Twitter Sentiment Analysis: Part 1

Kevin Bonds

The following is an analysis of the *Twitter Sentiment Analysis Dataset* available at: http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/. I will attempt to use this data to train a model to label unseen tweets into **"Positive"** or **"Negative"** sentiment. I will walk through my methodology and include code.

The following is an analysis of the *Twitter Sentiment Analysis Dataset* available at: http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/. I will attempt to use this data to train a model to label unseen tweets into **"Positive"** or **"Negative"** sentiment. I will walk through my methodology and include code. The github repo for my work can be found here: https://github.com/kwbonds/TwitterSentimentAnalysis. [1]

Libraries Used

```
library(tidyverse)
library(readr)
library(ggplot2)
library(caret)
library(knitr)
library(quanteda)
library(doSNOW)
library(gridExtra)
library(quanteda.textplots)
library(kableExtra)
library(formatR)
```

```
raw_tweets <- readRDS("raw_tweets.rds")
```

Load Data from .zip file

The Data Take a quick look at what we have.

```
# Examine the structure of the raw_tweets dataframe
str(raw_tweets)
```

```
## tibble [78,931 x 4] (S3: tbl_df/tbl/data.frame)
##  $ ItemID         : num [1:78931] 8 9 18 26 35 89 129 141 161 204 ...
##  $ Sentiment      : num [1:78931] 0 1 1 0 0 0 1 0 0 1 ...
##  $ SentimentSource: chr [1:78931] "Sentiment140" "Sentiment140" "Sentiment140" "Sentiment140" ...
##  $ SentimentText  : chr [1:78931] "Sunny Again        Work Tomorrow  :-|       TV Tonight" "handed i
##  - attr(*, "problems")= tibble [27 x 5] (S3: tbl_df/tbl/data.frame)
```

[1] The file is $>$ 50 MB, so I have taken a stratified sample and loaded it for this example (mainly so that this website will load). If you want to begin with the original, you will need to download it from the source above and read it into your working directory as an object named *raw_tweets*.

```
##   ..$ row     : int [1:27] 4285 4285 4286 4286 4287 4287 4287 4287 4287 4287 ...
##   ..$ col     : chr [1:27] "SentimentText" "SentimentText" "SentimentText" "SentimentText" ...
##   ..$ expected: chr [1:27] "delimiter or quote" "delimiter or quote" "delimiter or quote" "delimiter
##   ..$ actual  : chr [1:27] " " " " " " " " " " ...
##   ..$ file    : chr [1:27] "'./Sentiment Analysis Dataset.csv'" "'./Sentiment Analysis Dataset.csv'"
```

```
# Take a look at the dataframe
kable(head(raw_tweets))  %>% column_spec(4, width = "20em")
```

ItemID	Sentiment	SentimentSource	SentimentText
8	0	Sentiment140	Sunny Again Work Tomorrow :-| TV Tonight
9	1	Sentiment140	handed in my uniform today . i miss you already
18	1	Sentiment140	Feeling strangely fine. Now I'm gonna go listen to some Semisonic to celebrate
26	0	Sentiment140	BoRinG): whats wrong with him?? Please tell me........ :-/
35	0	Sentiment140	No Sat off...Need to work 6 days a week
89	0	Sentiment140	In case I feel emo in camp (feeling a wee bit of it alr)...am bringing in the Human Rights Watch World Report 2009..hope it'll work

```
# Convert Sentiment from num to factor and change levels
raw_tweets$Sentiment <- as.factor(raw_tweets$Sentiment)
levels(raw_tweets$Sentiment) <- c("Negative", "Positive")
raw_tweets$SentimentSource <- as.factor(raw_tweets$SentimentSource)
```

So we have 78,931 rows. Even though tweets are somewhat short, this is a lot of data. Tokenization would create too many features, to be handled efficiently, if we were to try to use this much data. Therefore, we should train and train on about 5% of these data–and validate on some of the rest later. We will make sure to maintain the proportionality along the way. Let's see what that is.

