Implementing KNN, CART, C5.0, and Random Forest models to predict and evaluate the popularity of Online News

Celestino B. Edaňo Jr.

March 19, 2017

Technological Institute of the Philippines, Quezon City

**Introduction**

In this information era, reading and sharing news have become the center of people’s entertainment lives. With the expansion of the Internet, more and more people enjoys reading and sharing online news articles. The number of shares under a news article indicates how popular the news is. Therefore, it would be greatly helpful if we could accurately predict the popularity of news prior to its publication, for social media workers like authors, advertisers, etc.

According to He Ren and Quan-Yang in their paper entitled “Predicting and Evaluating the Popularity of Online News”, Random Forest model turns out to be the best model for prediction because it achieved an accuracy of 70% with optimal parameters. In their work, they implemented 10 different learning algorithms on the dataset, ranging from various regressions to SVM and Random Forest. Their performances are recorded and compared. They also included feature selection methods to improve the performance and reduce the features of the dataset.

In this project proposal, with the use of different Machine Learning models, I intend to find the best model and set of feature to predict the popularity of online news. My data comes from Mashable, a well-known online news website. I am going to implement 4 different Machine Learning models on my dataset which are KNN, CART, C5.0, and Random Forest models. Each algorithm performances will be recorded and compared to come up which is the best model to use in predicting the popularity of online news. My work can help online news companies to predict news popularity before publication.

**Dataset**

My dataset is provided by UCI Machine Learning Repository, which is a real data originally acquired and preprocessed by K. Fernandes et al. This dataset summarizes a heterogeneous set of features about articles published by Mashable in a period of two years. It extracts 61 attributes as numerical values, describing different aspects of each article, from a total of 39797 articles. The goal is to predict the number of shares in social networks (popularity). There are no missing values, but some articles or topics were unclassified. The dimensionality of my dataset is 60d (58 predictive attributes, 2 non-predictive, and 1 goal field) and it is also represented in the form of a data frame, which is a table, array-like structure, in which each column contains measurements on one variable, and each row contains one case.

**Methods**

*Software*

The following are the software that I will be going to use:

* **R-Studio** – this software is needed to test different learning models such as CART, KNN, etc., and it also acts as a user interface for R Programming Language.
* **Python IDLE** – same like R-Studio, just an alternative software in case of R-Studio compatibility issues. Like R, I used it to test different learning models.
* **WEKA** – collection of machine learning algorithms for data mining tasks. Used for preliminary analysis and experiment.

*Programming Language*

The following are the programming languages that I will go to use:

* **R Language** – use to predict the popularity of the news article using the Online New Popularity dataset.
* **Python** – like R, I used it to predict the popularity of the news article.

*Machine Learning models*

The following are the Machine Learning models that I will go to use in my project:

* **K – Nearest Neighbors (KNN) model** – it is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition as a non-parametric technique.
* **Classification and regression tress (CART) model** – an algorithm that employs decision trees and can be used for a variety of business and scientific applications.
* **C5.0 model** – it is widely used as a decision tree method in machine learning. This type of decision tree model is based on entropy and information gain.
* **Random Forest model** - one of the best among classification algorithms - able to classify large amounts of data with accuracy.

**Performance Measures**

*Objectives*

The main goal of this paper is to make use of a largely and recently collected dataset with over 39,000 articles from Mashable website, to first select informative features and then analyze and compare the performance of several machine learning algorithms. Specifically, it aims to:

* Use different Machine Learning models such as KNN, CART, C5.0, and Random Forest models to predict and evaluate the popularity of online news.
* Select informative features and then analyze and compare the performance of each models.
* Individually make a sensitivity and specificity graph for each model.
* Make a true positive rate and false positive rate graph for all the machine learning models that I am going to use.

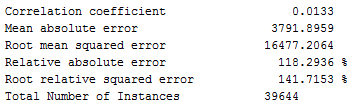
*Methods and Tools*

**WEKA**

I used WEKA just for my preliminary analysis.

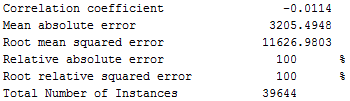
*IBk*

I run IBk which is the counterpart of KNN in R and used cross-validation with 10 folds and here are the results I got:



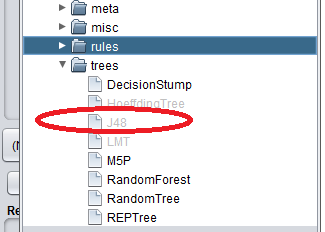
*REPTree (CART)*

Since there is no full implementation of CART in WEKA I used REPTree. This class implements cost complexity pruning for classification trees. It doesn't handle numeric classes, so it’s not the 'R' part of CART. REPTree does handle regression problems, but doesn't do cost complexity pruning. I also used cross-validation with 10 folds and here are the results I got:



*J48 (C4.5)*

Since WEKA does not support C5.0 classifier, I used C4.5 which is the older version of C5.0. J48 is an open source Java implementation of the C4.5 algorithm in the Weka data mining tool. But unfortunately, J48 classifier is not available.



*Random Forest*

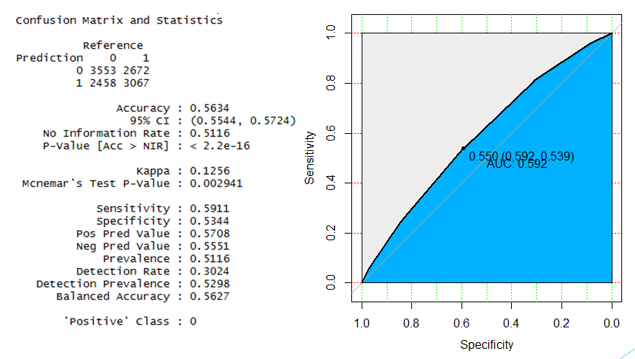
I run Random Forest but unfortunately there’s an pop up error saying that my machine has not enough memory.

