INFO I 590 - BIG DATA APPLICATIONS AND ANALYTICS

PROJECT REPORT

Computer Science
Fall 2015
Indiana University
Bloomington, IN

Ankit Sadana
asadana@indiana.edu

Jaini Vora
voraj@iu.edu

Github sources:

R-scripts : https://github.com/voraj/Online-News-Popularity

Java-transformation : https://github.com/asadana/ONP-data-transform

Online News Popularity

Project Aim:

With Internet expansion, the prediction of online news popularity is becoming a trendy research topic. In this project, we intend to predict if an article will become popular or not by using features like type of content, number of images/videos , the day on which the paper was published, etc. We are using a recently collected dataset which contains information about 40,000 articles from the Mashable website.

Importance:

There has been growing interest in online news with the expansion of Internet, which allows easy and wide spread of information around the globe. This is no longer limited to people browsing websites to find the articles; it is now also accessible through feeds, social media and mobile applications. Therefore, predicting the spread of information through online news is considered a hot topic. Popularity of information is often measured by considering the number of interactions in the Web and social networks (eg. Number of likes, views etc.), but the actual spreading of information can be estimated with the number of shares over the web and social network. Predicting the popularity is quite valuable for content publishers, authors, advertisers etc.

Dataset Source: UCI Machine Learning Repository
https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity

Dataset consists of 61 attributes: 58 predictive attributes, 2 non-predictive, 1 goal field

These attributes are as follows:

0. url: URL of the article (non-predictive)
1. timedelta: Days between the article publication and the dataset acquisition (non-predictive)
2. n_tokens_title: Number of words in the title
3. n_tokens_content: Number of words in the content
4. n_unique_tokens: Rate of unique words in the content
5. n_non_stop_words: Rate of non-stop words in the content
6. n_non_stop_unique_tokens: Rate of unique non-stop words in the content
7. num_hrefs: Number of links
8. num_self_hrefs: Number of links to other articles published by Mashable
9. num_imgs: Number of images

10. num_videos: Number of videos
11. average_token_length: Average length of the words in the content
12. num_keywords: Number of keywords in the metadata
13. data_channel_is_lifestyle: Is data channel 'Lifestyle'?
14. data_channel_is_entertainment: Is data channel 'Entertainment'?
15. data_channel_is_bus: Is data channel 'Business'?
16. data_channel_is_socmed: Is data channel 'Social Media'?
17. data_channel_is_tech: Is data channel 'Tech'?
18. data_channel_is_world: Is data channel 'World'?
19. kw_min_min: Worst keyword (min. shares)
20. kw_max_min: Worst keyword (max. shares)
21. kw_avg_min: Worst keyword (avg. shares)
22. kw_min_max: Best keyword (min. shares)
23. kw_max_max: Best keyword (max. shares)
24. kw_avg_max: Best keyword (avg. shares)
25. kw_min_avg: Avg. keyword (min. shares)
26. kw_max_avg: Avg. keyword (max. shares)
27. kw_avg_avg: Avg. keyword (avg. shares)
28. self_reference_min_shares: Min. shares of referenced articles in Mashable
29. self_reference_max_shares: Max. shares of referenced articles in Mashable
30. self_reference_avg_sharess: Avg. shares of referenced articles in Mashable
31. weekday_is_monday: Was the article published on a Monday?
32. weekday_is_tuesday: Was the article published on a Tuesday?
33. weekday_is_wednesday: Was the article published on a Wednesday?
34. weekday_is_thursday: Was the article published on a Thursday?
35. weekday_is_friday: Was the article published on a Friday?
36. weekday_is_saturday: Was the article published on a Saturday?
37. weekday_is_sunday: Was the article published on a Sunday?
38. is_weekend: Was the article published on the weekend?
39. LDA_00: Closeness to LDA topic 0
40. LDA_01: Closeness to LDA topic 1
41. LDA_02: Closeness to LDA topic 2
42. LDA_03: Closeness to LDA topic 3
43. LDA_04: Closeness to LDA topic 4
44. global_subjectivity: Text subjectivity
45. global_sentiment_polarity: Text sentiment polarity
46. global_rate_positive_words: Rate of positive words in the content
47. global_rate_negative_words: Rate of negative words in the content
48. rate_positive_words: Rate of positive words among non-neutral tokens

49. rate_negative_words: Rate of negative words among non-neutral tokens
50. avg_positive_polarity: Avg. polarity of positive words
51. min_positive_polarity: Min. polarity of positive words
52. max_positive_polarity: Max. polarity of positive words
53. avg_negative_polarity: Avg. polarity of negative words
54. min_negative_polarity: Min. polarity of negative words
55. max_negative_polarity: Max. polarity of negative words
56. title_subjectivity: Title subjectivity
57. title_sentiment_polarity: Title polarity
58. abs_title_subjectivity: Absolute subjectivity level
59. abs_title_sentiment_polarity: Absolute polarity level
60. shares: Number of shares (target)

Softwares Used:

RStudio (for R programming to generate models and plots)
Eclipse IDE (for Java programming to transform dataset)
Microsoft Excel/LibreOffice Calc (to preview datasets)

Steps Followed:

1) Dataset acquisition from UCI Machine Learning Repository
2) ETL - Created a new CSV file with relevant data for analysis using dataset acquired.
3) Analysis using various plot diagrams
4) Data prediction and analysis using Decision tree and Random forest generation

Step 1: Dataset acquisition from UCI Machine Learning Repository

We acquired the dataset from recently collected Mashable website in the form of "OnlineNewsPopularity.csv" consisting of 39,644 data samples.

Step 2: ETL - Created a new CSV file with relevant data for analysis using dataset acquired.

We extracted the data samples from the input CSV file and performed the below transformations on it.

- **Transformation of individual weekday columns to one column:** Columns 31-37 in the input dataset is used to specify if the article was published on which day of the week

(Monday to Sunday). There was a separate column for each day of the week, which has 1 to specify if the article was published on that day and 0 if not.

We collapsed these 7 columns into 1 weekday column containing the actual name of the day article was published. This allowed us to not only club the features together, but it allowed us to generate much more interpretable plots for the weekday for analysis.

