@Starbucks

Text Analysis to support Starbucks' Twitter Campaign

Introduction

My client for this capstone project is a global coffee company and a coffeehouse chain called Starbucks (SB). SB is known for lots of things: great coffee, friendly baristas, a near-complete takeover of practically every street corner in America and its excellent social media strategy. Here are some of the stats (as of July 2016):

- 36.3 million Facebook likes
- 11.7 million Twitter followers
- 9.8 million Instagram fans
- 4.7 million Google+ followers
- 259.5K Pinterest followers
- 100+K YouTube subscribers

The real power of social media is that it allows companies to possibly develop new products, and new revenue streams, by observing how customers interact with their product(s) and listening to their feedback, all to a degree that was previously unimaginable. It can also provide new avenues for bi-directional communication and deeper interaction with customers, fostering a closer relationship. An additional benefit is that agile companies can use all of these elements to more easily reposition a brand as a commercial environment changes. Starbucks, for quite some time now, has been on the cutting of edge of incorporating digital marketing and social media into its operations, allowing it to almost seamlessly merge its stores with the numerous digital marketing channels available. The company has also been adept at developing relationships with its customers, using the latest technology and social media platforms. Keeping this in mind, the aim of the project is to analyze SB's Twitter strategy which in particular, is very fascinating and unique.

The Hypothesis

Twitter as a new form of social media can potentially contain much useful information. Because tweets are compact and fast, Twitter has become widely used to spread and share breaking news, personal updates and spontaneous ideas. Several recent studies examined Twitter from different perspectives, including the topological characteristics of Twitter, tweets as social sensors of real-time events, the forecast of box-office revenues for movies, etc. However, the explorations are still in an early stage and our understanding of Twitter, especially its large textual content, still remains limited. Due to the nature of microblogging and the large amount of text in Twitter, Starbucks may presumably contain most of the useful information that can hardly be found in its traditional information sources such as the news media, distribution design, websites or company-owned retail stores.

But this does not clearly suggest if the popularity for Starbucks on Twitter is solely driven by its content. In order to properly evaluate this, the project is further developed on the theory that content is what drives SB's Twitter traffic. And, in order to test this hypothesis, we perform Topic Modelling and Linear Regression on a representative sample of SB's twitter data from July'16 and August'16. To begin with, we specifically try to answer the following questions:

- What are the types of topics covered on SB's twitter channel?
- What are the common characteristics of these latent topics?
- Do they trigger an information spread (In this case- retweet count)?

The Dataset

The dataset is created using Twitter's API for R using the 'search' method of the API. The Twitter Search API is part of Twitter's REST API. It allows queries against the indices of recent or popular Tweets and behaves similarly to, but not exactly like the Search feature available in Twitter mobile or web clients. The Twitter Search API searches against a sampling of recent Tweets published in the past 7 days.

Before getting involved, it's important to know that the Search API is focused on relevance and not completeness. This means that some Tweets and users may be missing from search results. For completeness we could consider using a ***Streaming API*** which does not fall under the scope of this project.

The best way to build a query and test if it's valid and will return matched Tweets is to first try it at ***twitter.com/search*** or using the ***Twitter advanced search query builder***. The query operator used in this project is **'@starbucks'** and the tweets are extracted for the months of July & August as permitted by the API. The extraction is preceded by a mandatory user authentication process. To understand the process in detail, please refer to ***https://dev.twitter.com/oauth/application-only***

To read about the API and understand the twitter authentication process in detail, please refer to ***https://dev.twitter.com/rest/public/search.***

User Authentication and Search Query Example:

```
# Twitter Authentication
library(twitteR)
library(ROAuth)

consumer_key <- 'xJZH0JuO2ndIby7kl0qCJ3Aq3'
consumer_secret <- 'AlTme1AbesN1v1ubKV6hBQOaNgsoWDD479C8UkB0TLIdFNwz9y'
access_token <- "251072504-L5UR9dgKCcmsrnGkzbas8p7jAUw3svX7qNiInnJz"
access_token_secret <- "WnsJ6KIm4E2Aaz7z1crBvAAS70whkM6eirExeIZhAN7oN"

request_url <- "https://api.twitter.com/oauth/request_token"
access_url <- "https://api.twitter.com/oauth/access_token"
auth_url <- "https://api.twitter.com/oauth/authorize"

twitCred <- setup_twitter_oauth(consumer_key=consumer_key,
                    consumer_secret=consumer_secret,
                    access_token = access_token,
                    access_secret = access_token_secret)

# Load tweets into R based on the maximum allowable limit of the query and convert outcome to dataframe
result1<-searchTwitter("@Starbucks",n=50)
result1df <- twListToDF(result1)
```

This process of querying is continued till the API rate limit is reached. For more understanding of the rate limit please refer to ***https://dev.twitter.com/rest/public/rate-limiting.*** We have two datasets, one for the month July (result_Max) and one for the month of August (result_max_new) and a total of 84090 observations

text	The actual text of the status account '@starbucks'
favorited	Indicates whether this Tweet has been liked by the authenticating user.
favoriteCount	Indicates approximately how many times this Tweet has been "liked" by Twitter users.
replyToSN	Screen name of the user this is in reply to
created	When this status was created
truncated	Whether this status was truncated
replyToSID	If the represented Tweet is a reply, this field will contain the string representation of the original Tweet's ID.
id	The integer representation of the unique identifier for this Tweet.
replyToUID	If the represented Tweet is a reply, this field will contain the integer representation of the original Tweet's author ID.
statusSource	Source user agent for this tweet
screenName	Screen name of the user who posted this status
retweetCount	The number of times this status has been retweeted
isRetweet	Whether this tweet is a retweet of another tweet or status

retweeted	TRUE if this status has been retweeted
longitude	Represents the geographic location of this Tweet as reported by the user or client application
latitude	Represents the geographic location of this Tweet as reported by the user or client application

Table 1: Unprocessed data attributes extracted using the Twitter API

Approach

We approach the task of analyzing the SB twitter data for significance of content in three stages:

- Data Preparation
- Topic Modelling
- In-Depth Analysis using Regression & Tree models

And, the following packages are installed and loaded to perform them:

```
# Load all the necessary packages
library(plyr)
library(dplyr)
require("NLP")
library(tm)
library(SnowballC)
library(topicmodels)
library(tree)
library(rpart)
library(party)
library(randomForest)
library(rpart.plot)
library(caret)
library(wordcloud)
library(data.table)
library(rattle)
```

Data Preparation

In this stage we get a sense of what is contained in the data. The benefit of text mining comes with the large amount of valuable information latent in texts which is not available in classical structured data formats. Text analysis involves several challenging process steps mainly influenced by the fact that texts, from a computer perspective, are rather unstructured collections of words. A text analyst typically starts with a set of highly heterogeneous input texts. So the first step is to import these texts into one's favorite computing environment, in our case R. Simultaneously it is important to organize and structure the texts to be able to access them in a uniform manner.

