Data Science Capstone: Quantifying Yelp User Reviews with Sentiment Analysis

Dax Gerts

Saturday, November 22, 2015

Intro: One of the most rapidly developing subsets of interest in data science is Natural Language Processing and its various uses. In the case of the Yelp Reviews Dataset, these tools are highly useful in attempting to salvage analytic data from masses of collected text reviews.

While there are a variety of goals and approaches that could be taken in analyzing this data, for the purposes of this report, the point of interest was narrowed to a single predictive goal, described as follows:

Predictive Task: Can the numeric component of user reviews (stars given) by predicted from the contents of the text provided? Or by evaluating the text in conjunction with other values?

If this information could be predicted successfully, it would be a useful tool not only for verifying and weighting existing numerical scores, but it might also be a means to give more "honest" reviews in which the words used really do speak for themselves.

Methods: The methodology that was adopted for this project was to apply the Natural Language Processing technique of "sentiment analysis" to the user reviews in order to score chunks of text based on their distinctly positive and negative content. The manner in which this was accomplished is described below.

1) Data preparation: A number of libraries were employed in the course of this project, most notably were *caret* and *randomForest* which were used for model building and testing. Additionaly, outside dictionaries were used to score sentiment, both positive and negative. These were provided courtesy of the University of Illinois at Chicago.

```r
library(plyr) # Data preparation
library(stringr) # String manipulation
library(e1071) # Support Vector Machines
library(AppliedPredictiveModeling) # Predictive tools
library(caret) # Machine learning tools
library(randomForest)
# Set seed
set.seed(1890)
```

The Yelp data was prepared at an earlier point and loaded from the working directory via .RData files. The sentiment files were also scanned and saved to corresponding data frames, "pos"itive and "neg"ative.

```r
load("dataReview.RData")
load("dataUser.RData")
# Download pos/neg files from http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar
# Make sure files are in working directory
pos <- scan("positive-words.txt", what = "character", comment.char = ";")
neg <- scan("negative-words.txt", what = "character", comment.char = ";")
```

2) Preprocessing: Due to the size of the data being handled, memory efficiency was a primary concern throughout the course of this project. As such, certain methods were constrained or used only with subsets of the overall data. For preprocessing, this meant that the User and Review data sets were merged on "user_id" and then pruned to remove unnecessary attributes (ids, friends, types). The data was split into training and test sets before all of the intermediary data frames were released from memory.

The attributes that were selected for model testing with were the following: review text, review votes (funny, useful, cool), stars, user votes (funny, useful, cool), and number of reviews given by user. All were deemed as potentially pertinent to the evaluation of the score given.

```r
# Remove excess columns
reviews <- merge(dataReview,dataUser,by ="user_id")
reviews <- reviews[,c(2,4,6,10,11)]

# Create partition space (80/20 split)
part <- createDataPartition(reviews[,2], p=0.8, list=FALSE)
training <- reviews[part,]
test <- reviews[-part,]

# Free up memory in workspace
rm(dataReview,dataUser,reviews,part)
```