Before Transformation

AF	AG	AH	AI	AJ	AK	AL
weekday_is_monday	weekday_is_tuesday	weekday_	weekday_	weekday_	weekday_	weekday_is_sunday
1	0	0	0	0	0	0

After transformation

Z
weekday
Monday

- **Transformation of individual types of data channels to one column:** Columns 13-18 specifies if the data channel is of type Lifestyle, Entertainment, Business, Social media, technology or World. Similar to weekday_is_* columns, these contained 1 if the current column was true, and 0 if not.

We collapsed these 6 columns into 1 column containing the actual name of the data channel that the article was related to. During this process we discovered there were data enteries that were false for all data_channel_is_*. We replaced those entry's value with “Unknown”. This allowed us to have a more robust dataset, while making the data more interpretable in plots/graphs.

Before Transformation

N	O	P	Q	R	S
data_channel_is_lifestyle	data_channel_is_entertainment	data_chan	data_char	data_char	data_channel_is_world
0	1	0	0	0	0

After transformation

M
data_channel
Entertainment

- **Removal of URL column:** Since the column ‘url’ is not useful in any kind of analysis or prediction, we have removed the column url from input dataset.
- **Transformation of column ‘shares’ from specific number of shares to Yes/No:** Number of shares of a particular article is directly proportional to its popularity. So in order to predict that the article is popular or not, we have set the threshold to 1400, i.e., if the number of shares is greater than threshold, then the value is replaced with “Yes” (popular), otherwise “No” (not popular).

The main purpose of doing this transformation was to create binary decision trees for analysis.

Before Transformation

BI
shares
593
711
1500

After transformation

AX
shares
No
No
Yes

After all these transformations, our input dataset had 49 columns.

The above transformations were done using the JAVA program given in Appendix A. The program writes the transformed data into new CSV file which we have used for various analysis and prediction.

Step 3: Analysis using various plot diagrams

For all the following steps, we have used “dat” as the variable to store our dataset, using the following command:

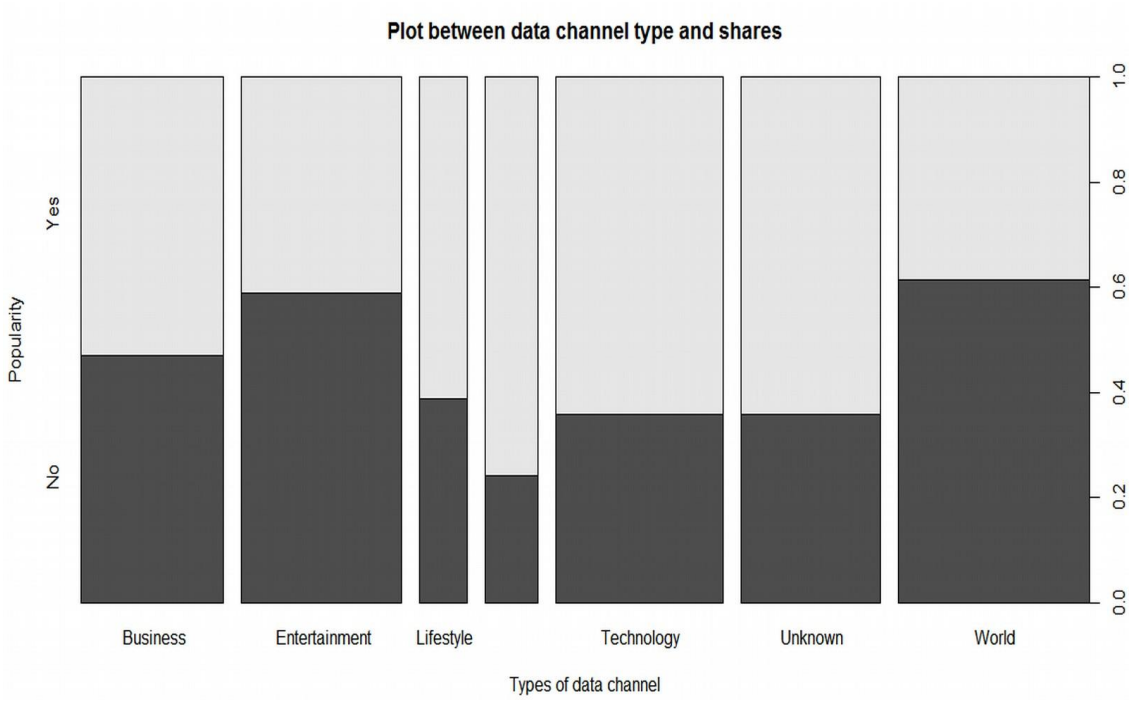
```
dat <- read.table("resources/OnlineNewsPopularity-new-binary-shares.csv", header = TRUE, sep = ",")
```

The code used for plotting the below plot diagrams can be found at https://github.com/voraj/Online-News-Popularity/blob/master/DataPlots.R

3.1) Analysis of importance of type of news to determine popularity

R code:

```
plot(dat$data_channel, dat$shares, type = "p",
     main = "Plot between data channel type and shares",
     xlab = "Types of data channel",
     ylab = "Popularity")
```