We started structuring our data by removing some of the columns that were unnecessary. The second step is tidying up the texts, including preprocessing the texts using regular expressions to obtain a convenient representation for later analysis.

```
# Combine both the datasets in to one and remove unnecessary columns from both the datasets
result_clean <- rbind(result_Max, result_max_new)
result_clean$favorited <- NULL
result_clean$favoriteCount <- NULL
result_clean$truncated <- NULL
result_clean$statusSource <- NULL
result_clean$retweeted <- NULL
result_clean$latitude <- NULL
result_clean$longitude <- NULL
result_clean$replyToSN <- NULL
result_clean$replyToSID <- NULL
result_clean$replyToUID <- NULL

# Arrange the data in descending order of the 'retweetCount' attribute
result_clean_sorted <- arrange(result_clean, desc(retweetCount))

# Convert the entire text to lower case
result_text_vector <- tolower(result_clean_sorted$text)
result_clean_sorted$text <- result_text_vector

# Remove old style retweets
result_text_vector = gsub("(RT|via)((?:\\b\\W*@\\w+)+)", "", result_text_vector)

# Remove everything except english letters and space- symbols, numbers, punctuations, emoticons, etc
result_text_vector = gsub("@\\w+", "", result_text_vector)
result_text_vector = gsub("[[:punct:]]", "", result_text_vector)
```

```
result_text_vector = gsub("[[:digit:]]", "", result_text_vector)
result_text_vector = gsub("http\\w+ |^ http | http", "", result_text_vector)
result_text_vector = gsub("^rt | rt", "", result_text_vector)

# Save cleaned dataset and continue with topic modelling
write.csv(result_clean_sorted, "result_final_new.csv", row.names = F)
```

Topic Modelling & Exploratory Data Analysis

We start topic modelling on our dataset by some more text processing to transform the texts into structured formats to be actually computed with. For "classical" text mining tasks, this normally implies the creation of a so-called document-term matrix, probably the most common format to represent texts for computation. For this there is a need for a conceptual entity similar to a database holding and managing text documents in a generic way: we call this entity a text document collection or corpus. To use this, we install and load the 'tm' package.

```
# Pull the text into a separate vector
result_final_vector <- result_final_new$text

# Remove any remaining space and update the dataset
result_final_vector <- gsub("^\\s+|\\s+$", "", result_final_vector)
result_final_new$text <- result_final_vector

# One of the main limitations in twitter analytics is that each retweet is identified as a separate text. This will have an adverse effect on our regression analysis that we will be performing shortly. So we have to remove all the duplicate text. The resultant dataset contains 40,512 unique observations
result_final_new <- result_final_new[!duplicated(result_final_new$text), ]

# Build a corpus, and specify the source to be character vector
result_corpus <- Corpus(VectorSource(result_final_new$text))

# Again, make sure that corpus contains only english letters and no extra spaces
result_corpus <- tm_map(result_corpus, removePunctuation)
result_corpus <- tm_map(result_corpus, removeNumbers)
removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
result_corpus <- tm_map(result_corpus, content_transformer(removeURL))
removeNumPunct <- function(x) gsub("[^[:alpha:][:space:]]*", "", x)
result_corpus <- tm_map(result_corpus, content_transformer(removeNumPunct))
result_corpus <- tm_map(result_corpus, stripWhitespace)
```

```
# Remove all the stopwords (default/defined)
myStopwords <- c("can", "say", "amp", "i", "isnt", "ok", "today", "dont", "worry", "eduaubdedubu", "me",
        "the", "hold", "against", "new", "also", "however", "tell", "will", "much","need", "take",
        "tend", "like", "particular", "rather", "said", "love", "get", "well", "make", "ask", "come",
        "first", "two", "help", "often", "may", "stand", "line", "internet", "youre", "you", "cool", "insanely",
        "might", "see", "thing", "some", "point", "look", "right", "now", "day", "etc", "quit", "drink",
        "hey", "that", "till", "tomorrow", "awsom", "aww", "awww", "awwww", "awwwwuufef", "awwwww")
result_corpus <- tm_map(result_corpus, removeWords, myStopwords)
result_corpus <- tm_map(result_corpus, removeWords, stopwords("english"))

# Build a document term matrix with text in each row being no less than 2 words.
result_dtm <- DocumentTermMatrix(result_corpus)
rowTotals_dtm <- apply(result_dtm, 1, sum)
result_dtm_nozero  <- result_dtm[rowTotals_dtm > 2, ]

# Inspect the top 500 words
freq.terms <- findFreqTerms(result_dtm_nozero, lowfreq = 500)

#freq.terms:
[1] "back"      "barista"   "brew"      "card"      "coconut"   "coffee"    "cold"
"cup"

[9] "drinks"    "free"      "going"     "great"     "ice"       "iced"      "macchiato"
"milk"

[17] "mocha"     "morning"   "please"    "really"    "starbucks" "still"     "store"
"tea"

[25] "thank"     "time"      "try"       "work"
```

A quick look at 'freq.terms' gives us an idea of the words that occur at least 500 times in the corpus. It is interesting to note that there aren't as many seasonal buzz -words (such as "berrysangria", "sangria") as one would expect from this outcome. A further analysis will help us evaluate this. Meanwhile, a word cloud could also be generated using the corpus to have a better visualization.


```
# Wordcloud
tdm <- TermDocumentMatrix(result_corpus)
m <- as.matrix(tdm)
word.freq <- sort(rowSums(m), decreasing = T)
d <- data.frame(word = names(word.freq),freq=word.freq)
head(d, 10)
wordcloud(words = d$word, freq = d$freq, min.freq = 350,
      max.words=68, random.order=FALSE, rot.per=0.35,
      colors=brewer.pal(9, "Set1"))
```

The word cloud clearly shows that "coffee" and "starbucks" are the two most important words. As observed earlier, names like "macchiato", "caramel", "coconut" are not as exposed as one would expect them to be and are rather dominated by the presence of regular words like "morning", "milk" and "tea".

