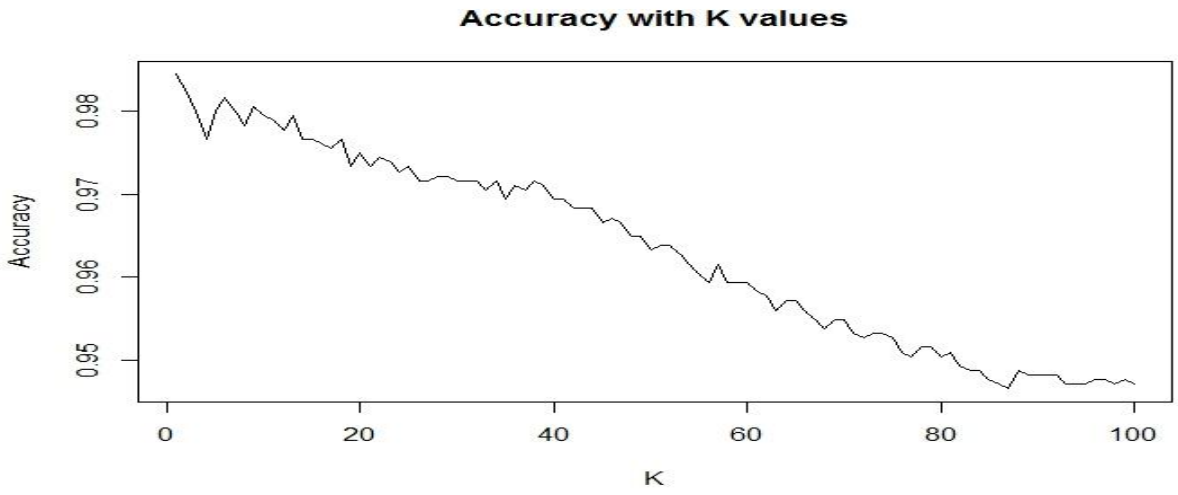
Optical Recognition of Handwritten Digits Dataset

Implementation:

- For the implementation of knn: **R language is used**
- Packages used: **class.**
- Load the packages by using **library** command in R if it is already installed, otherwise install the libraries by using **install. packages** command.
- After that load the training data by using **read.csv** command.
- The dataset for train and test data does not have a header. So, the heaters for the feature variables are names V1 to V64. The header for class variable is named V65.
- Training data is stored in the **train_data** variable.
- Test data is stored in the **test_data** variable.
- Class variables of both training data and test data is stored in the **train_class_variable** and **test_class_variable.**
- "knn" method from class library is used execute k-NN algorithm. **As parameters to the method, the train dataset, test dataset and the class variables for the train dataset is passed. This method is executed in a for loop with k value ranging from 1 to 100.** This k value is passed as parameter to the "knn" method at every iteration. We are checking with different values of k to find the accuracy variation with respect to k.
  knnTest <- knn (train_data, test_data, train_class_variable, k = i)
- There is **one more parameter named "l" passed** in the knn method of class library which needs a numeric value. Default value is 0. The value of this parameter defines the minimum vote for definite decision. We have tested with different values of l and found that better result is obtained when the value of l is 0.
- For every value of k, we have calculated and the graphs are plotted using plot function. Graphs are shown under the **Result & Observations heading.**