**R and R Studio**

I used this tool to find out which model has the highest accuracy.

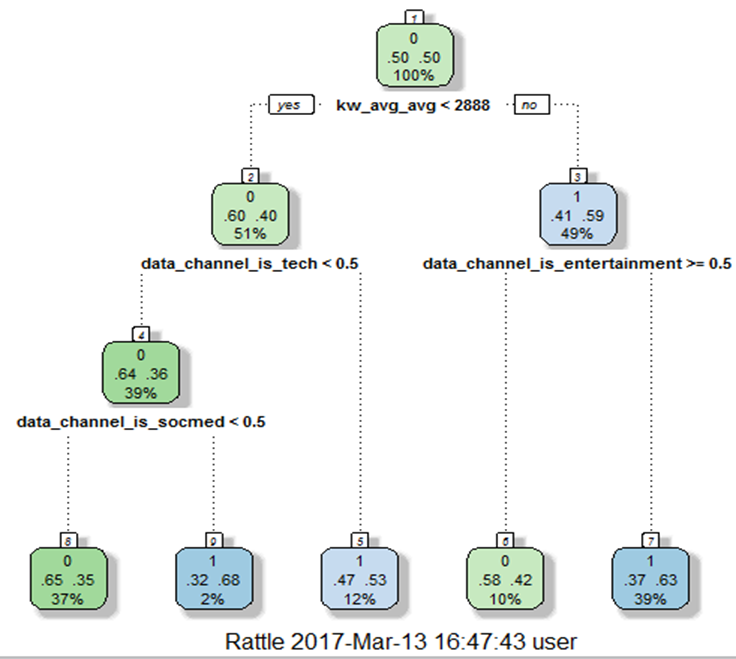
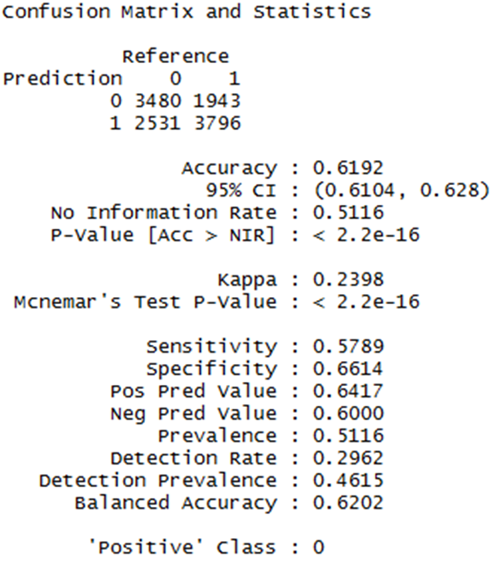
*KNN (k-nearest neighbors)*

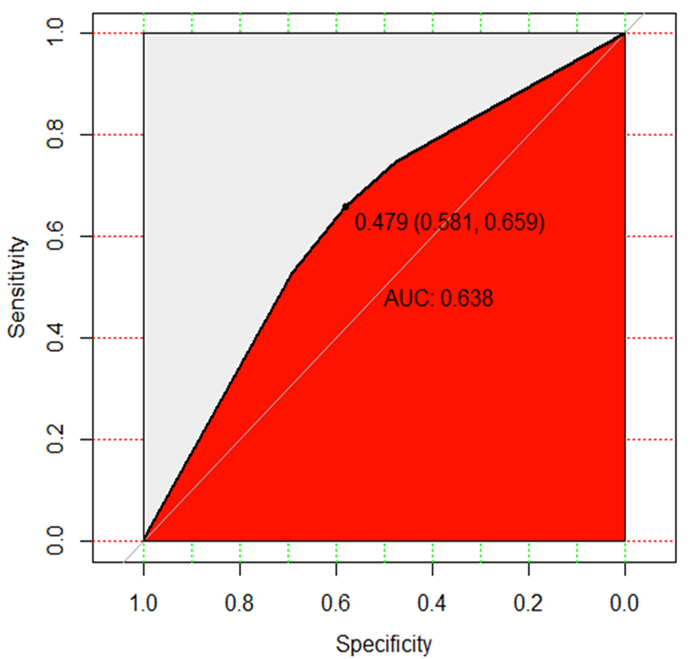
A non-parametric (make no assumptions about the probability distributions of the variables) model used for classification and regression. My results show the confusion matrix and ROC curve:



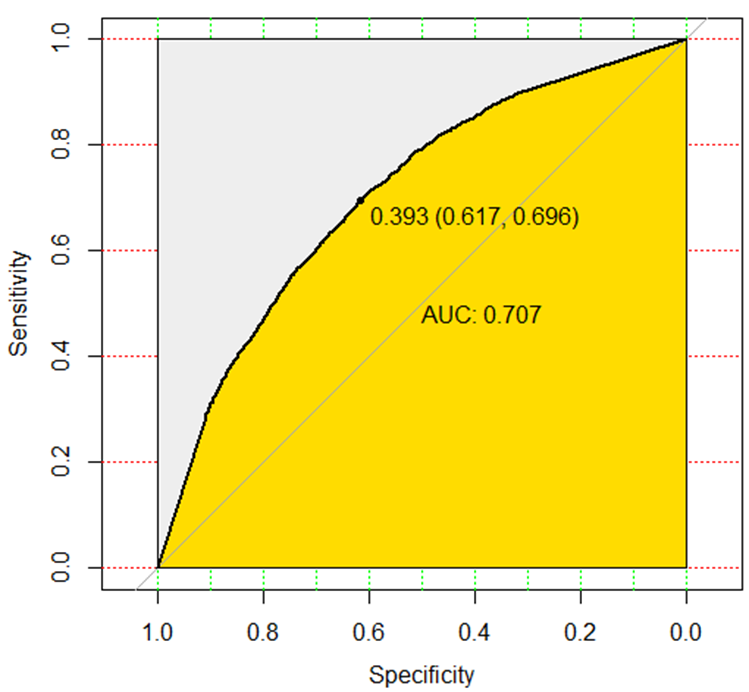
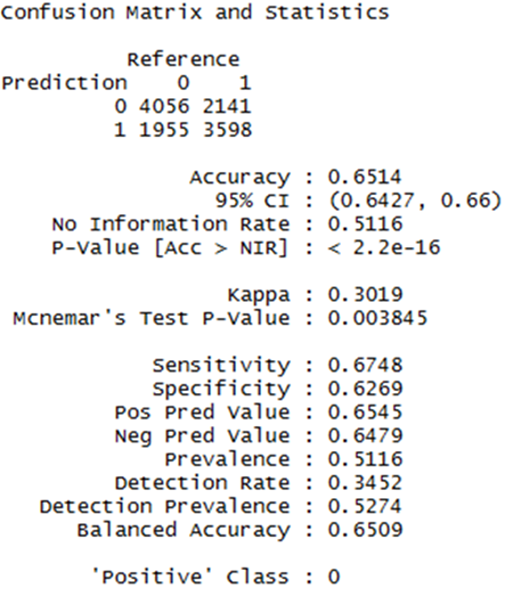
*CART (Classification and Regression Trees)*