Fig 1: Word cloud to visualize the frequently occurring words in the dataset

Latent Dirichlet Allocation (LDA)

In essence, LDA is a technique that facilitates the automatic discovery of themes in a collection of documents.
Intuitively, LDA provides a thematic summary of a set of documents (in our case, a set of tweets). It gives this summary by discovering 'topics', and telling us the proportion of each topic found in a document. To do so, LDA attempts to model how a document was 'generated' by assuming that a document is a mixture of different topics, and assuming that each word is 'generated' by one of the topics.

As a simple example, consider the following tweets:

- *Mornings are incomplete without starbucks coffee.*
- *Starbucks caramel macchiato is amazing.*

Let's remove stop words, giving:

- mornings starbucks coffee
- starbucks caramel macchiato

We'll let 'k' denote the number of topics that we think these tweets are generated from. Let's say there are 'k = 2' topics. Note that there are 'v = 6' words in our corpus. LDA would tell us that:

Topic 1 = mornings starbucks coffee	Tweet 1 = (3/3) Topic 1, (1/3) Topic 2
Topic 2 = caramel macchiato starbucks	Tweet 2 = (1/3) Topic 1, (3/3) Topic 2

We can conclude that there's a *morning coffee* topic and a caramel *macchiato* topic. Each topic in LDA is a probability distribution over the words. Now, we can jump into our code representing LDA to explore specifically what LDA does for our data.

```
# Implement Latent Dirichlet Allocation to extract six dominant themes in the text
result_lda <- LDA(result_dtm_nozero,k = 6)

# Inspect the first five terms in each topic
result_lda_term <- terms(result_lda, 5)
```

It is important to note that we come to this conclusion of 'k=6' topics and 'v=5' terms under each topic after several iterations and trials to check the optimum value. Too few topics result in heterogeneous set of words while too many diffuse the information with the same words shared across many topics. In our case six topics and five terms under each topic performed best and here's what the outcome looks like:

Topic 1	***Topic 2***	***Topic 3***	***Topic 4***	***Topic 5***	***Topic 6***
starbucks	*coffee*	*coffee*	*milk*	*coffee*	*coffee*
iced	*starbucks*	*starbucks*	*iced*	*free*	*starbucks*
coffee	*mocha*	*time*	*coconut*	*tea*	*morning*
card	*time*	*store*	*going*	*thank*	*milk*
caramel	*ever*	*work*	*macchiato*	*drinks*	*iced*

Table 2: Topics and terms derived using LDA

As observed in frequent terms and word cloud, there are more regular terms under each topic than certain buzz-words that we expect to identify.

Topic 1	***CaramelFix***
Topic 2	***MochaMania***
Topic 3	***StoreTime***
Topic 4	***CoconutEspresso***
Topic 5	***FreeDrinks***
Topic 6	***MorningMix***

Table 3: Topic Names

Moving forward, in order to accurately analyze the popularity quotient of each of these terms, it is important for us to first name each term and then, perform feature engineering on our dataset to create a statistical model from it. Meaning, our dataset has 40,512 unique observations or texts and each text will be a mixture of each of the thirty terms. In order to accurately determine that, we add binary features for each term and insert those binary features as separate columns.
For example, the code to separate columns for each term under the ***CaramelFix*** topic, will look like this:

```
# Add the 30 terms as 30 new columns in the dataset with each observation being a binary indicator of the presence of the term in the text
result_text_vector_new <- as.character(result_final_new$text)
result_final_new <- result_final_new %>% mutate(CaramelFix_1 = ifelse(grepl(pattern = "starbucks", x = result_text_vector_new), 1,0))
result_final_new <- result_final_new %>% mutate(CaramelFix_2 = ifelse(grepl(pattern = "iced", x = result_text_vector_new), 1,0))
result_final_new <- result_final_new %>% mutate(CaramelFix_3 = ifelse(grepl(pattern = "coffee", x = result_text_vector_new), 1,0))
result_final_new <- result_final_new %>% mutate(CaramelFix_4 = ifelse(grepl(pattern = "card", x = result_text_vector_new), 1,0))
result_final_new <- result_final_new %>% mutate(CaramelFix_5 = ifelse(grepl(pattern = "caramel", x = result_text_vector_new), 1,0))
```

We continue doing this for all the terms till we have 30 additional attributes in our dataset. Additionally we parse the 'created' column to form 11 extra columns. This feature helps us to identify if there a particular day, or time in the day that the featured terms attract traffic.

```
# Include 'month' from the 'created' attribute as a new column
result_final_new$month=sapply(result_final_new$created, function(x) {p=as.POSIXlt(x);format(p, "%m")})
```

```
# Include 'date' from the 'created' attribute as a new column
result_final_new$date=sapply(result_final_new$created, function(x) {p=as.POSIXlt(x);format(p, "%d %b")})

# Include 'day of the week' from the 'created' attribute as a new column
result_final_new$day=sapply(result_final_new$created, function(x) {p=as.POSIXlt(x);format(p, "%a")})

# Include two separate columns to identify weekdays and weekends
day_vector <- result_final_new$day
result_final_new <- result_final_new %>% mutate(wday = ifelse(grepl(pattern = "Mon|Tue|Wed|Thu|Fri", x = day_vector), 1,0))
result_final_new <- result_final_new %>% mutate(wend = ifelse(grepl(pattern = "Sat|Sun", x = day_vector), 1,0))

# Include 'hour' from the 'created' attribute as a new column
result_final_new$hour=sapply(result_final_new$created, function(x) {p=as.POSIXlt(x);format(p, "%H")})

# Process the hour value to include distinct columns for daytime, evening and night
time <- as.numeric(sub("(\\d{1,2}):.*", "\\1", result_final_new$hour))
result_final_new$LateNight <- ifelse(00 <= time & time <= 04, 1,0)
result_final_new$Morning <- ifelse(04 < time & time <= 12, 1,0)
result_final_new$Afternoon <- ifelse(12 < time & time <= 16, 1,0)
result_final_new$Evening <- ifelse(16 < time & time <= 20, 1,0)
result_final_new$Night <- ifelse(20 < time & time <= 23, 1,0)
```

The resultant dataset has 47 new columns with 41 new features. Given the nature of our dataset, the scope for exploratory data analysis is limited but a quick EDA before proceeding further with linear regression and trees, shows the following results.