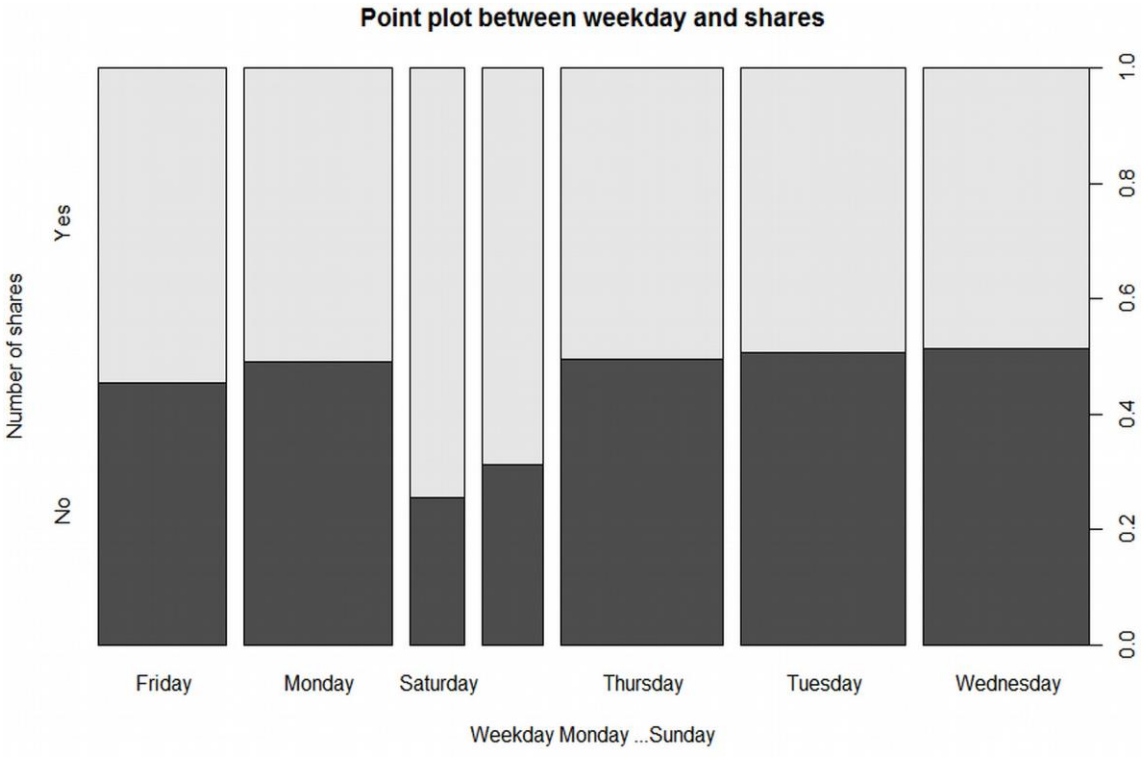
Insights:

- We see how important the type of data channel is determining its popularity.
- Social media and Lifestyle are the most popular types of data channel.
- There are 50 % chances that the news of category Business might be popular.
- News based on ‘Entertainment’ and ‘World’ are less popular (40% chances of being popular)

3.2) Analysis of importance of days of the week to determine popularity

R code:

```
plot(dat$weekday, dat$shares, type = "p",
     main = "Point plot between weekdays and shares",
     xlab = "Weekday Monday ...Sunday",
     ylab = "Number of shares")
```



Insights:

- We learn that news published on the weekends are more popular as compared to those published on the weekdays.
- One reason for this could be people are buzy on the weekdays with their work, so they do not get more time to listen to the news while on the weekends people have free time to hear the news. So the news published on the weekends have more chances to be popular.

3.3) Analysis for determining popularity of channel types published on ‘Sunday’

R code:

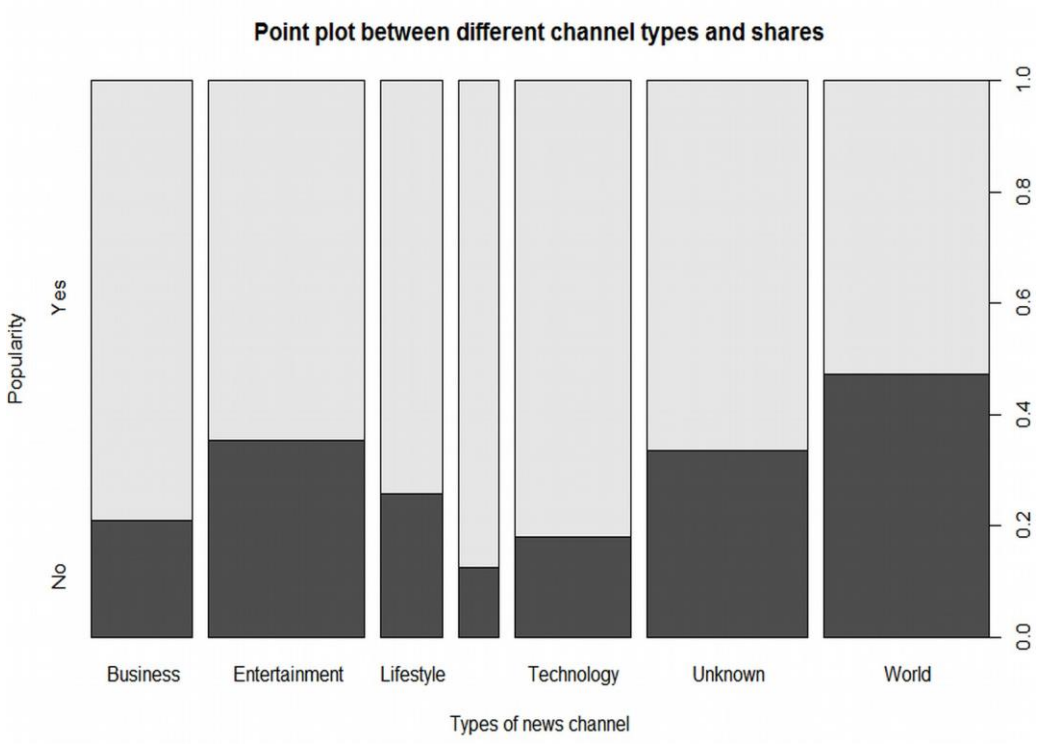
```
Sunday <- dat[grep("Sunday", dat$weekday), ]

    plot(Sunday$data_channel, Sunday$shares, type = "p",

        main = "Point plot between different channel types and shares",

        xlab = "Types of news channel",

        ylab = "Popularity")
```



Insights:

- We see that news belonging to categories like Business, Lifestyle and Social Media are more popular when published on Sunday
- News of category ‘Social Media’ are most popular when published on Sunday.
- News of category ‘World’ may or may not be popular when published on Sunday.