An algorithm that employs decision trees and can be used for a variety of business and scientific applications. My results show the confusion matrix, nodes of the tree and ROC curve:



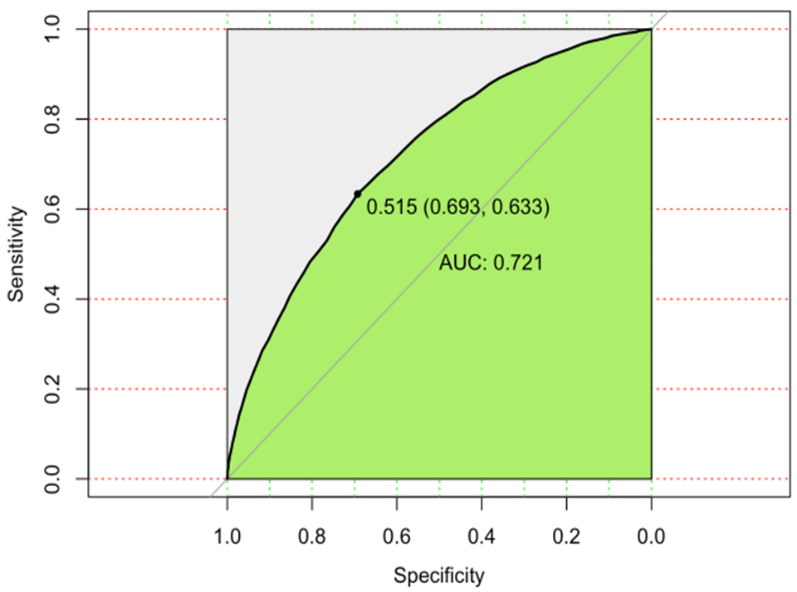
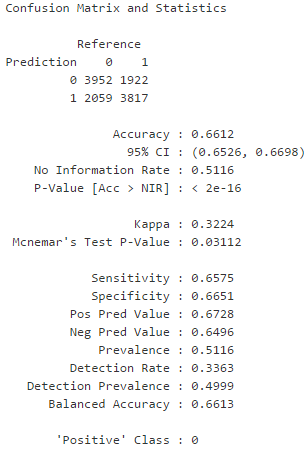


*C5.0*

A model which is widely used as a decision tree method in machine learning. This type of decision tree model is based on entropy (measure of the randomness in the information being processed) and information gain. My results show the confusion matrix and ROC curve: 

*Random Forest*

One of the best among classification algorithms - able to classify large amounts of data with high accuracy. My results show the confusion matrix and ROC curve:



Random Forest

In bagging (Bootstrap Aggregation), numerous replicates of the original dataset are created to reduce the variance in prediction. Random Forest use multiple decision trees which are built on separate sets of examples drawn from the dataset. In each tree, I can use a subset of all the features I have. By using more decision trees and averaging the result, the variance of the model can be greatly lowered. Given a training set x (1), x (2), · · ·, x(n) with responses y (1), y (2), · · ·, y (n), bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these examples:

* For b = 1, · · ·, B:

1. Sample, with replacement, n training examples called Xb, Yb

2. Train a decision or regression tree fb on Xb, Yb

* After training, predictions for unseen examples x1 can be made by averaging the predictions from all the individual regression trees on x1:



or by taking the majority vote in the case of decision trees.

For Random Forest, there are two main parameters to be considered: number of trees and number of features they select at each decision point. Theoretically, accuracy will increase with more trees making decision. I use cross validation to see how the performance changes with these parameters. I ensured that every value within a certain range that my computer can support is tested, and the result is plotted. In this case, I was able to see exactly the relationship between performance and parameters.

**Python**

We have identified Random Forest as being the best performing among several classifiers. With an accuracy of 67%, recall of 71%, and AUC of 73% for Random Forest. This time I used Python and replace CART with Naïve Bayes, and C5.0 with simple Decision Tree.

*Classifiers*

* Decision Tree – decision tree is commonly used in classification in machine learning, it is a treelike graph or model to make decisions and get consequent results. Decision tree implicitly perform variable screening or feature selection and the best feature of using it is easy to interpret and explain.
* KNN – a nonparametric method used for classification and regression. The input normally consists of k closest training instances in the space. An object is classified to a class because it is most common among its k nearest neighbors. I choose this classifier because KNN don’t need make any assumption on the underlying data distribution, it is also a lazy algorithm and don’t use training data to do any generalization.
* Random Forest – in the classification, random forest constructs multitude of decision trees at training time and then output the class that is the mode of classes or mean prediction of individual trees. It can overcome the shortage of single decision tree’s overfitting to training set.
* Naïve Bayes – a probabilistic classifier based on Bayes’ theorem. It depends on the assumption that some features are conditionally independent. It is super simple and can converge quicker than some other models, so it has a chance to achieve fast and easy process.