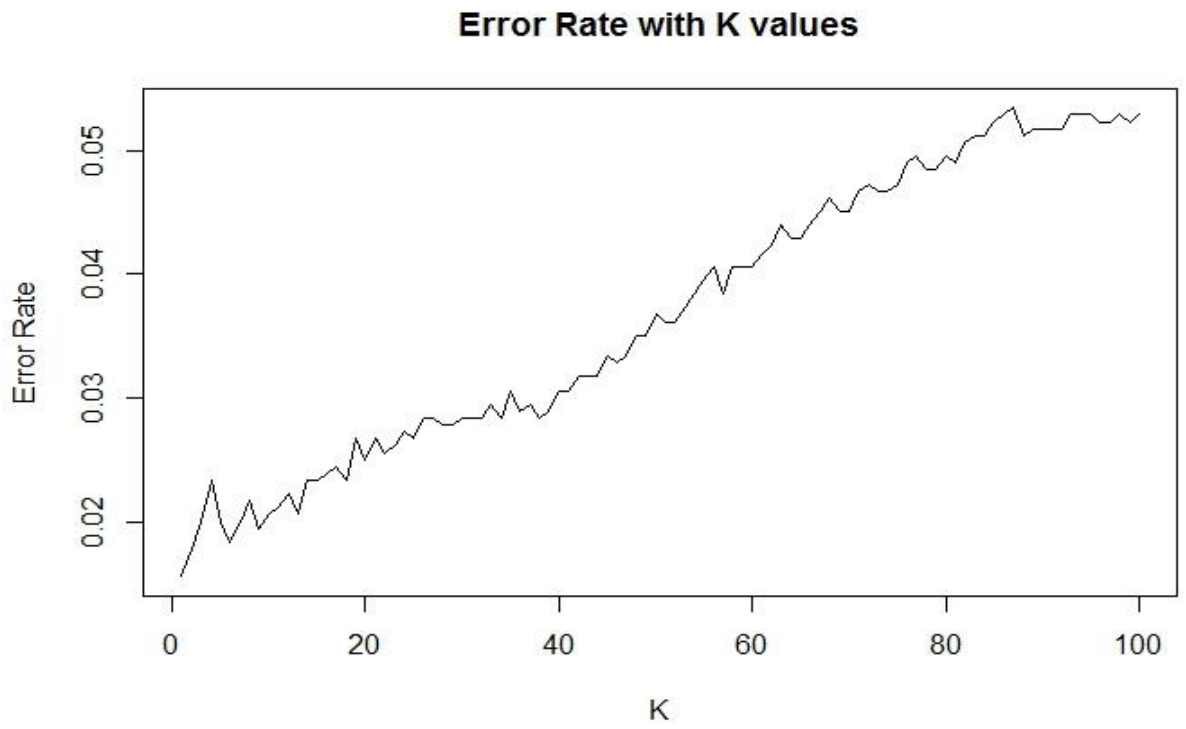
Results & Observations

Accuracy with different k values:



We can see from the above graph that the accuracy decreases when the k value increases

Error Rate Graph for different values of k:



We can see from the graph that error rate increases when the value of k increases.

Conclusions:

- We have obtained a maximum accuracy of 98.44% on the test data.
- We have constantly obtained an accuracy above 94.5% for all the values of k (1:100)
- We are getting more accuracy for small values of k.
- Successfully implemented knn algorithm by using class library.

Amazon Reviews Sentiment Analysis Dataset

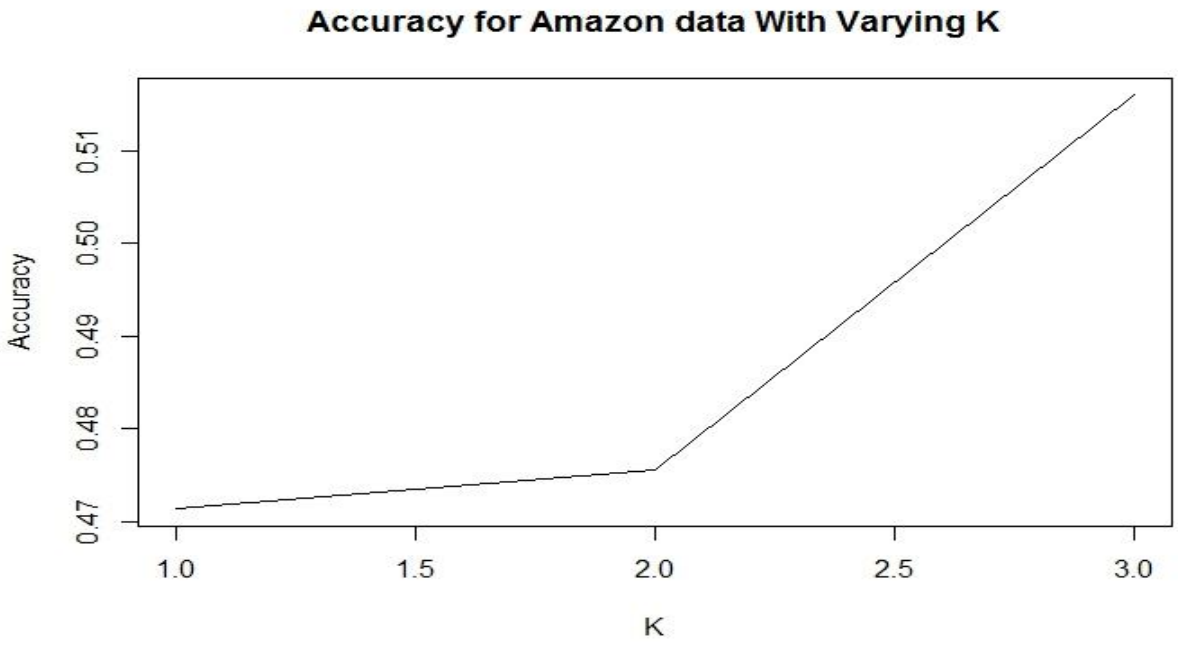
Implementation:

- For the implementation of knn: **R language is used**
- Packages used: **class.**
- Load the packages by using **library** command in R if it is already installed, otherwise install the libraries by using **install. packages** command.
- Headers of the datasets: name, review, rating.
- We have calculated the sentiment score by using stringr and plyr library in R.
- We obtained sentiment score, positive word count and negative word count for each column and stored.
- **While performing knn method on this data, we are getting error saying: "R error: Too many ties in knn**. We have tried with different values of k and it was failing with same error. For larger values of k, R studio started crashing.
- We have tried **the same code in High Performance cluster for higher values of k and it was showing segmentation fault error.**
- After careful analysis, we found that similarity measures of different reviews are having ties to different class variables and we understood that we need to improve our sentiment score calculation.
- To improve the sentiment score accuracy, we implemented sentiment calculation by using **Vader Sentiment library perform and it was giving better performance compared to RSentiment** library from R.
- The SentimentIntensityAnalyzer method takes the text review as input and output four values, namely, **"pos" (positive score of the review), "neg" (negative score of the review), "neu" (neutral score of the review), "compound" (normalized value of the sum of all sentiment scores)**
- After calculating the sentiment scores of both training data and test data, sentiment scores of both the files are stored in separated files namely sentiment_train.csv and sentiment_test.csv respectively.
- Now we are loading this files in our R script by using **read.csv** command.
- Class variables of both training data and test data is stored in the **train_target** and **test_target.**
- "knn" method from class library is used execute k-NN algorithm. **As parameters to the method, the train dataset, test dataset and the class variables for the train dataset is passed. This method is executed in a for loop with k value ranging from 1 to 3.** This k value is passed as parameter to the "knn" method at every iteration. We are checking with different values of k to find the accuracy variation with respect to k.
  predict <- knn (train = training_features, test = testing_features, cl = training_target [], k = k_value)
- We are not able to give k value greater than 3 because it is showing too many ties error
- There is **one more parameter named "l" passed** in the knn method of class library which needs a numeric value. Default value is 0. The value of this parameter defines the minimum vote for definite decision. We have tested with different values of l and found that better result is obtained when the value of l is 0.

- For every value of k, we have calculated and the graphs are plotted using plot function. Graphs are shown under the **Result & Observations heading.**