3.4) Analysis of popularity of ‘Entertainment’ news on different weekdays

R code:

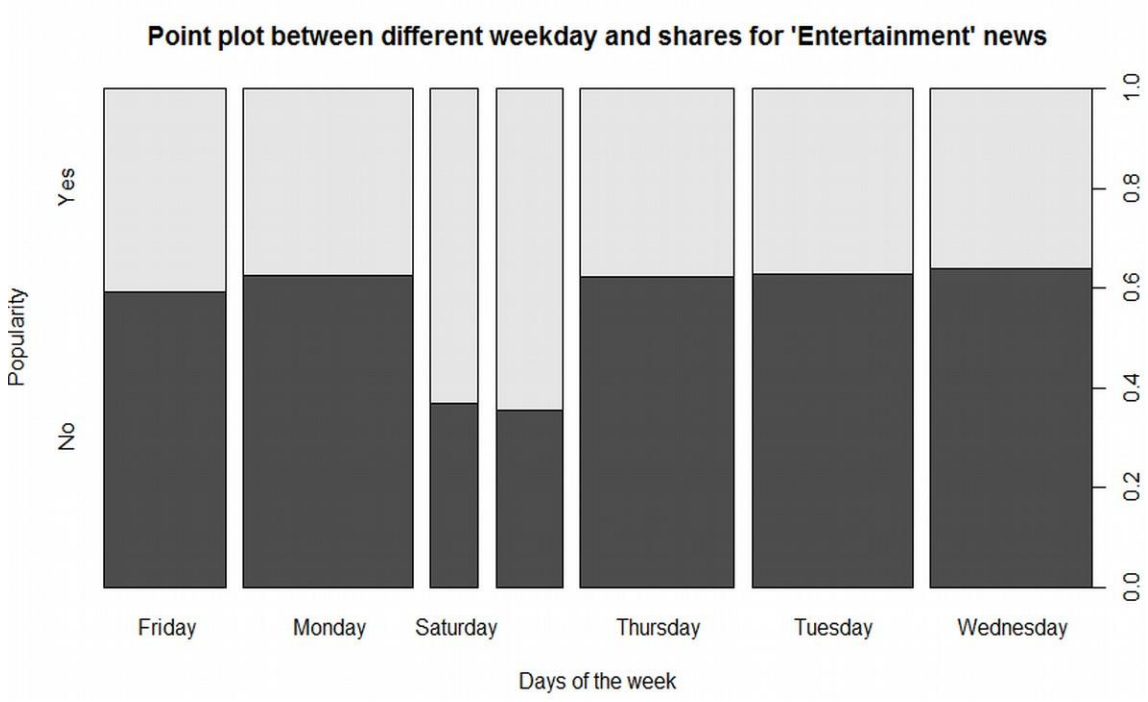
```
Bus <- dat[grep("Entertainment", dat$data_channel), ]

plot(Bus$weekday, Bus$shares, type = "p",

     main = "Point plot between different weekday and shares for 'Entertainment' news",

     xlab = "Days of the week",

     ylab = "Popularity")
```



Insights:

- We see that news belonging to ‘Entertainment’ category will be more popular on the weekends ie. Saturday and Sunday.
- There are less chances of ‘Entertainment’ news to be popular on the weekdays Monday to Friday.
- This might be because on weekends people get break from their office and try to amuse themselves to freshen up after the hectic week of work.

Step 4: Data prediction and analysis using Decision tree and Random forest generation

We have used different classification methods and various functions in R for various analysis and finding interesting insights.

4.1) Decision Tree: We have used "OnlineNewsPopularity-new-binary-shares.csv" which specifies that the news is popular or not (Yes or No) in shares. We have split the 39,644 input samples into 66% training data (26,165 samples) and used remaining as testing data. We built a decision tree model using the training data. For every sample in the testing data, we use the decision tree to predict if the news was popular or not. This prediction using the decision tree is then compared with the actual popularity attribute to generate a confusion matrix.

The R code for building the decision tree and prediction is given below (https://github.com/voraj/Online-News-Popularity/blob/master/DecisionTree.R)

```
library(rpart)
library(rpart.plot)

# Reading the database into dat
# OnlineNewsPopularity-new-binary-shares.csv : Yes/No values based on a threshold
dat <- read.table("resources/OnlineNewsPopularity-new-binary-shares.csv", header = TRUE, sep
= ",")

# Spliting dat into training and test
# Sample size is set to 66% of the rows in dat
sample_size <- floor(0.66 * nrow(dat))
# Seed is used here so that you can replicate this train/test dataset
set.seed(123)
# train_index <- sample matrix using sample_size from length(nrows)
train_index <- sample(seq_len(nrow(dat)), size = sample_size)

# Training and test data
train_dat <- dat[train_index, ]
test_dat <- dat[-train_index, ]

#Build a decision tree using rpart
dtree=rpart(shares~.,data=train_dat,control=rpart.control(10))

# Plotting a decision tree with most important features
rpart.plot(dtree)
```

```
#Make prediction using dtree
predictedDT = predict(dtree, test_dat, type="class")
print(predictedDT)
```

Insights:

- We studied the attributes that were most useful in generating the decision tree. The graphical structure of the decision tree and its textual representation is as follows:

```
1) root 26165 12338  Yes (0.4715460 0.5284540)
  2) data_channel= Business, Entertainment, World 14335  6167  No (0.5697942 0.4302058)
    4) kw_avg_avg< 2906.282 8957  3188  No (0.6440773 0.3559227) *
    5) kw_avg_avg>=2906.282 5378  2399  Yes (0.4460766 0.5539234) *
  3) data_channel= Lifestyle, SocialMedia, Technology, Unknown 11830  4170  Yes (0.3524937 0.6475063) *
```

Here we can see that data_channel was found to be the most important feature in predicting if the article was popular or not. kw_avg_avg was observed to be the second most feature.

- We further explored which attributes are important in deciding the popularity of the news using the decision tree method and what weightage each of these attributes carry.

```
> print(dtree$variable.importance)
data_channel   kw_avg_avg       LDA_04   kw_max_avg       LDA_02       LDA_03   kw_min_avg   kw_min_max   kw_avg_max
   612.08727    385.53603    235.15948    233.29363    140.00914    139.25262     88.72511     56.24320     53.00970
```

- For every sample in the testing data set, we use the decision tree model generated to decide if the news was popular or not. This prediction using the decision tree is then compared with the actual popularity attribute in order to compare true positives, true negatives, false positives and false negatives, this is also known as the Confusion Matrix.
- We also determined how effective the decision tree was in prediction by finding out the news popularity by using this confusion matrix.

```
predictedDT   No  Yes
         No  2832 1753
         Yes 3320 5574
```

Here we can see that, True Negative values is 2832 and True Positive value is 5574, while False Positive value is 1753 and False Negative value is 3320. Using this we can manually find out the accuracy of the above decision tree using the below formula

Accuracy = (True Positive + True Negative) / Sum of all test samples
= (2832+5574)/ (2832+5574+ 3320+ 1753)
= 8406 / 13479
= 0.623636 ie. Approx. 63%

We verified this accuracy using the mean function as below, which shows how much percentage of predicted decisions do not match with the actual decisions.

```
> print(mean(predictedDT!=test_dat$shares)) # 0.37% is not equal
[1] 0.3763632
```

- We also ran the decision tree for 3 samples in "predict.csv" and predicted if the news will be popular or not. The numerical shares of those test records were 1200, 2400 and 24300 respectively.

```
> print(predict(dtree,New, type=c("class")))
   1    2    3
  No   No  Yes
Levels:  No  Yes
```

Out of these, first is correctly predicted as unpopular, second is incorrectly predicted to be unpopular and third is correctly predicted as popular.