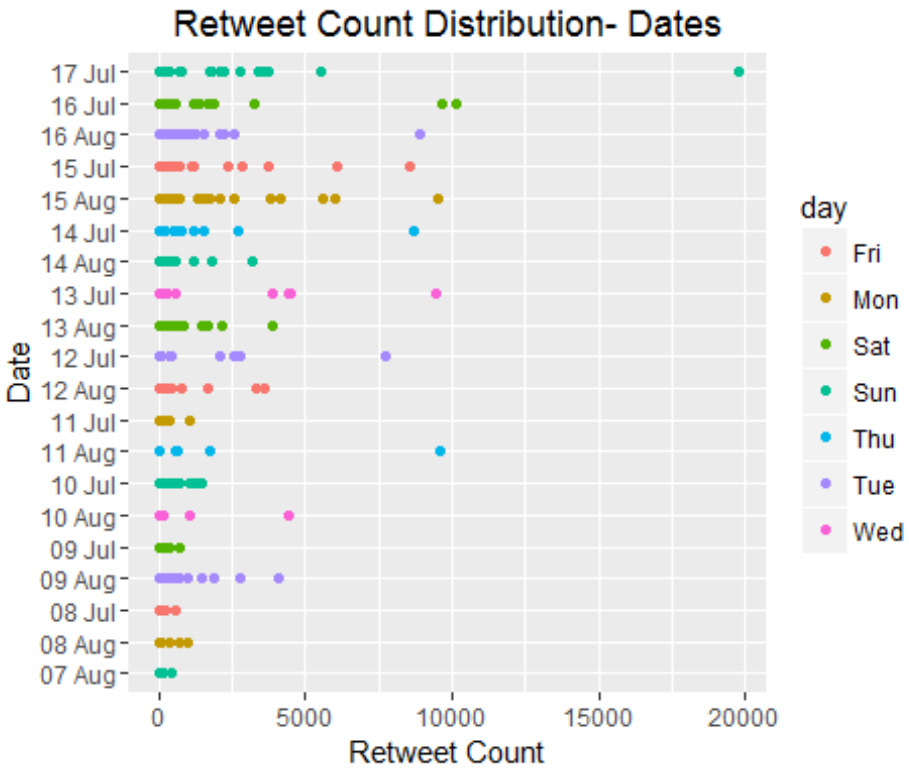

Fig 2: Distribution of Retweet Count over dates

```
# Retweet count distribution over 'dates' attribute
ggplot(result_final_new, aes(x = retweetCount, y = date)) +
geom_point(aes(col = day)) + xlab("Retweet Count") + ylab("Date") +
ggtitle("Retweet Count Distribution- Dates")
```

Fig 2 gives us an overview of the dates where the retweetCount is highest – 7/17/2016. A quick look at the data to identify the text, shows this:

"Ordered my drink @Starbucks Asked the barista if she wanted my name. She winked and said. "We gotcha" #JodieFoster"

This is a very interesting observation as it creates a possibility for us to develop this analysis towards social network analysis. The plot is also successful is giving us the comparable data between the same dates in each month and it can be observed that a large number of tweets have a retweet count of 0 to less than 5000 followed by the range of 5000-1000

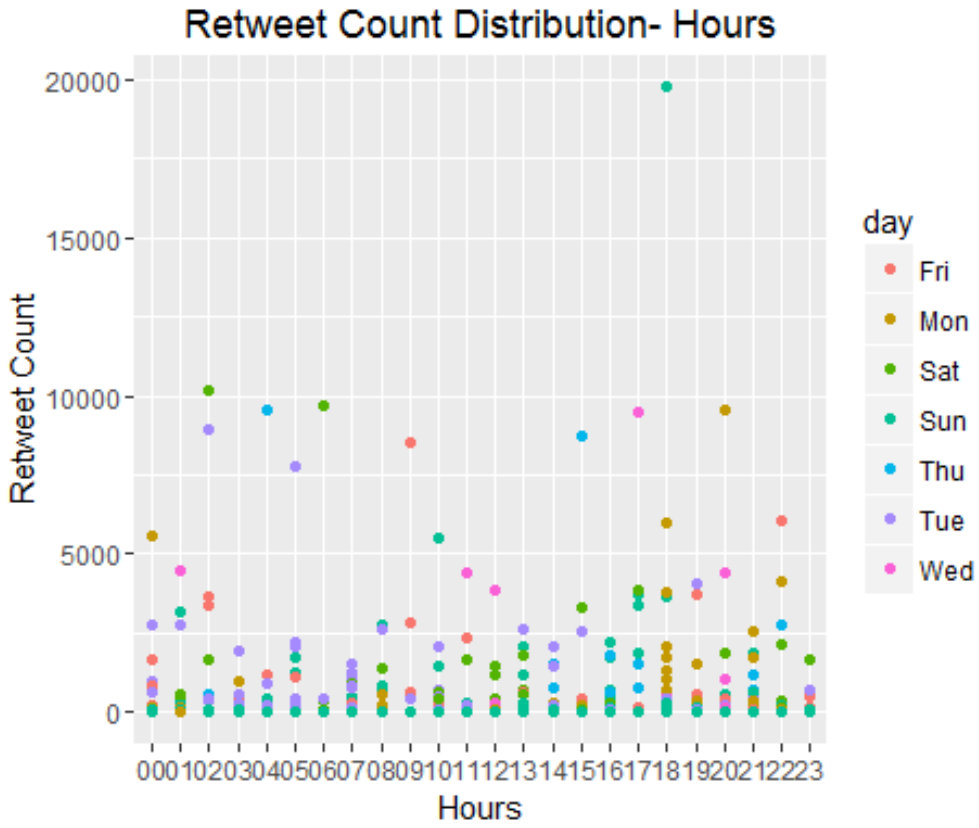

Fig 3: Distribution of Retweet Count over hours

It is also important to know the time at which there was maximum information spread and for that, we create a similar plot using the 'hour' attribute. If we exclude the one outlier, we can notice that most of the information spread took place either in the ***Latenight-Earlymorning*** or **Evening** periods.

```
#Retweet count distribution over 'hour' attribute
ggplot(result_final_new, aes(x = hour, y = retweetCount)) + geom_point(aes(col = day)) +
xlab("Hours") + ylab("Retweet Count") +
ggtitle("Retweet Count Distribution- Hours")
```