*External Python libraries for data analysis*

* Pandas – Pandas is an open source, BSD licensed library. It provides high performance and simple tools for data analysis. It is mainly used to load data from the raw Excel .csv file.
* Scikitlearn – is an open source, BSD licensed, machine learning python library. It built on NumPy, SciPy and matplotlib. It also provides simple and efficient tools for data mining and data analysis. For this project, I used it for many purposes: split dataset into training and testing sets; construct 4 different estimators(model) and then fit the model; get the score(accuracy) with input of testing data set; or do cross validation directly and get the score (which is considered an improvement process in this project).
* XlsxWriter – is a Python module that can be used to write text, numbers, formulas and hyperlinks to multiple worksheets in an Excel 2007+ XLSX file. I used it to write results (accuracy and time per train/test set increase, based on 4 models) into an excel output file(.xlsx). Then in the excel file, I use internal Excel chart graphing functionality to plot comparison charts.

*Process*

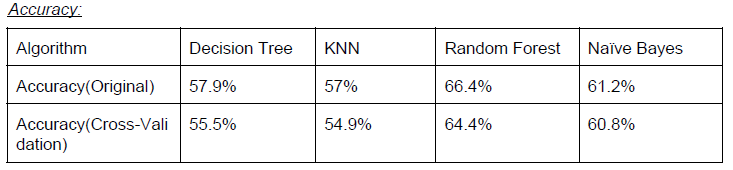
1. Use Pandas, to read data from Excel .csv file and then process the dataset: Replace the goal attribute values with binary value according to the following decision threshold:



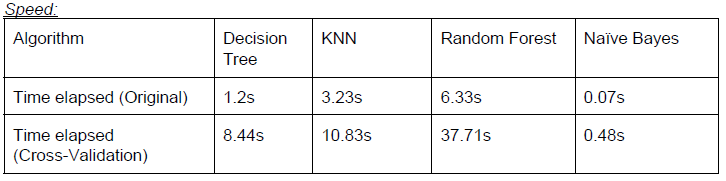
1. Use method from Scikitlearn, cross\_validation.train\_test\_split to split the dataset to two parts: training set(70%) and testing set(30%)
2. Using the classifier in Scikitlearn, construct estimator object based on different algorithms (Decision Tree, KNN, Random Forest, Naïve Bayes):
   * dt = DecisionTreeClassifier(min\_samples\_split=20,random\_state=99)
   * knn = KNeighborsClassifier()
   * rf = RandomForestClassifier(n\_estimators=100,n\_jobs=1)
   * nb = BernoulliNB()
3. Use fit method to fit the mode, with training data set as the input, and then use score method to get the score(accuracy) with testing data set as the input.
4. Use timestamp t=time() to record every model’s construction start and end time, then end\_time minus start\_time to get every model’s time elapsed(construction and use model). Collect every model’s result (time elapsed and accuracy) and plot it on tables below.
5. Keep training/testing set unchanged and increases another set(testing/training) 5% of original size every time, and then get each result (accuracy and time elapsed) for every model, record it and use XlsxWriter to write it to excel file(.xlsx). In the excel file, use internal chart graphing functionality to plot charts for comparison and analysis.

*Performance analysis and results*

Accuracy



Speed

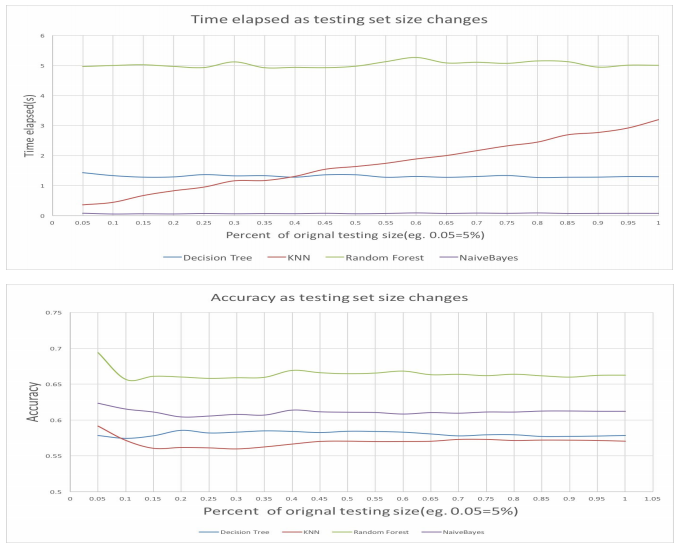


From two tables above, we can see Random Forest model achieves the highest accuracy (66.4%), but it consuming the most time (6.63s) among 4 models. Naive Bayes model achieves the least time elapsed(0.07s) and a properly medium accuracy. The other two models have bad accuracy. The k-fold cross-validation has proved that the accuracy should be less than that gained from the original method. But considering its big time consuming, the difference can be ignored relatively under the purpose of comparing these 4 models.

Scalability

* Performance by changing testing set.

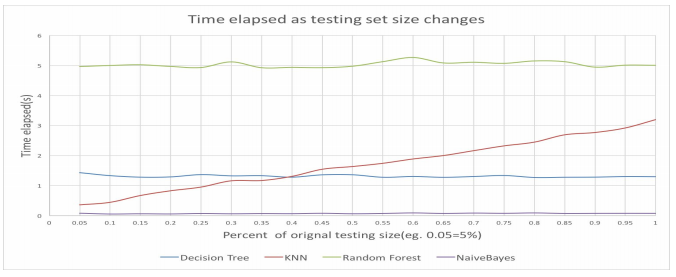
For every model, training set keeps unchanged (60% of total dataset), testing set increases 5% of original testing set (40% of total dataset) each time. So, testing set increases 2%(40 x 5%) of total size each time. Here is how result (Time elapsed/Accuracy) looks:

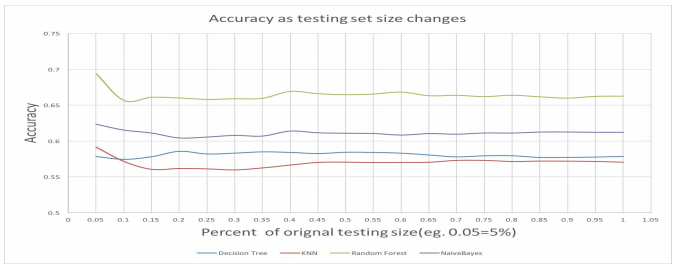


From the two charts above, as size of testing set increases, every model’s accuracy keeps stable. For time, elapsed KNN increases linearly, and other three models keeps stable.