What proportion of "Sentiment" do we have in our corpus?

```
# Get the proportion of Sentiment in the corpus
prop.table(table(raw_tweets[, "Sentiment"]))
```

```
## Sentiment
##  Negative  Positive
## 0.4977005 0.5022995
```

Looks like almost 50/50. Nice. In this case a random sample would probably give us very similar proportions, we will use techniques to hard maintain this proportion i.e. just as if we had an unbalanced data set.

```
# Get the proportion of the SentimentSource
prop.table(table(raw_tweets[, "SentimentSource"]))
```

```
## SentimentSource
##       Kaggle Sentiment140
##  0.001026213  0.998973787
```

I'm not sure what this *SentimentSource* column is, but it looks like the vast majority is "Sentiment140". We'll ignore it for now.

Count Features

Let's add some features based on counts of how many hash-tags, web-links, and @refs are in each tweet.

```
# Count how many http links are in the tweet
raw_tweets$web_count <- str_count(raw_tweets$SentimentText,
                                  "http:/*[A-z+/+.+0-9]*")
# Count haw many hashtags are in the tweet
raw_tweets$hashtag_count <- str_count(raw_tweets$SentimentText,
                                      "#[A-z+0-9]*")
# Count how many @reply tags are in the tweet
raw_tweets$at_ref_count <- str_count(raw_tweets$SentimentText,
                                     "@[A-z+0-9]*")
# Count the number of characters in the tweet
raw_tweets$text_length <- nchar(raw_tweets$SentimentText)
```

```
# View the first few rows
kable(head(raw_tweets %>%
             select(-ItemID, -SentimentSource))) %>%
  column_spec(1, width = "6em") %>%
  column_spec(2, width = "12em")
```

Sentiment	SentimentText	web_count	hashtag_count	at_ref_count	text_length
Negative	Sunny Again Work Tomorrow :-| TV Tonight	0	0	0	54
Positive	handed in my uniform today . i miss you already	0	0	0	47
Positive	Feeling strangely fine. Now I'm gonna go listen to some Semisonic to celebrate	0	0	0	78
Negative	BoRinG): whats wrong with him?? Please tell me........ :-/	0	0	0	67
Negative	No Sat off...Need to work 6 days a week	0	0	0	39
Negative	In case I feel emo in camp (feeling a wee bit of it alr)...am bringing in the Human Rights Watch World Report 2009..hope it'll work	0	0	0	131

Some Manual Work

One thing to note: looking into the data it appears that there is a problem with the csv. There is a text_length greater than the maximum text length twitter allows.

```
# get the max character length in the corpus
max(raw_tweets$text_length)
```

```
## [1] 1045
```

Upon manual inspection we can see that several texts are getting crammed into the column of one–resulting in a very long string not properly parsed.

How many do we have that are over the 280 character limit?

```
# Count of the tweets that are over the character limit
count(raw_tweets[which(raw_tweets$text_length > 280),])$n
```

```
## [1] 4
```

Looking at these we see a few more examples like above, but also see a bunch or garbage text (i.e. special characters). We'll remove special characters later. This will take care of this by proxy. Also, we'll remove incomplete cases (after cleaning) in case we are left with only empty strings.

For now let's just remove all tweets that are over the limit. We have an abundance of data so it's ok to remove some noise. And check to make sure they are gone.

```
# Remove any tweets that are over 280 character counts
raw_tweets <- raw_tweets[-which(raw_tweets$text_length > 280),]
# Check that they have been removed
count(raw_tweets[which(raw_tweets$text_length > 280),])$n
```

```
## [1] 0
```

Also, I did notice that many of the problem tweets above came from the "Kaggle" source. Kaggle is a Data Science competition platform. It is a great resource for competition and learning. My theory is that this data was used and enriched during a Kaggle competition. It seems disproportionate that several of the problem tweets were from this source. Let's remove them all.

```
# Count of "Kaggle" records
count(raw_tweets[which(raw_tweets$SentimentSource == "Kaggle"),])$n
```

```
## [1] 80
```

```
# Remove the "Kaggle" treets
raw_tweets <- raw_tweets[-which(raw_tweets$SentimentSource == "Kaggle"),]
# Check that they have been removed
count(raw_tweets[which(raw_tweets$SentimentSource == "Kaggle"),])$n
```

```
## [1] 0
```