Another interesting visualization could be the information spread between weekdays and weekends.

```
# Retweet count distribution over 'day' attribute
ggplot(result_final_new, aes(x = day, y = retweetCount)) +
geom_point(aes(col = hour)) + facet_grid(wday ~.) +
geom_line(colour = "plum") + xlab("Days") + ylab("Retweet
Count") + theme(legend.position='none') + ggtitle("Retweet
Count Distribution- Days")
```

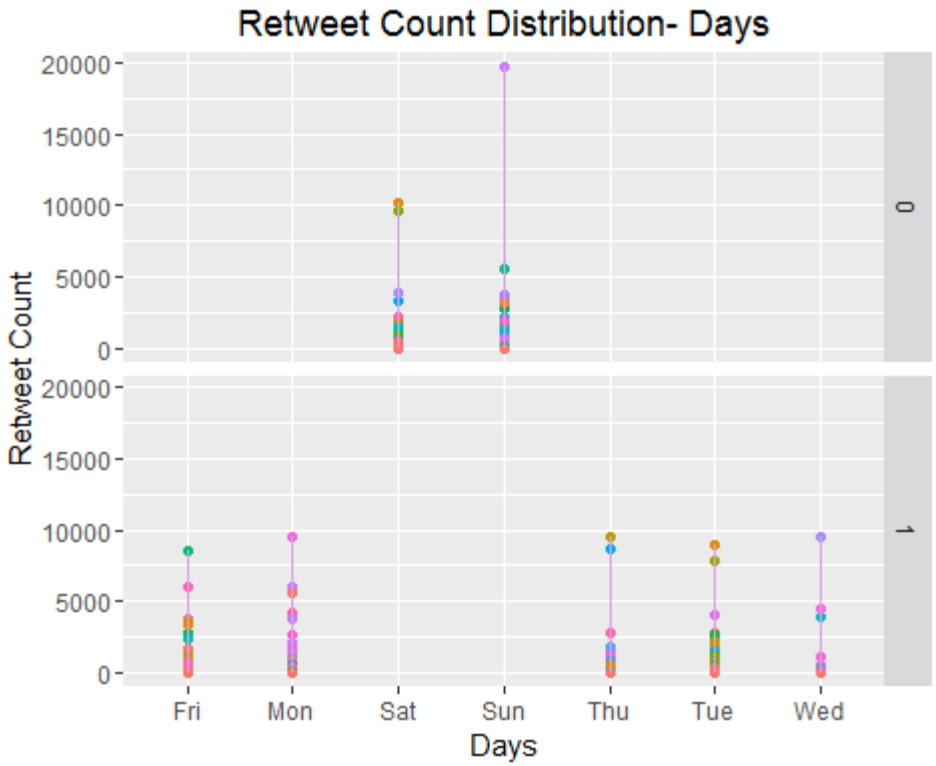

Fig 4: Distribution of Retweet Count: 'wend/wday' attribute

While the plot shows the numbers, we can a quick calculation to know the total number of retweet counts for weekdays and weekends to generally compare the two.

```
DT <- as.data.table(result_final_new)
DT[, list(totalretweetCount = sum(retweetCount), num = .N), by
= wday]
```

	wday	totalretweetCount	num
1:	0	126427	11609
2:	1	230985	28903

This means that we have 28903 weekday observations with a total retweet count of 230985 and 11609 weekend observations with a total of 126427. The data for wend is noticeably biased due to the one outlier with a retweet count of '19774'. Subtracting that from the original sum gives us '106653'

Linear Regression

One of the simplest and most frequently used techniques in statistics is linear regression where we investigate the potential relationship between a variable of interest (often called the response variable but there are many other names in use) and a set of one of more variables (known as the independent variables or some other term). Unsurprisingly there are flexible facilities in R for fitting a range of linear models from the simple case of a single variable to more complex relationships. In our project we will consider the case of multivariate linear regression with one response variable- 'retweetCount' and multiple independent variables.

At face value, model building appears straightforward: pick a modeling technique, plug in data, and generate a prediction. While this approach will generate a predictive model, it will most likely not generate a reliable, trustworthy model for predicting new samples. To get this type of model, we must first understand the data and the objective of the modeling. Upon understanding the data and objectives, we then pre-process and split the data. Only after these steps, do we finally proceed to building, evaluating, and selecting models.

The idea behind this analysis is to know if the 'retweetCount' has any kind of relationship with our new binary features or any other relevant attribute in the dataset representing content and time. To do this we will have to split our data. If we build the model on the entire dataset, there is no way to predict how the model will behave on new data. So the preferred practice is to split your dataset into a 80:20 sample (training:test), then, build the model on the 80% sample and then use the model thus built to predict the dependent variable on test data.

Doing it this way, we will have the model predicted values for the 20% data (test) as well as the actuals (from the original dataset). By calculating accuracy measures and error rates we can find out the prediction accuracy of the model.

```
# Split the dataset into training and testing set to perform linear regression and predict the model
# Split the data into 80:20 ratio
result_smp_size <- floor(0.80 * nrow(result_final_new))

 # set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(result_final_new)), size = result_smp_size)
train <- result_final_new[train_ind, ]
test <- result_final_new[-train_ind, ]
```

We now have a training set on which we can create our model with 32410 observations and 47 variables and a testing set on which we can predict our model with 8102 observations and 47 variables. The formula to create a predictive modelling for our dataset will be:

```
# Create the model
Model_1 <- lm(retweetCount ~ isRetweet + CaramelFix_2 + CaramelFix_3 + CaramelFix_5 + StoreTime_4 + Morning, data = train)
# Inspect the model
summary(Model_1)
```