4.2) Random Forests: We used random forests to generate more advanced model of decision trees which implemented the ensemble methods.

The code for random forests generation in R(https://github.com/voraj/Online-News-Popularity/blob/master/RandomForest.R):

```
library(randomForest)

# Reading the database into dat
# OnlineNewsPopularity-new-binary-shares.csv : Yes/No values based on a threshold
dat <- red.table("resources/OnlineNewsPopularity-new-binary-shares.csv", header = TRUE,
        sep = ",")
```

```
# Spliting dat into training and test
# Sample size is set to 66% of the rows in dat
sample_size <- floor(0.66 * nrow(dat))
# Seed is used here so that you can replicate this train/test dataset
set.seed(123)
# train_index <- sample matrix using sample_size from length(nrows)
train_index <- sample(seq_len(nrow(dat)), size = sample_size)

# Training and test data
train_dat <- dat[train_index, ]
test_dat <- dat[-train_index, ]

# Splitting training data in features/label
trainFeatures <- train_dat[ , -(ncol(train_dat))]
trainLabel <- train_dat[ , (ncol(train_dat))]
# Splitting test data in features/label
testFeatures <- test_dat[ , -(ncol(test_dat))]
testLabel <- test_dat[ , (ncol(test_dat))]

# Creates the random forest with shares as the predicting label
rf=randomForest(x = trainFeatures, y = trainLabel, xtest = testFeatures , ytest = testLabel,
ntree=100, importance = TRUE)

plot(rf)  #plots the random forest
# creating a legend for rf
rfPlot.legend <- (if (is.null(rf$test$err.rate)) {colnames(rf$err.rate)}
else {colnames(rf$test$err.rate)})
# Putting the legend on the plot
legend("top", cex =1.0, legend=rfPlot.legend, lty=c(1,2,3), col=c(1,2,3), horiz=T)

# Generating Importance plots
varImpPlot(rf, type = 1)
```

Insights:

- We determined how effective the random forest was in prediction by finding out the news popularity by using the confusion matrix

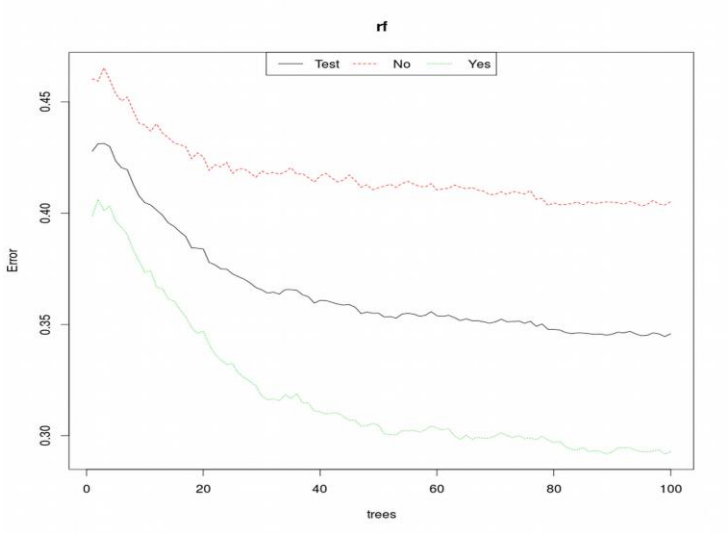
```
> print(rf$confusion)
      No  Yes class.error
No  7339 4999   0.4051710
Yes 4050 9777   0.2929052
```

Accuracy = (7339 + 9777) / (10533+13990+7957+7164)

= 17116/ 26165 = 0.654 ie. Approx. 65%

As expected, random forest had better accuracy and lower error rate as compared to decision trees. We tried running the random forest for 20, 50 and 100 trees.

- We learn that as more trees are generated in random forest the, error rate goes on decreasing. We have created random forest with 100 decision trees.

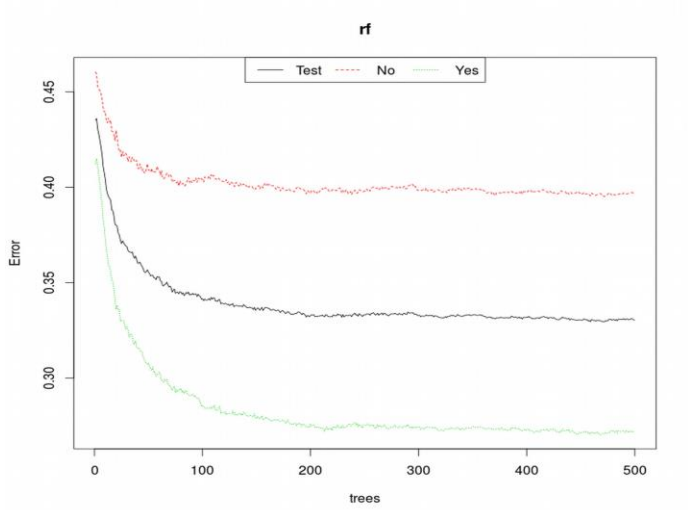


Our experiments with varied number of trees in the random forest showed what this plot conveys. The accuracy of our random forest increased from 62% at 20 trees to 65% at 100 trees. Here the black line represents the error rate for our test set in general while the green and red line represent the error rate of Yes and No label respectively. All three seem to be decreasing with the increase in the number of trees in random forest. But after a certain number of trees, the error rate line seems to be becoming more flat.