* Performance by changing training set:

For every model, testing set keeps unchanged (40% of total dataset), training set increases 5% of original testing set (60% of total dataset) each time. So, testing set increases 3%(60% x 5%) of total size each time. Here is how result (Time elapsed/Accuracy) looks:





From the two charts above, as size of training set increases, every model’s accuracy increases linearly at the beginning and then slightly increases and tend to keep stable, and KNN’s change rate is higher than any other three models. For the time, elapsed Naïve Bayes model still keeps stable and other three models change linearly: Random Forest changed faster than other two models.

*Observation and Conclusion*

Decision Tree has a bad performance (55.5% accuracy based on cross validation), it is because the dataset’s volume is not enough to take so many attributes (58 predictive attributes in total), and then it occurs overfitting due to relatively lack of samples. KNN also has a bad performance (54.9% accuracy based on cross validation), it is because KNN is very sensitive to local structure of the data, every node’s value is mostly determined by the contribution of near neighborhoods than further nodes. Random Forest has the best accuracy, which is 64.4% based on cross validation and 66.4% based on original dataset splitting (60% training and 40% testing). It builds a forest of decision trees based on differing attributes in the nodes. Different trees have access to a different random sub collection of the dataset, and it constructs a collection of decision trees with controlled variance. So, it overcome the shortage of decision tree’s easy overfitting and can have a better performance. Naive Bayes model has a medium accuracy, which is 61.2%. But it has a faster (always less than 1s) constructing and model testing time than any other models. And as the training/testing sets increases, it always keeps stable. It is basically because it depends on the assumption that some features are conditionally independent. It is super simple and can converge quickly.

Overall, if anyone pursues a faster process, Naïve Bayes is the best choice for this problem without sacrificing too much accuracy. Otherwise, if accuracy is considered first, obviously Random Forest is a better one.

**Relation to Artificial Intelligence and Machine Learning**

My project is related to both AI and Machine Learning because I’m using Machine Learning tools to predict the popularity of online news. I also used AI approach to test whether the model or classifier has a good performance or not.

**Limitations**

* Limited memory - my laptop’s RAM is not enough to handle or process big data set. In that case, I can’t do bagging, boosting, and stacking since it requires a lot of memory.
* Too long to process – random forest takes hours to return result since my data set is large.
* I can’t use GLM, svmRadial models even though my dataset can do both classification and regression tasks.

**Future Work**

As is seen from the result, no algorithm can reach 70% accuracy given the data set I have, even though they are state-of-the-art. To improve accuracy, there is little room in model selection but much room in feature selection. In the preprocessing round, 59 features were extracted from news articles, and my later work is based on these features. However, the content of news articles hasn’t been fully explored. Some features are related to the content, such as LDA topics (feature #39 - #43), which are convenient to use for learning, but reflect only a small portion of information about the content. Random Forest classifier can be considered as a generalizable model across different variations of the data set when classes are labeled at the median. This is because even though sensitivity is not very high, the difference between the sensitivity in training and testing set is very low, indicating this model can be used to predict future unknown cases. When the classifier is trained on datasets with labels set at the third quartile resulting in unbalanced distribution, the classifier has very poor sensitivity – that is, it cannot recognize the minority class, the popular articles. However, when the same dataset is balanced, it performs very well in training and unbalanced testing set. In real life, extremely popular instances of viral news articles are very rare (Lee, Hi, Fang 2016). As a result, our dataset does not contain enough cases of popular articles. In fact, it has no online news articles with more than one million shares. Hence, the classifier may not perform as effectively on other datasets.

In the future, I could directly treat all the words in an article as additional features, and then apply machine learning algorithms like Naive Bayes and SVM. In this way, what the article really talks about is taken into account, and this approach should improve the accuracy of prediction if combined with my current work.

**Summary**

Random Forest has the best result for this classification problem. It can have different number of decision trees and different number of features used for each decision point. The number of training examples can also change. Therefore, implementation should be done in a systematic way. I change only one variable at a time. I first use a default setting for Random Forest and increase the number of training examples. The error decreases to a certain level. Then I set the number of trees to be constant, and change the number of features used for decision. It turns out that log(Nmax) is the best value. Finally, I change the number of trees continuously from 5 to 500. The accuracy reaches a limit of 67%, which is the best among all algorithms.

# **APPENDICES**

*R Codes*

library(randomForest) #Classification and regression model

library(miscTools) #Extracting the standard errors, obtaining the number of (estimated) parameters.

library(caret) #Training and plotting classification and regression models

library(ROCR) #For graphs, sensitivity/specificity curves, lift charts, and precision/recall plots

library(pROC) #visualizing, smoothing and comparing receiver operating characteristic (ROC curves)

library(e1071) #For latent class analysis

library(C50) #Decision trees and rule-based models for pattern recognition

library(rpart) #Recursive partitioning for classification, regression and survival trees

library(rpart.plot) #Plot rpart models

library(rattle) #Load data from a CSV file

library(RColorBrewer) #Provides color schemes for maps

library(ggplot2) #Creating graphics

#Loading the dataset.

news <- read.csv(file="C:/Users/user/Desktop/MLFinalProject/OnlineNewsPopularity.csv", head=TRUE, sep= ",")

dataset <- news

summary(news)

#Deleting URL and timedelta columns

newsreg <- subset( news, select = -c(url, timedelta ) )