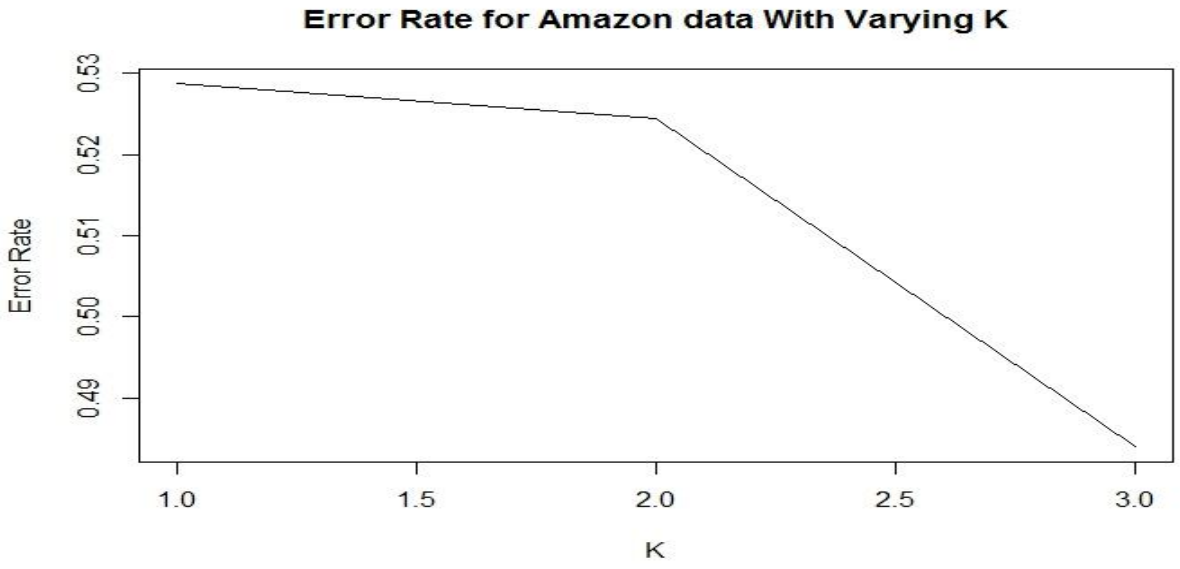
Results & Observations

Accuracy with different k values:



Accuracy is higher for k value =3 and for k value greater than 3, we are not able to calculate because of high number of ties.

Error Rate Graph for different values of k:



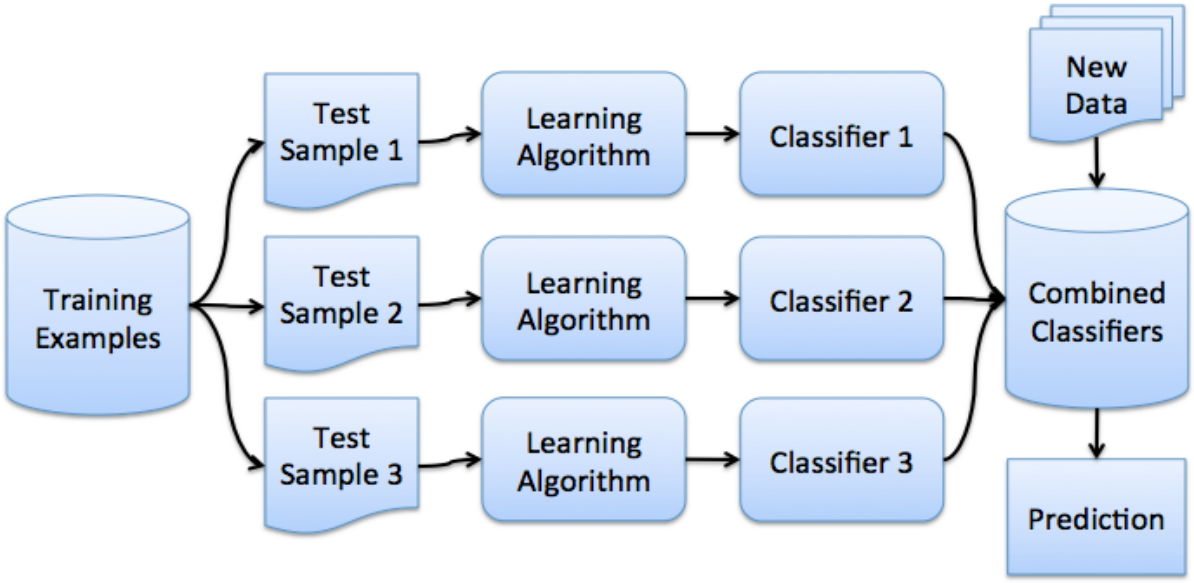
Conclusions:

- We have obtained a maximum accuracy of 51% on the test data.
- We have obtained an accuracy above 48% for all the values of k (1:3)
- Successfully implemented knn algorithm by using class library.
- Anomalies in the sentiment score calculation affecting the accuracy for the amazon data.

Boosting

Boosting is machine learning ensemble algorithm which reduces bias and variance in supervised learning. Boosting is family of machine learning algorithm which converts weak learners into strong ones. This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly or a maximum number of models are added.