The diagnostic measures for this model are:

```
Call:
lm(formula = retweetCount ~ isRetweet + CaramelFix_2 + CaramelFix_3 +
    CaramelFix_5 + StoreTime_4 + Morning, data = train)

Residuals:
    Min      1Q  Median      3Q     Max
 -193.6    -7.4     3.3     3.3 19774.5

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    -3.325      1.654  -2.010  0.04446 *
isRetweetTRUE  90.052      4.368  20.615  < 2e-16 ***
CaramelFix_2   41.874      7.486   5.594 2.24e-08 ***
CaramelFix_3   -7.381      4.147  -1.780  0.07509 .
CaramelFix_5   55.281     12.137   4.555 5.27e-06 ***
StoreTime_4    18.716      9.908   1.889  0.05891 .
Morning        10.752      3.545   3.033  0.00242 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 242.3 on 32403 degrees of freedom
Multiple R-squared:  0.01513,  Adjusted R-squared:  0.01495
F-statistic: 82.96 on 6 and 32403 DF,  p-value: < 2.2e-16
```

#	Name	Description
1	Residuals	The residuals are the difference between the actual values of the variable you're predicting and predicted values from your regression--$y - \hat{y}$. For most regressions you want your residuals to look like a normal distribution when plotted.
2	Significance Stars	The stars are shorthand for significance levels, with the number of asterisks displayed according to the p-value computed. *** for high significance and * for low significance. In this case, *** indicates that it's unlikely that no relationship exists b/w CaramelFix_2 (the term 'iced') and retweetCount.
3	Estimated Coefficient	The estimated coefficient is the value of slope calculated by the regression. It might seem a little confusing that the Intercept also has a value, but just think of it as a slope that is always multiplied by 1. This number will obviously vary based on the magnitude of the variable you're inputting into the regression, but it's always good to spot check this number to make sure it seems reasonable.
4	Standard Error of the Coefficient Estimate	Measure of the variability in the estimate for the coefficient. Lower means better but this number is relative to the value of the coefficient. As a rule of thumb, you'd like this value to be at least an order of magnitude less than the coefficient estimate. For instance in our case, the standard error for the StoreTime_4 variable is 9.908 which is almost 2x less than the estimate of the coefficient
5	t-value of the Coefficient Estimate	Score that measures whether or not the coefficient for this variable is meaningful for the model. You probably won't use this value itself, but know that it is used to calculate the p-value and the significance levels.
6	Variable p-value	Probability the variable is NOT relevant. You want this number to be as small as possible. If the number is really small, R will display it in scientific notation. In our example 2.2e-16 means that the odds that retweetCount is meaningless is about 1/5000000000000000
7	Significance Legend	The more punctuation there is next to your variables, the better. Blank=bad, Dots=pretty good, Stars=good, More Stars=very good
8	Residual Std Error / Degrees of Freedom	The Residual Std Error is just the standard deviation of your residuals. You'd like this number to be proportional to the quantiles of the residuals in #1. For a normal distribution, the 1st and 3rd quantiles should be 1.5 +/- the std error.

		The Degrees of Freedom is the difference between the number of observations included in your training sample and the number of variables used in your model (intercept counts as a variable).

9	*R-squared*	*Metric for evaluating the goodness of fit of your model. Higher is better with 1 being the best. Corresponds with the amount of variability in what you're predicting that is explained by the model.*
		While a high R-squared indicates good correlation, ***correlation does* not *always imply causation.***
		
		`cor(result_final_new$retweetCount, result_final_new$CaramelFix_2)`
		`[1] 0.03877479`
		
		For instance, even if the the correlation between retweetCount and CaramelFix_2 (term: 'iced') is positive and significant, it does not necessarily mean that word 'iced' is causing the retweetCount to be high. Given the fact that our data is textual and not numeric, it was not possible to determine the correlation between all the variables in the dataset.

10	*F-statistic & resulting p-value*	*Performs an F-test on the model. This takes the parameters of our model (in our case we only have 1) and compares it to a model that has fewer parameters. In theory the model with more parameters should fit better and have a lower p-value.* *The DF, or degrees of freedom, pertains to how many variables are in the model. In our case there are 24 variables.*

Moving forward, as one can notice, the multiple r-squared and adjusted r-squared values for the model are quite low. In general, adjusted *R*-square is a modification of R-square that adjusts for the number of terms in a model. Multiple *R*-square always increases when a new term is added to a model, but adjusted R-square increases only if the new term improves the model more than would be expected by chance.

R-squared is always between 0 and 100%:

- 0% indicates that the model explains none of the variability of the response data around its mean.
- 100% indicates that the model explains all the variability of the response data around its mean.

Since our r-squared value is closer to 0% we can infer that our model does not clearly explain the variability of the response data. But we also want to understand if our model does well on test data or data it has never seen before.

```
# Test the model on 'test' data
result_predict <- predict(Model_1, newdata = test)

# Inspect predicted model for difference in predicted and observed value of the test data
head(test$retweetCount)
10          20          24          34          38          52
4483        2769        2360        1696        1412        698

head(result_predict)
        10          20          24          34          38          52
 86.72743   194.63542   139.35420   97.47972   97.47972   97.47972

# And create a dataframe
lmValues1 <- data.frame(obs = test$retweetCount, pred = result_predict)

# Inspect the predicted RMSE & R-squared
defaultSummary(lmValues1)

RMSE      Rsquared
153.94716132   0.02965626
```

As expected, the model shows a poor accuracy on the test set but interestingly, as one can notice that the out of sample r-squared is greater than the original r-squared which is 0.02002. The RMSE is the square root of the variance of the residuals. It indicates the absolute fit of the model to the data–how close the observed data points are to the model's predicted values. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit. RMSE is a good measure of how accurately the model predicts the response, and is the most important criterion for fit if the main purpose of the model is prediction. If the out-of-sample *mean squared error*, also known as the *mean squared prediction error*, is substantially higher than the in-sample mean square error, this is a sign of deficiency in the model.

Trees

Given the deficiency of our linear model, it is highly unlikely that we can improve the outcome with single regression trees. But, it is a good practice to consider them. Basic regression trees partition the data into smaller groups that are more homogenous with respect to the response. To achieve outcome homogeneity, regression trees determine:

- The predictor to split on and value of the split
- The depth or complexity of the tree
- The prediction equation in the terminal nodes.

Two widely used implementations for single regression trees in R are **rpart** and **party**. The rpart package makes splits based on the CART methodology using the rpart function, whereas the party makes splits based on the conditional inference framework using the ctree function. Both rpart and ctree functions use the formula method.