Visualize Distributions of Engineered Features

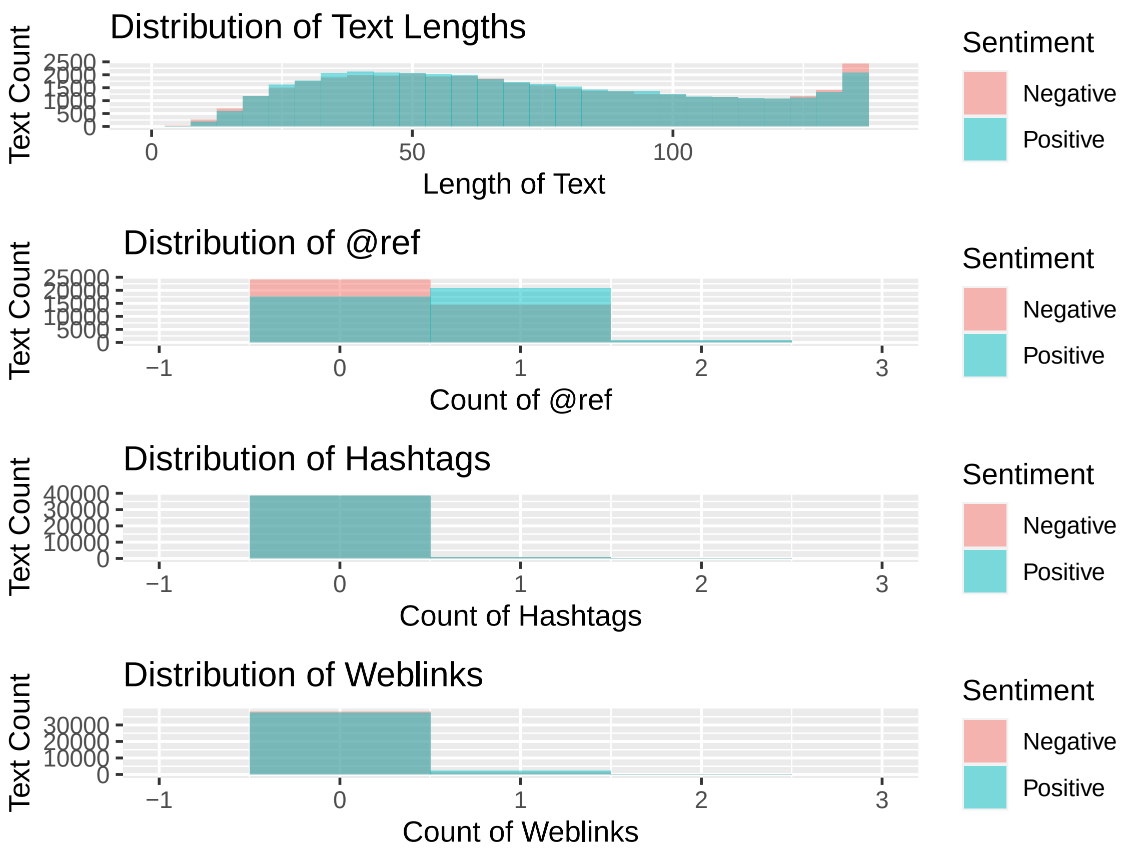
```
# Create 3 plots and display side-by-side
plot1 <- ggplot(raw_tweets,aes(x = text_length,
                               fill = Sentiment)) +
        geom_histogram(binwidth = 5,
                       position = "identity",
                       alpha = 0.5) +
        xlim(-1,140) +
        labs(y = "Text Count",
             x = "Length of Text",
             title = "Distribution of Text Lengths")

plot2 <- ggplot(raw_tweets,
                aes(x = at_ref_count,
                    fill = Sentiment)) +
        geom_histogram(binwidth = 1,
                       position = "identity",
                       alpha = 0.5) +
        xlim(-1,3) +
        labs(y = "Text Count", x = "Count of @ref",
             title = "Distribution of @ref")

plot3 <- ggplot(raw_tweets,
                aes(x = hashtag_count,
                    fill = Sentiment)) +
        geom_histogram(binwidth = 1,
                       position = "identity",
                       alpha = 0.5) +
        xlim(-1,3) +
        labs(y = "Text Count",
             x = "Count of Hashtags",
             title = "Distribution of Hashtags")

plot4 <- ggplot(raw_tweets,
                aes(x = web_count,
                    fill = Sentiment)) +
        geom_histogram(binwidth = 1,
                       position = "identity",
                       alpha = 0.5) +
        xlim(-1,3) +
        labs(y = "Text Count",
             x = "Count of Weblinks",
             title = "Distribution of Weblinks")

grid.arrange(plot1, plot2, plot3, plot4, nrow=4, ncol=1)
```