This hypothesis was confirmed when we ran the same train and test set for 500 trees in random forest.
This gave us an accuracy of ~66.9%, which is only a slight improvement from 100 trees.

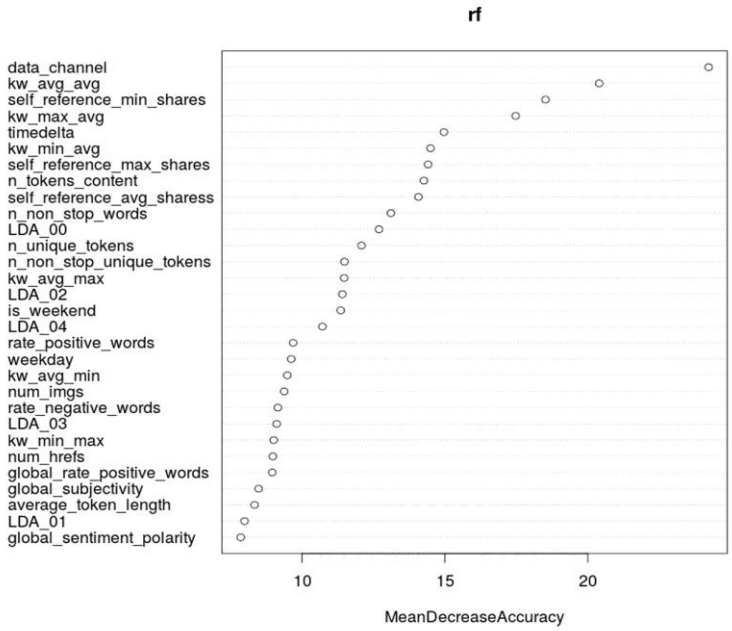
```
> print(rf$confusion)
      No   Yes class.error
No  7449  4889   0.3962555
Yes 3755 10072   0.2715701
```

As can be seen in this plot, the decrease in error rate for random forest starts to approach 0 after the number of trees has crossed a certain count.



- We also looked into the importance of each attribute in decision making of the news popularity.

The MeanDecreaseAccuracy showed us that the variables "data_channel" and "kw_avg_avg" were the most significant variables, which if dropped would decrease the model fitness significantly.



Conclusion:

The analysis of the data using various comparisons between the attributes and the label (shares) revealed that while some attributes influence the label greatly, like data_channel, there are attributes like weekday, which was expected to have influence, but ended up being almost constant for most of the weekdays.

Furthermore, we observed that generating a random forest model for this dataset performs slightly better than the decision tree model. Random forest in R also provided us with a much more comprehensive analysis of the dataset; however, decision tree model is much more interpretable for visual analysis.

Future Work:

Although random forest proved to be a better model than decision tree, in our opinion using a statistical transformation like Principal Component Analysis over the dataset in order to get better features based on correlation between the attributes and the label (shares) could prove to be a valuable improvement. This would not only reduce the correlation between the features themselves but also filter out the features that were not affecting the number of shares without having to rely on expert opinion.

References:

1. UCI database and feature list : https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity
2. K. Fernandes, P. Vinagre and P. Cortez. A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News. Proceedings of the 17th EPIA 2015 - Portuguese Conference on Artificial Intelligence, September, Coimbra, Portugal.
3. Random Forest package in R : https://cran.r-project.org/web/packages/randomForest/randomForest.pdf
4. Rpart package in R : https://cran.r-project.org/web/packages/rpart/rpart.pdf
5. Rpart.plot package in R : https://cran.r-project.org/web/packages/rpart.plot/rpart.plot.pdf
6. *Draw nicer Classification and Regression Trees with the rpart.plot package*, by Joseph Rickert : http://blog.revolutionanalytics.com/2013/06/plotting-classification-and-regression-trees-with-plotrpart.html
7. *A Brief Tour of the Trees and Forests*, by Wesley : http://www.r-bloggers.com/a-brief-tour-of-the-trees-and-forests/
8. Decision Trees in R Data Mining : http://www.rdatamining.com/examples/decision-tree
9. http://stackoverflow.com/a/17200430/4756508
10. http://stackoverflow.com/a/12868602/4756508
11. *Classification and Regression by randomForest*, by Andy Liaw and Matthew Wiener : http://www.bios.unc.edu/~dzeng/BIOS740/randomforest.pdf
12. randomForest {randomForest} : http://www.inside-r.org/packages/cran/randomforest/docs/randomforest

Appendix A: How to Run

1. **Java Dataset Transformation**

https://github.com/asadana/ONP-data-transform/blob/master/README.md

i. Ensure that the database is in resources/

i. Comment/Uncomment any of the function calls in Launcher.java, depending on the need for transformation

ii. Compile and execute Launcher.java

iii. Transformed database will be generated in resources/ with the current milliseconds appended to the name

2. **R scripts**

https://github.com/voraj/Online-News-Popularity/blob/master/README.md

b) The following libraries need to be installed for the scripts,

i. For DecisionTree.R : "rpart" and "rpart.plot"

ii. For RandomForest.R : "randomForest"

iii. DataPlots.R : none

c) To run the scripts,

i. Ensure that the database is in resources/

ii. Set the working directory to the current working directory

iii. Run the desired script

Appendix B: Data Transformation using Java

JAVA parser program to read and transform the Online News Popularity Database in order to reduce and transform the feature columns to make the plots more readable and interpretable for our Project. The below code can be found at https://github.com/asadana/ONP-data-transform