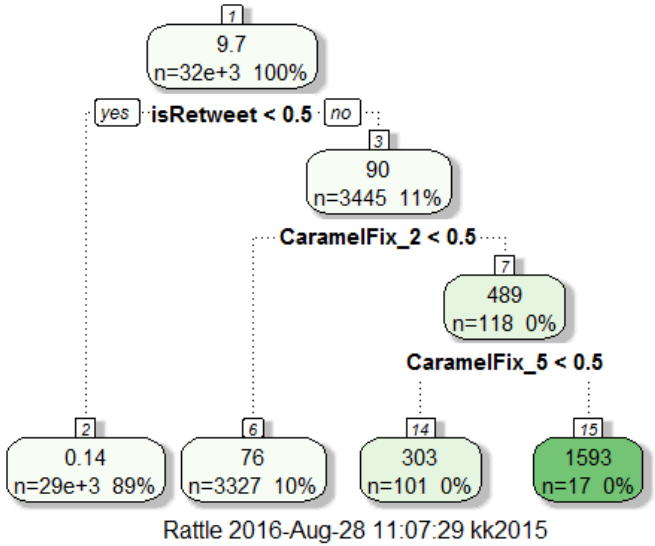

Fig 5: Single regression tree using rpart package: The splits do not involve the same number of predictors

```
# Single Regression trees
# with rpart package
myFormula <- retweetCount ~ isRetweet + CaramelFix_2 + CaramelFix_3 +
CaramelFix_5 + StoreTime_4 + Morning
rpart_model <- rpart(myFormula, data = train)
predict_rpart <- predict(rpart_model, newdata = test)
head(predict_rpart)
```

```
   10          20          24          34          38          52
76.30718 1592.52941  303.03960   76.30718   76.30718   76.30718
```

```
fancyRpartPlot(rpart_model)

postResample(predict_rpart, test$retweetCount)
```

```
RMSE    Rsquared
144.129987   0.150209
```

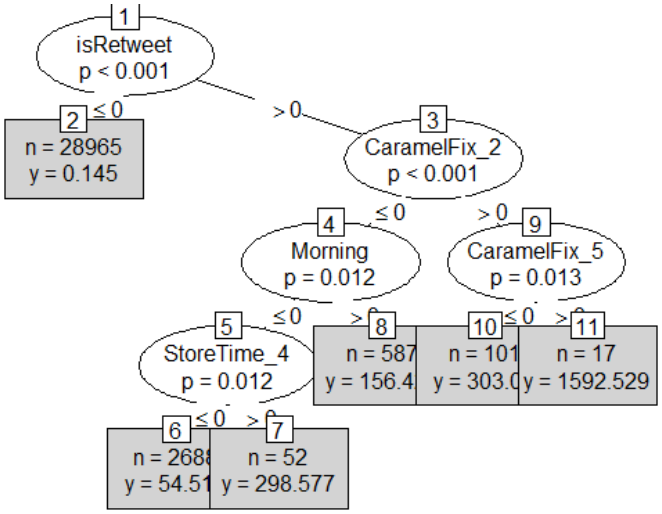

Fig 6: Single Regression Tree using ctree package:

```
# with party package
ctree_model <- ctree(myFormula, data=train)
predict_ctree <- predict(ctree_model, newdata = test)
head(predict_ctree)
```

```
  retweetCount
[1,]      54.5119
[2,]    1592.5294
[3,]     303.0396
[4,]     156.4225
[5,]     156.4225
[6,]     156.4225
```

```
plot(ctree_model, type="simple")
postResample(predict_ctree, test$retweetCount)
```

```
RMSE     Rsquared
144.9320012   0.1396499
```

Random Forest

Random Forest was designed to improve the prediction accuracy of CART methodology and works by building a large number of CART trees. Unfortunately this makes the method less interpretable. So we need to decide if we value the interpretability or the increase in accuracy more. To make a prediction for a new observation, each tree in the forest 'votes' for an outcome and we pick the outcome that receives the majority of the votes. Each tree can split on only a random subset of variables and each tree is built from what we call a 'bagged' or a 'bootstrapped' sample of data.

For instance if the original data is: 1, 2, 3, 4, 5
The new data could be:
Tree 1: 3, 2, 1, 2, 4
Tree 2: 4, 1, 5, 2, 3, and so on.

Five data points are used when constructing each CART tree. Since the sampling is random, it is highly possible that the data points could be recurring. Since each tree sees a different set of variables and a different set of data, we get what is called a forest. The number of trees is decided by the 'ntree' parameter which should neither be too small (bagging procedure may miss observations) nor too big (hard to interpret and will take longer to build).

varImpPlot(rf_model)

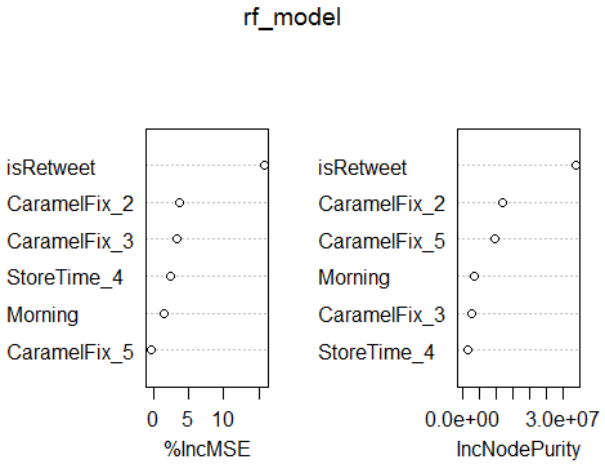

Fig 7: Dot chart of variable importance as measured by Random Forest

```
# Random Forest
rf_model <- randomForest(myFormula, data=train, importance = T, ntree= 100)
predict_rf <- predict(rf_model, newdata = test)
head(predict_rf)
```

```
       10        20        24        34        38        52
 54.69354 984.97765 244.50212 132.17526 132.17526 132.17526
```

```
importance(rf_model)
```

```
           %IncMSE IncNodePurity
isRetweet    15.7118084      33628044
CaramelFix_2  3.7063003      11952700
CaramelFix_3  3.4273785       2590935
CaramelFix_5 -0.3298282       9653486
StoreTime_4   2.4423517       1542611
Morning       1.4469124       3622104
```

```
postResample(predict_rf, test$retweetCount)
```

```
      RMSE   Rsquared
147.4209753   0.1237538
```

Conclusion

In this analysis, we looked at a very significant feature of text analysis: Topic Modelling. We started with data extraction using the twitter API for R followed by simple data tidying techniques and a twitter-LDA model designed for short texts. We settled with in-depth analysis of data using linear regression, single trees and random forest. While the hypothesis was interesting, given the nature of our text data, our regression and tree models were deficient and did not help us prove our hypothesis. A quick overview of the predicted and observed test data values for all the four models confirms that.