Doesn't look like any of the features we engineered suggest much predictive value. We'll have to rely on tokenizing the text to get our features–unless we can come up with other ideas. We can start with simple tokenozation (i.e. "Bag of Words") and also try some N-grams. Simple Bag of Words tokenization does not preserve the word order or association, but N-grams will cause our feature space to explode and is typically very sparse. This will require some dimensionality reduction–which will certainly add complexity and is a "black-box"" method. i.e we lose the ability to inspect or explain the model.

Let's start creating our test/train set and start modeling.

Stratified Sampling

Let's create a data partition. First we'll take 10% of the 78,847 tweets to build our model. We'll further split this into test and train data sets. We'll preserve the indexes so we can further partition later if necessary.

```
# Set seed for randomizer
set.seed(42)
# Retrieve indexes for partitioning
partition_1_indexes <- createDataPartition(raw_tweets$Sentiment,
                                           times = 1, p = 0.10, list = FALSE)
# Create dataframe
train_validate <- raw_tweets[partition_1_indexes, c(2, 4, 7)]
# Reset seed
set.seed(42)
# Retrieve indexes for train and test partition
train_indexes <- createDataPartition(train_validate$Sentiment,
                                     times = 1, p = 0.60, list = FALSE)
# Use the indexes to create the train and test dataframes
train <- train_validate[train_indexes, ]
test <- train_validate[-train_indexes, ]
# Return the number of records in the training set
nrow(train)
```

```
## [1] 4732
```

So, now we have 4,732 tweets. Check proportions just to be safe.

```
# Check proportion is same as original table
prop.table(table(train$Sentiment))
```

```
##
##  Negative  Positive
## 0.4976754 0.5023246
```

And we have almost exactly the same proportions as our original, much larger, data set.

Tokenization

Let's now tokenize our text data. This is the first step in turning raw text into features. We want the individual words to become features. We'll cleanup some things, engineer some features, and maybe create some combinations of words a little later.

There are lots of decisions to be made when doing this sort of text analysis. Do we want our features to contain punctuation, hyphenated words, etc.? Typically in text analysis, special characters, punctuation, and numbers are removed because they don't tend to contain much information to retrieve. However, since this is Twitter data, our corpus does contain some emoticons that are represented as special characters (ex: ":-)", ":-/"). If we remove them we will lose the textual representations of emotion. But, in looking closely at the data, these emoticons are surprisingly not very prevalent. So let's just remove them.

```
# Convert SentimentText column to tokens
train_tokens <- tokens(train$SentimentText,
                       what = "word",
                       remove_numbers = TRUE,
                       remove_punct = TRUE,
                       remove_twitter = TRUE,
                       remove_symbols = TRUE,
                       remove_hyphens = TRUE)
```

```
## Warning: remove_twitter, remove_hyphens arguments are not used.
```

Let's look at a few to illustrate what we did.

```
# Inspect tweets tokens
train_tokens[[29]]
```

```
##  [1] "quot"    "she"     "has"     "a"       "great"   "smile"   "quot"
##  [8] "Thank"   "you"     "I"       "GOT"     "that"    "comment" "from"
## [15] "DEMI"    "quot"    "S"       "single"  "la"      "la"      "land"
## [22] "music"   "video"   "haha"    "she"     "really"  "has"     "a"
## [29] "good"    "SMILE"
```

These are the tokens, from the 29th record, of the training data set. i.e. the tweet below.

```
train[29,2]
```

```
## # A tibble: 1 x 1
##   SentimentText
##   <chr>
## 1 &quot; she has a great smile   &quot; Thank you! I GOT that comment from DEMI~
```

Also this one has some Uppercase, special characters, and stop words:

```
train_tokens[[30]]
```

```
##  [1] "Wasn't"  "allowed" "on"      "the"     "drums"   "stuck"   "on"
##  [8] "the"     "bass"    "instead" "I'm"     "no"      "Jack"    "Bruce"
## [15] "but"     "still"   "blew"    "my"      "other"   "band"    "members"
## [22] "away"    "Natch"
```