#Standardize the data

#Generate z-scores using the scale() function

for(i in ncol(newsreg)-1){

newsreg[,i]<-scale(newsreg[,i], center = TRUE, scale = TRUE)

}

#Define articles with shares larger than 1400 (median) as popular article

# Dataset for classification

newscla <-newsreg

newscla$shares <- as.factor(ifelse(newscla$shares > 1400,1,0))

#Split the data train 70% test 30%

#set random situation

set.seed(100)

#Select traning data and prediction data

ind<-sample(2,nrow(newscla),replace=TRUE,prob=c(0.7,0.3))

#Color palatte

color.knn<-'#00B2FF'#blue for KNN

color.cart<-'#FF1300' #red for cart

color.c50<-'#FFDC00' #yellow for c50

color.rf<-'#00FF49' #green for random forest

####KNN####

newscla.knn <- knn3(shares ~.,newscla[ind==1,])

newscla.knn.pred <- predict( newscla.knn,newscla[ind==2,],type="class")

newscla.knn.prob <- predict( newscla.knn,newscla[ind==2,],type="prob")

# Confusion matrix

confusionMatrix(newscla.knn.pred, newscla[ind==2,]$shares)

# ROC Curve

newscla.knn.roc <- roc(newscla[ind==2,]$shares,newscla.knn.prob[,2])

plot(newscla.knn.roc, print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),

grid.col=c("green", "red"), max.auc.polygon=TRUE,

auc.polygon.col=color.knn, print.thres=TRUE)

####CART###

newscla.cart<-rpart(shares ~.,newscla[ind==1,],method='class')

fancyRpartPlot(newscla.cart) #Generating nodes

summary(newscla.cart)

# ROC Curve

newscla.cart.roc <- roc(newscla[ind==2,]$shares,newscla.cart.prob[,2])

plot(newscla.cart.roc, print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),

grid.col=c("green", "red"), max.auc.polygon=TRUE,

auc.polygon.col=color.cart, print.thres=TRUE)

####C5.0####

newscla.c50<-C5.0(shares ~.,newscla[ind==1,],trials=5)

summary(newscla.c50)

#predict

newscla.c50.pred<-predict( newscla.c50,newscla[ind==2,],type="class" )

newscla.c50.prob<-predict( newscla.c50,newscla[ind==2,],type="prob" )

# Confusion matrix

confusionMatrix(newscla.c50.pred, newscla[ind==2,]$shares)

# ROC Curve

newscla.c50.roc <- roc(newscla[ind==2,]$shares,newscla.c50.prob[,2])

plot(newscla.c50.roc, print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),

grid.col=c("green", "red"), max.auc.polygon=TRUE,

auc.polygon.col=color.c50, print.thres=TRUE)

# Precision/Recall graph

newscla.c50.pred.pred <- prediction(newscla.c50.prob[,2],newscla[ind==2,]$shares)

newscla.c50.pred.perf <- performance(newscla.c50.pred.pred,"prec","rec")

plot(newscla.c50.pred.perf, avg= "threshold", colorize=T, lwd= 3,

main= "... Precision/Recall graphs ...")

plot(newscla.c50.pred.perf, lty=3, col="grey78", add=T)

####Random Forest####

newscla.rf<-randomForest(shares ~.,newscla[ind==1,],ntree=100,nPerm=10,mtry=3,proximity=TRUE,importance=TRUE)

summary(newscla.rf)

# Plotting the number of trees vs error

plot(newscla.rf)

# Feature importance

newscla.rf.imp <- importance(newscla.rf)

newscla.rf.impvar <- newscla.rf.imp[order(newscla.rf.imp[, 3], decreasing=TRUE),]

newscla.rf.impvar

# Plot feature importance

varImpPlot(newscla.rf)

# Partial dependence

partialPlot(newscla.rf, newscla[ind==1,], kw\_avg\_avg, "0", main='' , xlab='kw\_avg\_avg', ylab="Variable effect")

#predict

newscla.rf.pred<-predict( newscla.rf,newscla[ind==2,], type="class")

newscla.rf.prob<-predict( newscla.rf,newscla[ind==2,], type="prob")

# Confusion matrix

confusionMatrix(newscla.rf.pred, newscla[ind==2,]$shares)

# ROC Curve

newscla.rf.roc <- roc(newscla[ind==2,]$shares,newscla.rf.prob[,2])

plot(newscla.rf.roc, print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),

grid.col=c("green", "red"), max.auc.polygon=TRUE,

auc.polygon.col=color.rf, print.thres=TRUE)

#Comparing all the models#

ROCCurve<-par(pty = "s")

plot(performance(prediction(newscla.knn.prob[,2],newscla[ind==2,]$shares),'tpr','fpr'),

col=color.knn, lwd=3

)

text(0.55,0.6,"KNN",col=color.knn)

plot(performance(prediction(newscla.cart.prob[,2],newscla[ind==2,]$shares),'tpr','fpr'),

col=color.cart, lwd=3, add=TRUE

)

text(0.3,0.4,"CART",col=color.cart)

plot(performance(prediction(newscla.c50.prob[,2],newscla[ind==2,]$shares),'tpr','fpr'),

col=color.c50, lwd=3, add=TRUE

)

text(0.15,0.5,"C5.0",col=color.c50)

plot(performance(prediction(newscla.rf.prob[,2],newscla[ind==2,]$shares),'tpr','fpr'),

col=color.rf, lwd=3, add=TRUE

)

text(0.3,0.7,"Random Forest",col=color.rf)

*Python codes*

**Change Test code:**

import os

from sklearn.tree import DecisionTreeClassifier, export\_graphviz

import pandas as pd

import numpy as np

from sklearn.cross\_validation import train\_test\_split

from sklearn import cross\_validation, metrics

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import BernoulliNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from time import time

import xlsxwriter

# load dataset to padas dataframe

csv\_filename="OnlineNewsPopularity.csv"

df=pd.read\_csv(csv\_filename)

popular = df.shares >= 1400

unpopular = df.shares < 1400

df.loc[popular,'shares'] = 1

df.loc[unpopular,'shares'] = 0

# split original dataset into 60% training and 40% testing

features=list(df.columns[2:60])