Following image will give an idea about how boosting is performed.



We have used two algorithms to perform the boosting technique for our datasets.

1) **C5.0 (Decision Trees)** is an advancement of C4.5, which is basically an extension of ID3 algorithm.
2) **Gradient Boosting**, which is basically about "boosting" many weak predictive models into a strong one in the form of an ensemble of weak models.

Optical Recognition of Handwritten Digits Dataset

Implementation:

- For the implementation of knn: **R language is used**
- Packages used: **class, caret, caretEnsemble, C50.**
- Load the packages by using **library** command in R if it is already installed, otherwise install the libraries by using **install. packages** command.
- After that load the training data by using **read.csv** command.
- The dataset for train and test data does not have a header. So, the heaters for the feature variables are names V1 to V64. The header for class variable is named V65.
- Training data is stored in the **train_data** variable.
- Test data is stored in the **test_data** variable.
- Now we are training the data by using train method.
- Parameters used for this method are:
    1) Formula: V65~
    2) Training data: train_data
    3) Training method: Either we are choosing C5.0 or gradient boosting (gbm).
    4) Metric: Default it is given as Accuracy. It can also have value “Kappa” for classification. For regression, RMSE” and “Rsquared is used.
    5) Trcontrol: A trainControl object is passed. This object is created with three parameters, method (“repeatedcv” for repeated training and test splits), number (5, specifies the number of folds/number of resampling iterations), repeats (3, for repeated k-fold cross validation only: number of complete sets of folds to compute).

control <- trainControl (method="repeatedcv", number=5, repeats=3)

model_c50 <- train (V65~., data=train_data, method="C5.0", metric="Accuracy", trControl=control)

model_gbm <- train (V65~., data=train_data, method="gbm", metric="Accuracy", trControl=control, verbose=FALSE)

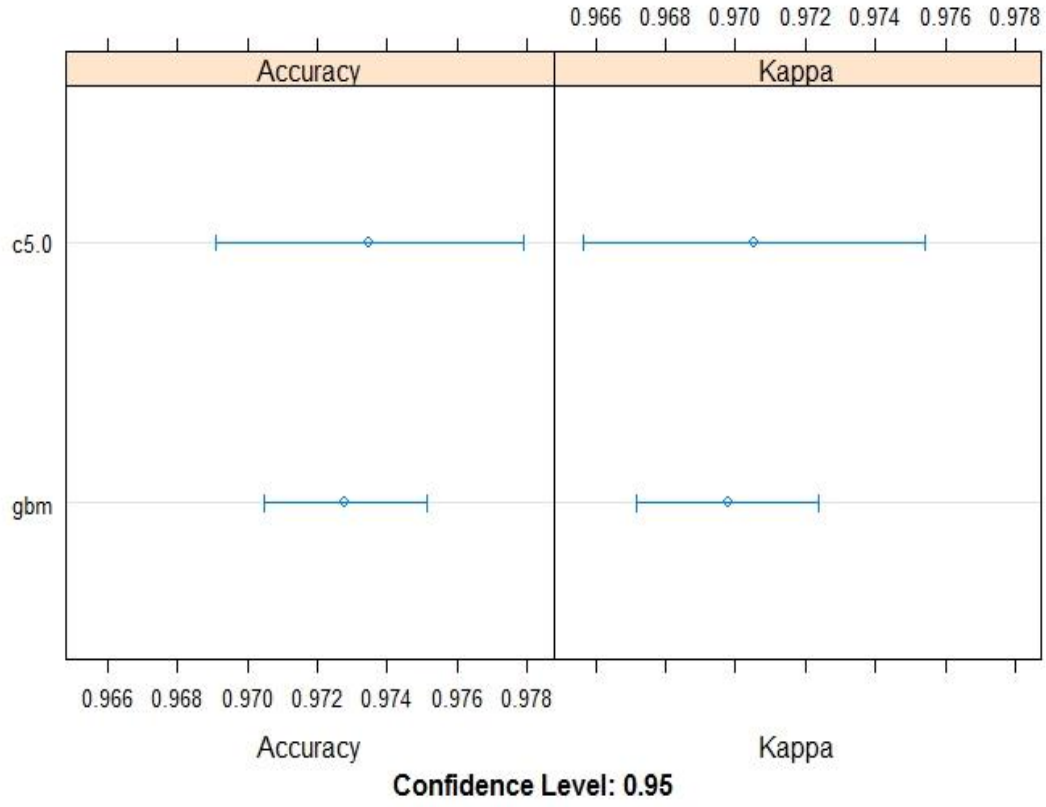
- We are creating two models by using C5.0 and gradient boosting algorithm.
- Now we are using resamples method, we are comparing the models.
  boosting_results <- resamples (list (c5.0=model_c50, gbm=model_gbm))
- Now we are using the above models to predict the result on test data.
  summary(boosting_results)
  dotplot(boosting_results)
  predict.c50 <- predict(model_c50, test_data)
  mean(test_data_target == predict.c50)
  predict.gbm <- predict(model_gbm, test_data)
  mean(test_data_target == predict.gbm)
  splom(boosting_results)