Top six values for actual retweet count-test data	***4483***	***2769***	***2360***	***1696***	***1412***	***698***
Values predicted by Linear Model	*86.72743*	*194.63542*	*139.35420*	*97.47972*	*97.47972*	*97.47972*
Values predicted by Rpart model	*76.30718*	*1592.52941*	*303.03960*	*76.30718*	*76.30718*	*76.30718*
Values predicted by CART model	*54.5119*	*1592.5294*	*303.0396*	*156.4225*	*156.4225*	*156.4225*
Values predicted by Random Forest	*54.69354*	*984.97765*	*244.50212*	*132.17526*	*132.17526*	*132.17526*

Fig 8: Table to compare top six 'retweetCount' observations for test data using all the four models.

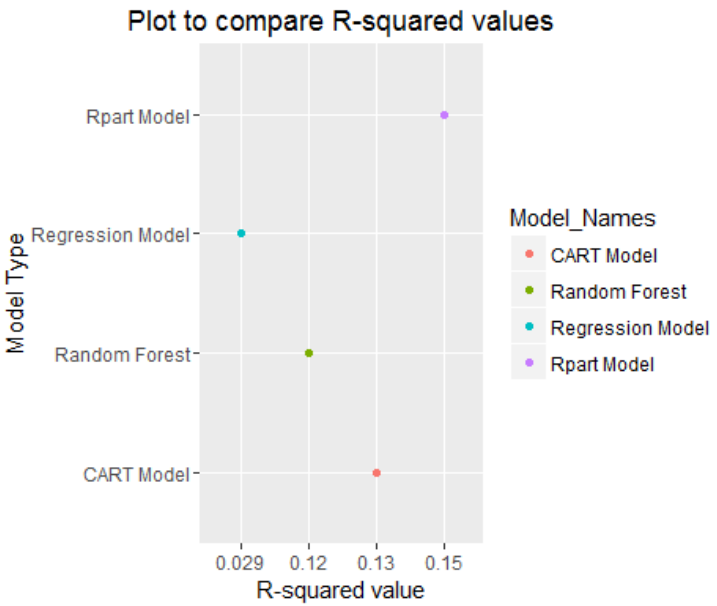

Fig 9: Comparison of the all the predicted R-squared values

```
# Plot r-squared for all the three models

Model_rsquared <- c("0.029", "0.15", "0.13", "0.12")

Model_Names <- c("Regression Model", "Rpart Model", "CART
Model", "Random Forest")

Model_compare <- data.frame(Model_rsquared =
Model_rsquared, Model_Names = Model_Names)

ggplot(Model_compare, aes(x = Model_rsquared, y =
Model_Names)) + geom_point(aes(col = Model_Names)) +
xlab("R-squared value") + ylab("Model Type") + ggtitle("Plot to
compare R-squared values")
```

In conclusion, our analysis revealed some interesting findings. While twitter can be a good source of entity-oriented topics that have low coverage on traditional news media, we find that such content may not be solely responsible for Starbucks' Twitter stardom. This very interestingly directs us towards the argument that it is not just content but also distribution that matters for brand popularity and, by this means, we can contemplate towards the less popular research approach of "content-based social network analysis" for Starbucks.

We certainly know that text analysis and social network analysis are not the same thing. Text analysis allows capturing and analyzing the text, while neglecting the social structure. Whereas, social network analysis provides powerful methods to study the relationships between people expressed as binary or weighted adjacency matrices. It can be used to find influential or popular nodes, communities and informal hierarchies. Social network analysis is a push proposition and text analysis, on the other hand, is a pull proposition. Our analysis can further be developed to combine these two methods. This combination can be used to describe people's interests and to find out if twitter users who have similar interests actually communicate.

Alternatively, keeping this analysis as the basis we can compare the influence of Starbucks' twitter content with other social or traditional news media to know if the distributions of topic categories and types differ in Twitter and in other media. We can also know if there are there specific topics covered in Twitter but rarely covered in other media and vice versa. If so, are there common characteristics of these specific topics, do certain categories and types of topics trigger more information spread in Twitter than in other media and so on.

Limitations and Future Work

Based on the results of our data analysis we can see that there is certainly more need for domain research and more importantly additional data. The existing dataset had the following limitations:

- Our text data was highly unstructured and did not always comply with the rules of topic modelling and regression analysis.
- This brings us to the second limitation. In the extracted data each retweet is identified as a separate text. This had an adverse effect on our regression analysis. To avoid this, we had to remove all the duplicate text and this in turn resulted in a poor regression model.
- In order to accurately determine the information spread, it is important for us to have more data and additional attributes such as total followers/friends, geolocations for the different screen names in the dataset. In addition to the memory issues, we had challenges with the Twitter API's inconsistency, due to which we were unable to collect this more potentially useful information.

Owing to the availability of more data, the learned topic models can definitely complement a more complicated social network analysis. Our initial results show that topic models are able to obtain meaningful categories from unsupervised data and show promise in revealing network-like statistics to support Starbucks' social media popularity.

Acknowledgements

I would like to express gratitude to my supervisor Matt Fornito for his valuable guidance and engagement through the learning process of this capstone project.

References:

- Max Kuhn, Kjell Johnson: Applied Predictive Modelling (2013)
- Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan and Xiaoming Li1: Comparing Twitter and Traditional Media using Topic Models (2011)
- Angela Bohn, Ingo Feinerer, Kurt Hornik and Patrick Mair: Content based Social Network Analysis on Mailing Lists (2011)
- http://blog.yhat.com/posts/r-lm-summary.html
  https://eight2late.wordpress.com/2015/09/29/a-gentle-introduction-to-topic-modeling-using-r/