JAVA File 1: \\ Launcher.java

```
package com.onp.main;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import com.onp.util.DataEntry;

/*** Launcher : Main class to parse and transform our database
 * Project: Online News Popularity
 * Class: I-590 (Fall 2015)
 * @author asadana, voraj
 */

public class Launcher {

	// List to contain all data examples
	private static ArrayList<DataEntry> dataEntryList;
	// Temporary object to reference current DataEntry object
	private static DataEntry dataEntryObj;
	// Temporary object to reference current DataEntry object's features ArrayList
	private static ArrayList<String> tmpFeatures;
	// Temporary object to store and compare string
	private String checkString;

	// Folder name in local workspace which contains the source csv file
	// Also the folder where the new csv file will be generated
	private String folderName = "resources";
	// Source csv file
	private String inputFileName = "OnlineNewsPopularity.csv";
	// Target output csv file
	private String outputFileName = "OnlineNewsPopularity-" + System.currentTimeMillis() +
".csv";
	// Threshold to split shares into popular or not
```

```
private int labelThreshold = 1400;

// main function call
public static void main(String[] args) {
	System.out.println("Starting Main");

	Launcher objLauncher = new Launcher();
	dataEntryList = new ArrayList<DataEntry>();

	// Loading file
	objLauncher.loadFile();
	objLauncher.printPart();
	objLauncher.check();
	objLauncher.printPart();
	// Merging weekday_is_* into weekdays
	objLauncher.mergeDays();
	objLauncher.printPart();
	objLauncher.check();
	objLauncher.printPart();
	// Merging data_channel_* into data_channel
	objLauncher.mergeDataChannel();
	objLauncher.printPart();
	objLauncher.check();
	objLauncher.printPart();
	// Replacing shares' values with yes/no
	//objLauncher.splitLabels();
	// Writing the new database into a csv file
	objLauncher.writeCSV();
}

// Function to load the source csv file
private void loadFile() {

	// Variable to read streams from file
	BufferedReader br = null;
	// Temporary variable to read a line at a time
	String dataLine = "";
	// Temporary variable to store dataLine in parts
	String[] lineArray;
	// Delimeter to read the file by
	String splitBy = ", ";
	// Counter variable to count number of lines read
```

```
		int counter = 0;

		try {
			br = new BufferedReader (new FileReader (folderName + "/" + inputFileName));
			System.out.println("Loading successful.\nReading file...");

			// Read a line at a time till end of file
			while((dataLine = br.readLine()) != null) {
				counter++;
				dataEntryObj = new DataEntry();
				tmpFeatures = new ArrayList<String>();
				lineArray = dataLine.split(splitBy);

				// First entry is the name
				dataEntryObj.setName(lineArray[0]);
				// Next entries before the last are the features
				for (int i = 1; i < lineArray.length - 1; i++) {
					tmpFeatures.add(lineArray[i]);
				}
				dataEntryObj.setFeatures(tmpFeatures);
				// Last entry is the label
				dataEntryObj.setLabel(lineArray[lineArray.length - 1]);

				// Adding the DataEntry to the list
				dataEntryList.add(dataEntryObj);
			}
			System.out.println("Reading complete.\nNumber of lines read : " + counter);
		}
		catch (Exception e) {
			e.printStackTrace();
		}
}

/*
 *  Function to display,
			*	the size of the dataset
			*	the number of features
			*	names of the features
 */
private void check() {
		System.out.println("Size of the data: " + dataEntryList.size());
		System.out.println("Number of features: " + dataEntryList.get(0).getFeatures().size());
```

```
                System.out.println(dataEntryList.get(0).getFeatures());
        }

        // Function to merge weekday_is_* into weekdays
        private void mergeDays() {
                // Variable to store feature of the index where we merge
                int featureIndex = 0;
                // Get the title entry
                dataEntryObj = dataEntryList.get(0);
                // Loops through features to find weekday_is_monday
                for(int i = 0; i < dataEntryObj.getFeatures().size(); i++) {
                        if(dataEntryObj.getFeatures().get(i).contains("monday")) {
                                featureIndex = i;
                                System.out.println(dataEntryObj.getFeatures().get(i) + "\nFeatureIndex:
" + featureIndex);
                                dataEntryObj.getFeatures().set(featureIndex, "weekday");
                                System.out.println("Renaming weekday_is_monday to: " +
dataEntryObj.getFeatures().get(i));
                        }
                }

                // For every 1 in weekday_* column, the name of the day is added in the first column
                for(int i = 1; i < dataEntryList.size(); i ++) {
                        dataEntryObj = dataEntryList.get(i);
                        if (featureIndex != 0) {
                                // Starts on monday, ends at sunday
                                for(int j = featureIndex; j < featureIndex + 7; j++) {
                                        // for Monday
                                        if(j == featureIndex) {
                                                checkString = dataEntryObj.getFeatures().get(j);
                                                if (checkString.compareTo("1.0") == 0) {
                                        dataEntryObj.getFeatures().set(featureIndex, "Monday");
                                                }
                                        }
                                        // for Tuesday
                                        if(j == featureIndex + 1) {
                                                checkString = dataEntryObj.getFeatures().get(j);
                                                if (checkString.compareTo("1.0") == 0) {
                                        dataEntryObj.getFeatures().set(featureIndex, "Tuesday");
                                                }
                                        }
                                        // for Wednesday
```

```
                                    if(j == featureIndex + 2) {
                                    checkString = dataEntryObj.getFeatures().get(j);
                                      if (checkString.compareTo("1.0") == 0) {
                                    dataEntryObj.getFeatures().set(featureIndex, "Wednesday");
                                            }
                                    }
                                    // for Thursday
                                    if(j == featureIndex + 3) {
                                            checkString = dataEntryObj.getFeatures().get(j);
                                            if (checkString.compareTo("1.0") == 0) {
                                    dataEntryObj.getFeatures().set(featureIndex, "Thursday");
                                            }
                                    }
                                    // for Friday
                                    if(j == featureIndex + 4) {
                                            checkString = dataEntryObj.getFeatures().get(j);
                                            if (checkString.compareTo("1.0") == 0) {
                                            dataEntryObj.getFeatures().set(featureIndex, "Friday");
                                            }
                                    }
                                    // for Saturday
                                    if(j == featureIndex + 5) {
                                            checkString = dataEntryObj.getFeatures().get(j);
                                            if (checkString.compareTo("1.0") == 0) {
                                    dataEntryObj.getFeatures().set(featureIndex, "Saturday");
                                            }
                                    }
                                    // for Sunday
                                    if(j == featureIndex + 6) {
                                            checkString = dataEntryObj.getFeatures().get(j);
                                            if (checkString.compareTo("1.0") == 0) {
                                            dataEntryObj.getFeatures().set(featureIndex, "Sunday");
                                            }
                                    }