Results & Observations

- The model generated using gradient boosting algorithm is slightly better with approximately >97% accuracy. There is not much difference between accuracies of the two algorithms.

```
summary(boosting_results)

Call:
summary.resamples(object = boosting_results)

Models: c5.0, gbm
Number of resamples: 15

Accuracy
       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
c5.0 0.9607  0.9667 0.9725 0.9735  0.9817 0.9844    0
gbm  0.9660  0.9705 0.9739 0.9728  0.9758 0.9791    0

Kappa
       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
c5.0 0.9564  0.9630 0.9695 0.9705  0.9797 0.9826    0
gbm  0.9622  0.9673 0.9710 0.9698  0.9731 0.9767    0
```

- Accuracy of boosting model is shown in the following figure:



Conclusions:

- Successfully implemented boosting algorithm by using C5.0 and gradient boosting algorithms
- Gradient boosting gives a better accuracy than C5.0 model
- We can clearly see that boosting has increased the accuracy of the test data. We got only 77% accuracy with decision tree algorithm and we are getting an accuracy of 96.4% by using boosting with C5.0 model.

Amazon Reviews Sentiment Analysis Dataset

Implementation:

- For the implementation of knn: **R language is used**
- Packages used: **class, caret, caretEnsemble, C50.**
- Load the packages by using **library** command in R if it is already installed, otherwise install the libraries by using **install. packages** command.
- After that load the training data by using **read.csv** command.
- Headers of the datasets: name, review, rating.
- We have calculated the sentiment score by using stringr and plyr library in R.
- We obtained sentiment score, positive word count and negative word count for each column and stored.
- **While performing knn method on this data, we are getting error saying: "R error: Too many ties in knn**. We have tried with different values of k and it was failing with same error. For larger values of k, R studio started crashing.
- We have tried **the same code in High Performance cluster for higher values of k and it was showing segmentation fault error.**
- After careful analysis, we found that similarity measures of different reviews are having ties to different class variables and we understood that we need to improve our sentiment score calculation.
- To improve the sentiment score accuracy, we implemented sentiment calculation by using **Vader Sentiment library perform and it was giving better performance compared to RSentiment** library from R.
- The SentimentIntensityAnalyzer method takes the text review as input and output four values, namely, **"pos" (positive score of the review), "neg" (negative score of the review), "neu" (neutral score of the review), "compound" (normalized value of the sum of all sentiment scores)**
- After calculating the sentiment scores of both training data and test data, sentiment scores of both the files are stored in separated files namely sentiment_train.csv and sentiment_test.csv respectively.
- Now we are training the data by using train method.
- Parameters used for this method are:
    1) Formula: rating ~ negative_score + positive_score + neutral_score + compound_value
    2) Training data: Amazon_train_data
    3) Training method: Either we are choosing C5.0 or gradient boosting (gbm).