X\_train, X\_test, y\_train, y\_test = cross\_validation.train\_test\_split(df[features], df['shares'], test\_size=0.4, random\_state=0)

# open one ouput excel file and two worksheets

workbook = xlsxwriter.Workbook('changeTest\_output.xlsx')

worksheet = workbook.add\_worksheet()

worksheet.write("A1","scala\_index")

worksheet.write("B1","DecisionTree")

worksheet.write("C1","KNN")

worksheet.write("D1","RandomForest")

worksheet.write("E1","NaiveBayes")

worksheet2 = workbook.add\_worksheet()

worksheet2.write("A1","scala\_index")

worksheet2.write("B1","DecisionTree")

worksheet2.write("C1","KNN")

worksheet2.write("D1","RandomForest")

worksheet2.write("E1","NaiveBayes")

# increasingly add size of testing set 5% of orginal, keep training size unchanged

for i in range(0,100,5):

X\_rest, X\_test\_part, y\_rest, y\_test\_part= cross\_validation.train\_test\_split(X\_test, y\_test, test\_size=0.049+i/100.0, random\_state=0)

print "====================== loop: ", i

t0=time()

print "DecisionTree"

dt = DecisionTreeClassifier(min\_samples\_split=20,random\_state=99)

# dt = DecisionTreeClassifier(min\_samples\_split=20,max\_depth=5,random\_state=99)

clf\_dt=dt.fit(X\_train,y\_train)

score\_dt=clf\_dt.score(X\_test\_part,y\_test\_part)

print "Acurracy: ", score\_dt

t1=time()

dur\_dt=t1-t0

print "time elapsed: ", dur\_dt

print "\n"

t6=time()

print "KNN"

# knn = KNeighborsClassifier(n\_neighbors=3)

knn = KNeighborsClassifier()

clf\_knn=knn.fit(X\_train, y\_train)

score\_knn=clf\_knn.score(X\_test\_part,y\_test\_part)

print "Acurracy: ", score\_knn

t7=time()

dur\_knn=t7-t6

print "time elapsed: ", dur\_knn

print "\n"

t2=time()

print "RandomForest"

rf = RandomForestClassifier(n\_estimators=100,n\_jobs=-1)

clf\_rf = rf.fit(X\_train,y\_train)

score\_rf=clf\_rf.score(X\_test\_part,y\_test\_part)

print "Acurracy: ", score\_rf

t3=time()

dur\_rf=t3-t2

print "time elapsed: ", dur\_rf

print "\n"

t4=time()

print "NaiveBayes"

nb = BernoulliNB()

clf\_nb=nb.fit(X\_train,y\_train)

score\_nb=clf\_nb.score(X\_test\_part,y\_test\_part)

print "Acurracy: ", score\_nb

t5=time()

dur\_nb=t5-t4

print "time elapsed: ", dur\_nb

# write result data to excel file

list1=[]

list2=[]

list1.append(i/100.0+0.05)

list1.append(score\_dt)

list1.append(score\_knn)

list1.append(score\_rf)

list1.append(score\_nb)

list2.append(i/100.0+0.05)

list2.append(dur\_dt)

list2.append(dur\_knn)

list2.append(dur\_rf)

list2.append(dur\_nb)

for col in range(len(list1)):

worksheet.write(i/5+1,col,list1[col])

worksheet2.write(i/5+1,col,list2[col])

workbook.close()

**Change train code:**

import os

from sklearn.tree import DecisionTreeClassifier, export\_graphviz

import pandas as pd

import numpy as np

from sklearn.cross\_validation import train\_test\_split

from sklearn import cross\_validation, metrics

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import BernoulliNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from time import time

import xlsxwriter

# load dataset to padas dataframe

csv\_filename="OnlineNewsPopularity.csv"

df=pd.read\_csv(csv\_filename)

popular = df.shares >= 1400

unpopular = df.shares < 1400

df.loc[popular,'shares'] = 1

df.loc[unpopular,'shares'] = 0

# split original dataset into 60% training and 40% testing

features=list(df.columns[2:60])

X\_train, X\_test, y\_train, y\_test = cross\_validation.train\_test\_split(df[features], df['shares'], test\_size=0.4, random\_state=0)

# open one ouput excel file and two worksheets

workbook = xlsxwriter.Workbook('changeTrain\_output.xlsx')

worksheet = workbook.add\_worksheet()

worksheet.write("A1","scala\_index")

worksheet.write("B1","DecisionTree")

worksheet.write("C1","KNN")

worksheet.write("D1","RandomForest")

worksheet.write("E1","NaiveBayes")

worksheet2 = workbook.add\_worksheet()

worksheet2.write("A1","scala\_index")

worksheet2.write("B1","DecisionTree")

worksheet2.write("C1","KNN")

worksheet2.write("D1","RandomForest")

worksheet2.write("E1","NaiveBayes")

# increasingly add size of training set 5% of orginal, keep testing size unchanged

for i in range(0,100,5):

X\_rest, X\_trian\_part, y\_rest, y\_train\_part= cross\_validation.train\_test\_split(X\_train, y\_train, test\_size=0.049+i/100.0, random\_state=0)

print "====================== loop: ", i

t0=time()

print "DecisionTree"

dt = DecisionTreeClassifier(min\_samples\_split=20,random\_state=99)

# dt = DecisionTreeClassifier(min\_samples\_split=20,max\_depth=5,random\_state=99)

clf\_dt=dt.fit(X\_trian\_part,y\_train\_part)

score\_dt=clf\_dt.score(X\_test,y\_test)

print "Acurracy: ", score\_dt

t1=time()

dur\_dt=t1-t0

print "time elapsed: ", dur\_dt

print "\n"

t6=time()

print "KNN"