Let's change all Uppercase to lower to reduce the possible combinations.

```
# Convert to lower-case
train_tokens <- tokens_tolower(train_tokens)
# Check same tokens as before
train_tokens[[29]]
```

```
##  [1] "quot"    "she"     "has"     "a"       "great"   "smile"   "quot"
##  [8] "thank"   "you"     "i"       "got"     "that"    "comment" "from"
## [15] "demi"    "quot"    "s"       "single"  "la"      "la"      "land"
## [22] "music"   "video"   "haha"    "she"     "really"  "has"     "a"
## [29] "good"    "smile"
```

Remove Stopwords

Let's remove stopwords using the quanteda packages built in *stopwords()* function and look at record 26 again.

```
# Remove stopwords
train_tokens <- tokens_select(train_tokens,
                              stopwords(),
                                selection = "remove")
train_tokens[[29]]
```

```
##  [1] "quot"    "great"   "smile"   "quot"    "thank"   "got"     "comment"
##  [8] "demi"    "quot"    "s"       "single"  "la"      "la"      "land"
## [15] "music"   "video"   "haha"    "really"  "good"    "smile"
```

And record 29 again:

```
train_tokens[[30]]
```

```
##  [1] "allowed" "drums"   "stuck"   "bass"    "instead" "jack"    "bruce"
##  [8] "still"   "blew"    "band"    "members" "away"    "natch"
```

Stemming

Next, we need to stem the tokens. Stemming is a method of getting to the word root. This way, we won't have multiple versions of the same root word. We can illustrate below.

```
# Stem tokens
train_tokens <- tokens_wordstem(train_tokens,
                                  language = "english")
train_tokens[[29]]
```

```
##  [1] "quot"    "great"   "smile"   "quot"    "thank"   "got"     "comment"
##  [8] "demi"    "quot"    "s"       "singl"   "la"      "la"      "land"
## [15] "music"   "video"   "haha"    "realli"  "good"    "smile"
```

```
train_tokens[[30]]
```

```
##  [1] "allow"   "drum"    "stuck"   "bass"    "instead" "jack"    "bruce"
##  [8] "still"   "blew"    "band"    "member"  "away"    "natch"
```

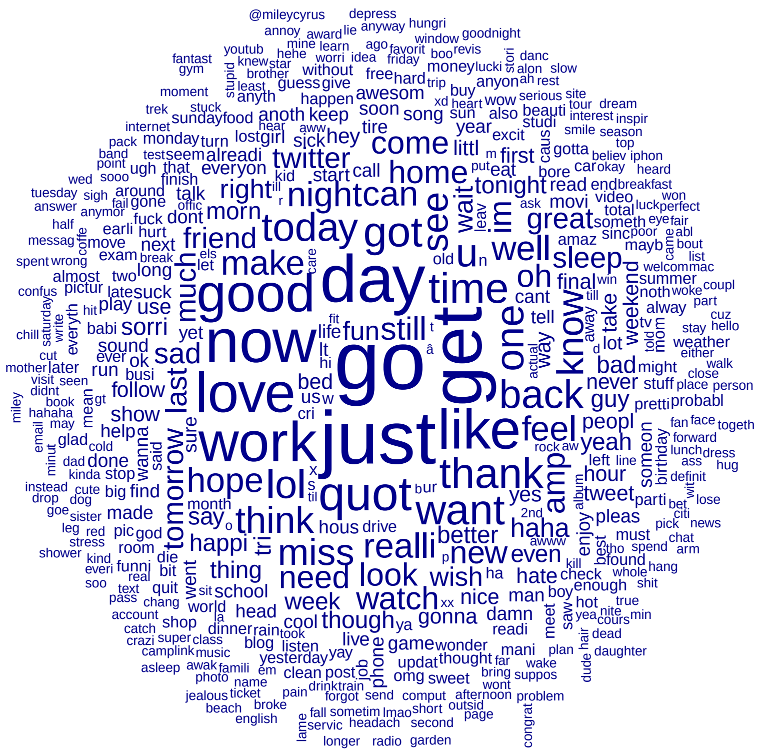
You can see that "allowed" becomes "allow", and "drums" becomes "drum", etc.