4) Metric: Default it is given as Accuracy. It can also have value “Kappa” for classification. For regression, RMSE” and “Rsquared is used.
5) Trcontrol: A trainControl object is passed. This object is created with three parameters, method (“repeatedcv” for repeated training and test splits), number (5, specifies the number of folds/number of resampling iterations), repeats (3, for repeated k-fold cross validation only: number of complete sets of folds to compute).

control <- trainControl(method="repeatedcv", number=5, repeats=3)

model_c50 <- train(rating ~ negative_score + positive_score + neutral_score + compound_value, data = Amazon_train_data, method = "C5.0", metric = "Accuracy", trControl = control)

model_gbm <- train(rating ~ negative_score + positive_score + neutral_score + compound_value, data = Amazon_train_data, method = "gbm", metric = "Accuracy", trControl = control, verbose = FALSE)

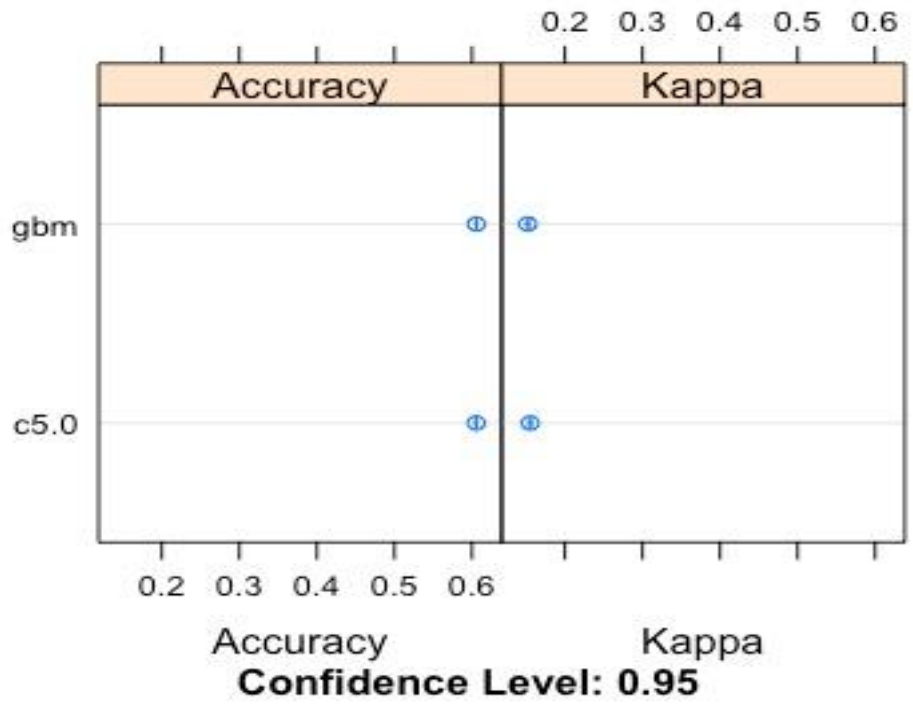
- We are creating two models by using C5.0 and gradient boosting algorithm.
- Now we are using resamples method, we are comparing the models.
  boosting_results <- resamples (list (c5.0=model_c50, gbm=model_gbm))
- Now we are using the above models to predict the result on test data.
  summary(boosting_results)
  dotplot(boosting_results)
  pr.c50 <- predict(model_c50, Amazon_test_data)
  mean(Amazon_test_data$rating == pr.c50)
  pr.gbm <- predict(model_gbm, Amazon_test_data)
  mean(Amazon_test_data$rating == pr.gbm)

Results & Observations

We have obtained an accuracy of 61% for both the models

```
> summary(boosting_results)

Call:
summary.resamples(object = boosting_results)

Models: c5.0, gbm
Number of resamples: 15

Accuracy
       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
c5.0 0.6028  0.6038 0.6057 0.6053  0.6065 0.6073    0
gbm  0.6042  0.6051 0.6055 0.6058  0.6066 0.6080    0

Kappa
       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
c5.0 0.1483  0.1544 0.1549 0.1552  0.1569 0.1599    0
gbm  0.1452  0.1508 0.1524 0.1520  0.1527 0.1582    0
```

Accuracy for the boosting model is represented below:



Conclusions:

- Successfully implemented boosting algorithms on amazon dataset
- We got better accuracy by using boosting in amazon data set
- Still the sentiment score analysis is not improving as expected even by using python.

Collaborated with: Ashwin Venkatesh Prabhu

References:

https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/

http://machinelearningmastery.com/machine-learning-ensembles-with-r/