# knn = KNeighborsClassifier(n\_neighbors=3)

knn = KNeighborsClassifier()

clf\_knn=knn.fit(X\_trian\_part, y\_train\_part)

score\_knn=clf\_knn.score(X\_test,y\_test)

print "Acurracy: ", score\_knn

t7=time()

dur\_knn=t7-t6

print "time elapsed: ", dur\_knn

print "\n"

t2=time()

print "RandomForest"

rf = RandomForestClassifier(n\_estimators=100,n\_jobs=-1)

clf\_rf = rf.fit(X\_trian\_part,y\_train\_part)

score\_rf=clf\_rf.score(X\_test,y\_test)

print "Acurracy: ", score\_rf

t3=time()

dur\_rf=t3-t2

print "time elapsed: ", dur\_rf

print "\n"

t4=time()

print "NaiveBayes"

nb = BernoulliNB()

clf\_nb=nb.fit(X\_trian\_part,y\_train\_part)

score\_nb=clf\_nb.score(X\_test,y\_test)

print "Acurracy: ", score\_nb

t5=time()

dur\_nb=t5-t4

print "time elapsed: ", dur\_nb

# write result data to excel file

list1=[]

list2=[]

list1.append(i/100.0+0.05)

list1.append(score\_dt)

list1.append(score\_knn)

list1.append(score\_rf)

list1.append(score\_nb)

list2.append(i/100.0+0.05)

list2.append(dur\_dt)

list2.append(dur\_knn)

list2.append(dur\_rf)

list2.append(dur\_nb)

for col in range(len(list1)):

worksheet.write(i/5+1,col,list1[col])

worksheet2.write(i/5+1,col,list2[col])

workbook.close()

**Performance Calculation code**

import os

from sklearn.tree import DecisionTreeClassifier, export\_graphviz

import pandas as pd

import numpy as np

from sklearn.cross\_validation import train\_test\_split

from sklearn import cross\_validation, metrics

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import BernoulliNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from time import time

from sklearn.metrics import roc\_auc\_score

# read .csv from provided dataset

csv\_filename="OnlineNewsPopularity.csv"

# df=pd.read\_csv(csv\_filename,index\_col=0)

df=pd.read\_csv(csv\_filename)

# handle goal attrubte to binary

popular = df.shares >= 1400

unpopular = df.shares < 1400

df.loc[popular,'shares'] = 1

df.loc[unpopular,'shares'] = 0

features=list(df.columns[2:60])

# split dataset to 60% training and 40% testing

X\_train, X\_test, y\_train, y\_test = cross\_validation.train\_test\_split(df[features], df['shares'], test\_size=0.4, random\_state=0)

# print X\_train.shape, y\_train.shape

# Decision Tree accuracy and time elapsed caculation

t0=time()

print "DecisionTree"

dt = DecisionTreeClassifier(min\_samples\_split=20,random\_state=99)

# dt = DecisionTreeClassifier(min\_samples\_split=20,max\_depth=5,random\_state=99)

clf\_dt=dt.fit(X\_train,y\_train)

print "Acurracy: ", clf\_dt.score(X\_test,y\_test)

t1=time()

print "time elapsed: ", t1-t0

# cross validation for DT

tt0=time()

print "cross result========"

scores = cross\_validation.cross\_val\_score(dt, df[features], df['shares'], cv=5)

print scores

print scores.mean()

tt1=time()

print "time elapsed: ", tt1-tt0

print "\n"

# Random Forest accuracy and time elapsed caculation

t2=time()

print "RandomForest"

rf = RandomForestClassifier(n\_estimators=100,n\_jobs=-1)

clf\_rf = rf.fit(X\_train,y\_train)

print "Acurracy: ", clf\_rf.score(X\_test,y\_test)

t3=time()

print "time elapsed: ", t3-t2

#cross validation for RF

tt2=time()

print "cross result========"

scores = cross\_validation.cross\_val\_score(rf, df[features], df['shares'], cv=5)

print scores

print scores.mean()

tt3=time()

print "time elapsed: ", tt3-tt2

print "\n"

# Naive Bayes accuracy and time elapsed caculation

t4=time()

print "NaiveBayes"

nb = BernoulliNB()

clf\_nb=nb.fit(X\_train,y\_train)

print "Acurracy: ", clf\_nb.score(X\_test,y\_test)

t5=time()

print "time elapsed: ", t5-t4

# cross-validation for NB

tt4=time()

print "cross result========"

scores = cross\_validation.cross\_val\_score(nb, df[features], df['shares'], cv=5)

print scores

print scores.mean()

tt5=time()

print "time elapsed: ", tt5-tt4

print "\n"

#KNN accuracy and time elapsed caculation

t6=time()

print "KNN"

# knn = KNeighborsClassifier(n\_neighbors=3)

knn = KNeighborsClassifier()

clf\_knn=knn.fit(X\_train, y\_train)

print "Acurracy: ", clf\_knn.score(X\_test,y\_test)

t7=time()

print "time elapsed: ", t7-t6

# cross validation for KNN

tt6=time()

print "cross result========"

scores = cross\_validation.cross\_val\_score(knn, df[features], df['shares'], cv=5)

print scores

print scores.mean()

tt7=time()

print "time elapsed: ", tt7-tt6

print "\n"

# **Bibliography**

He Ren, Quan-Yang. (2015). Predicting and Evaluating the Popularity of Online News.

Liu, B. (2012). Online News Popularity dataset. 8.

S., G. T. (2015). FEATURE SELECTION AND CLASSIFICATION APPROACH FOR ONLINE NEWS POPULARITY. *Machine Learning and Applications: An International Journal (MLAIJ) Vol.2, No.2, June 2015*, 16.

Shailendra Kumar Singh, S. P. (2014). Approaches on Different Data set Domain: Survey. *International Journal of Database Theory and Application*, 12.

Vohra, S. (2013). A COMPARATIVE STUDY OF DATASETS. *Conference Paper · April 2014*, 6.