                            }
                    }
            }

            // from monday, next 6 columns need to be deleted
            deleteFeatures(featureIndex, 6);
    }
```

```
	// Function to merge data_channel_* to data_channel
	private void mergeDataChannel() {
		// Variable to store feature of the index where we merge
		int featureIndex = 0;
		// Get the title entry
		dataEntryObj = dataEntryList.get(0);
		// Loops through features to find data_channel_lifestyle
		for(int i = 0; i < dataEntryObj.getFeatures().size(); i++) {
			if(dataEntryObj.getFeatures().get(i).contains("lifestyle")) {
				featureIndex = i;
				System.out.println(dataEntryObj.getFeatures().get(i) + "\nFeatureIndex:
" + featureIndex);
				dataEntryObj.getFeatures().set(featureIndex, "data_channel");
				System.out.println("Renaming data_channel_is_lifestyle to: " +
dataEntryObj.getFeatures().get(i));
			}
		}

// For every 1 in data_channel_* column, the name of the data channel is added in the first column
		for(int i = 1; i < dataEntryList.size(); i ++) {
			dataEntryObj = dataEntryList.get(i);
			if (featureIndex != 0) {
				// Starts on lifestyle, ends at world
				for(int j = featureIndex; j < featureIndex + 6; j++) {
					// for Lifestyle
					if(j == featureIndex) {
						checkString = dataEntryObj.getFeatures().get(j);
						if (checkString.compareTo("1.0") == 0) {
						dataEntryObj.getFeatures().set(featureIndex, "Lifestyle");
						}
						else if (checkString.compareTo("0.0") == 0) {
					dataEntryObj.getFeatures().set(featureIndex, "Unknown");
						}
					}
					// for Entertainment
					if(j == featureIndex + 1) {
						checkString = dataEntryObj.getFeatures().get(j);
						if (checkString.compareTo("1.0") == 0) {
					dataEntryObj.getFeatures().set(featureIndex, "Entertainment");
						}
					}
					// for Business
```

```
                                        if(j == featureIndex + 2) {
                                                checkString = dataEntryObj.getFeatures().get(j);
                                                if (checkString.compareTo("1.0") == 0) {
                                                dataEntryObj.getFeatures().set(featureIndex, "Business");
                                                }
                                        }
                                        // for Social Media
                                        if(j == featureIndex + 3) {
                                                checkString = dataEntryObj.getFeatures().get(j);
                                                if (checkString.compareTo("1.0") == 0) {
                                        dataEntryObj.getFeatures().set(featureIndex, "SocialMedia");
                                                }
                                        }
                                        // for Tech
                                        if(j == featureIndex + 4) {
                                                checkString = dataEntryObj.getFeatures().get(j);
                                                if (checkString.compareTo("1.0") == 0) {
                                        dataEntryObj.getFeatures().set(featureIndex, "Technology");
                                                }
                                        }
                                        // for World
                                        if(j == featureIndex + 5) {
                                                checkString = dataEntryObj.getFeatures().get(j);
                                                if (checkString.compareTo("1.0") == 0) {
                                                dataEntryObj.getFeatures().set(featureIndex, "World");
                                                }
                                        }
                                }
                        }
                }

                // from monday, next 5 columnds need to be deleted
                deleteFeatures(featureIndex, 5);
        }

        // Function to split shares into Yes/No
        private void splitLabels() {
                String yesString = "Yes";
                String noString = "No";
                int tmpLabel;

                for(int i = 1; i < dataEntryList.size() ; i++) {
```

```
                        dataEntryObj = dataEntryList.get(i);
                        tmpLabel = Integer.parseInt(dataEntryObj.getLabel());

                        if (tmpLabel >= labelThreshold) {
                                dataEntryObj.setLabel(yesString);
                        }
                        else if (tmpLabel < labelThreshold) {
                                dataEntryObj.setLabel(noString);
                        }
                        else {
                                System.out.println("Error. Label value: " + dataEntryObj.getLabel());
                        }
                }
        }

        // Function to delete a set of columns by giving seed (featureIndex) and iterations
(numberOfForwardDeletes)
        private void deleteFeatures(int featureIndex, int numberOfForwardDeletes) {
                System.out.println("Removing features");
                for(DataEntry tmpDataEntry : dataEntryList) {
                        for(int i = featureIndex + 1; i <= (featureIndex + numberOfForwardDeletes);
i++)                                    {
                                tmpDataEntry.getFeatures().remove(featureIndex + 1);
                               }
                }
        }

        // Function to write the new database into csv file
        private void writeCSV() {
                String splitBy = ", ";
                FileWriter outputFile = null;
                try {

                        outputFile = new FileWriter(folderName + "/" + outputFileName);
                        System.out.println("Creating a file: " + outputFileName);
                        System.out.println("Writing to file..");

                        // Traversing the dataEntryList
                        for(DataEntry tmpData : dataEntryList) {
                                // commented out Name to remove URL column
                                //outputFile.append(tmpData.getName());
                                //outputFile.append(splitBy);
```

```
                                        // Traversing the feature list
                                        for (String tmpString : tmpData.getFeatures()) {
                                                outputFile.append(tmpString);
                                                outputFile.append(splitBy);
                                        }
                                        outputFile.append(tmpData.getLabel());
                                        outputFile.append("\n");
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        } finally {

                                try {
                                        System.out.println("Flushing out file");
                                        outputFile.flush();
                                        outputFile.close();
                                        System.out.println("All done");
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }

                }

// Function to print a divider in console
private void printPart() {
                System.out.println("\n\n=========================================\n\n");
                }
}
```

JAVA File 2 : \\ DataEntry.java

```
package com.onp.util;
import java.util.ArrayList;

/*** DataEntry: Class to store data examples in an ordered format
 * Project: Online News Popularity
 * Class: I-590 (Fall 2015)
 * @author asadana, voraj       */

public class DataEntry {
        private String name;
```

```
private ArrayList<String> features;
private String label;

public DataEntry() {
	setName("");
	setFeatures(new ArrayList<String>());
	setLabel("");
}

public void setDataEntry(String tmpName, ArrayList<String> tmpFeatures, String tmpLabel) {
	setName(tmpName);
	setFeatures(tmpFeatures);
	setLabel(tmpLabel);
}

public DataEntry getDataEntry() {
	return this;
}

public ArrayList<String> getFeatures() {
	return features;
}

public void setFeatures(ArrayList<String> features) {
	this.features = features;
}

public String getLabel() {
	return label;
}

public void setLabel(String label) {
	this.label = label;
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}
```