Create a Document-Feature Matrix

```
# Create a DFM
train_dfm <- dfm(train_tokens,
                 tolower = FALSE)
```

Let's take a quick look at a wordcloud of what is in the dfm.

```
# Create wordcloud
train_dfm %>% textplot_wordcloud()
```



```r
# Convert to matrix
train_dfm <- as.matrix(train_dfm)
```

We now have a matrix–the length of our original data frame–now with 9581 features in the term. That is a lot of features. We are definitely suffering from the "curse of dimensionality". We'll need to do some feature reduction at some point.

```r
# Check dimensions of the DFM
dim(train_dfm)
```

```
## [1] 4732 9581
```

Let's look at the first 6 documents (as rows) and the first 20 features of the term (as columns).

```r
# View part of the matrix
kable(head(train_dfm[1:6, 1:20]))
```

	case	feel	emo	camp	wee	bit	alr	bring	human	right	watch	world	report	hope	it'll	wor
text1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	
text2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
text3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
text4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
text5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
text6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Now we have a nice DFM. The columns are the features, and the column-space is the term. The rows are the documents and the row-space are the corpus.

```r
# Bind the DFM, Sentiment together as a dataframe
train_df <- cbind("Sentiment" = as.factor(train$Sentiment),
                  as.data.frame(train_dfm))
kable(train_df[5:15, 35:50])
```

	eurgh	silli	cold	tire	need	stuff	back	gov’t	motor	just	peed	littl	http://bit.ly/aldxf	via
text5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
text6	1	3	1	1	1	1	1	0	0	0	0	0	0	0
text7	0	0	0	0	0	0	0	1	1	1	1	1	1	1
text8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
text9	0	0	0	0	0	0	0	0	0	1	0	0	0	0
text10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
text11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
text12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
text13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
text14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
text15	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Unfortunately, R cannot handle some of these tokens as columns in a data frame. The names cannot begin with an integer or a special character for example. We can use the `make.names()` function, to insure we don’t have any invalid names.

```r
# Alter any names that don't work as columns
names(train_df) <- make.names(names(train_df),
                              unique = TRUE)
```

Setting up for K-fold Cross Validation

We will set up a control plan for 30 models. We should be able to use this plan for all our subsequent modeling.

```r
# Set seed
set.seed(42)
# Define indexes for the training control
cv_folds <- createMultiFolds(train$Sentiment,
                             k = 10, times = 3)
# Build training control object
cv_cntrl <- trainControl(method = "repeatedcv",
                         number = 10,
                         repeats = 3,
                         index = cv_folds)
```

Train the First Model

Let’s train the first model to see what kind of accuracy we have. Let’s use a single decision tree algorithm. This algorithm will, however, create 30 * 7 or 210 models. [2]

```r
# Train a decision tree model using
# the training control we setup
 rpart1 <- train(Sentiment ~ .,
                 data = train_df,
                 method = "rpart",
                 trControl = cv_cntrl,
                 tuneLength = 7)
```

```r
# Inspect the model output
rpart1
```

[2]Note: I am loading a pre-processed model. Training the model takes a long time. If you wishd to run the model yourself, you will have to modify the code below. You’ll need to remove the `eval=FALSE` on the next to sections and add `eval=FALSE` to the `load("../../rpart1.rds")` section.

```
## CART
##
## 4732 samples
## 9157 predictors
##    2 classes: 'Negative', 'Positive'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 4258, 4258, 4259, 4259, 4259, 4260, ...
## Resampling results across tuning parameters:
##
##   cp            Accuracy   Kappa
##   0.0004246285  0.6254591  0.24905423
##   0.0012738854  0.6288400  0.25581092
##   0.0014437367  0.6289804  0.25608746
##   0.0076433121  0.5839126  0.16641297
##   0.0162774239  0.5546603  0.10857242
##   0.0184713376  0.5383143  0.07756101
##   0.0441613588  0.5101443  0.01978729
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.001443737.
```

Outputting the model results we see that we have an accuracy 62.9% accuracy already. That isn't bad. Really we want to get to about 90% if we can. This is already better than a coin flip and we haven't even begun. Let's take some steps to improve things.

To be continued…