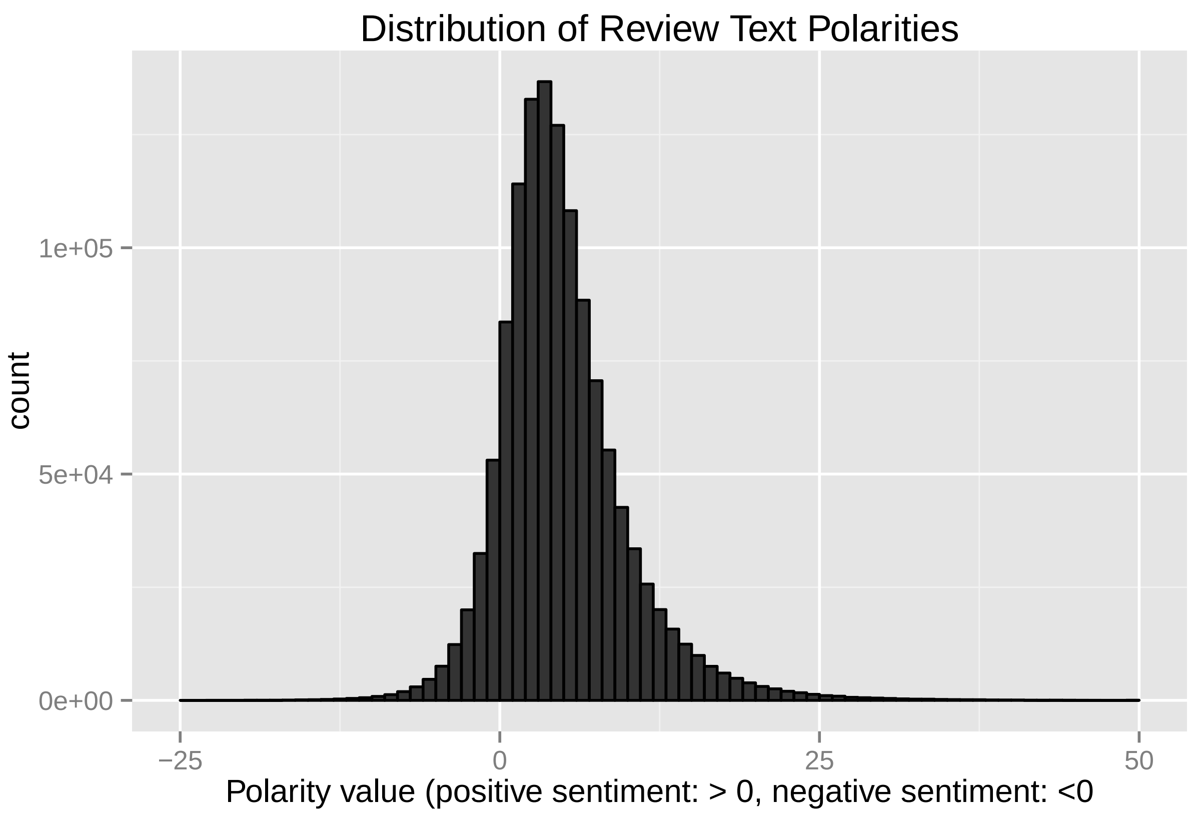
3) Prepare text analytics: The most important part of this procedure was the mechanic for scoring the sentiment of each user review, a process that was accomplished by creating a "sentiment" function. This function employed regular expressions to reduce each text string to all lower case, alphabetic characters without any punction. Once this was accomplished, each word was checked agains the positive and negative sentiment lists and scored according to the aggregate frequencies of each, **score = positive.frequency - negative.frequency**. This function was called on both the test and training data sets.

```r
# Sentiment function
sentiment = function(sent, pos, neg) {

# Calculate sentiment scores
scores = laply(sent, function(sent, pos.words, neg.words) {

# Clean sentences with regex
sent = gsub('[[:punct:]]', '', sent)
sent = gsub('[[:cntrl:]]', '', sent)
sent = gsub('\\d+', '', sent)
sent = tolower(sent)

# Atomize to words
word.list = str_split(sent, '\\s+')
words = unlist(word.list)

# Compare words to sentiment lists
pos.matches = !is.na(match(words, pos.words))
neg.matches = !is.na(match(words, neg.words))
score = sum(pos.matches) - sum(neg.matches)

return(score)
}, pos, neg)

scores.df = data.frame(score=scores, text=sent)
return(scores.df)
}

# Calculate sentiment scores on each chunk of text
training$text <- sentiment(training$text,pos,neg)
test$text <- sentiment(test$text,pos,neg)
```

```r
saveRDS(training,"trainingBackup.rds")
```

4) Exploration & Visualization Before the model-building process was initiated, the value to be predicted, "# of stars given", and the particular predictor of interest, sentiment score, were each plotted as frequency histograms in order to visualize some of the challenges that the distributions might present (Due to space constraints only one plot will be shown here.

Both plots showed are significant skew to the right, with a majority of "stars given" being 4 or 5 and almost all sentiment scores falling above 0 (positive). This would inidicating that any predictive model is likely to evaluate towards a higher rating than what is reflected by reality.



5) Model Training and Testing Numerous iterations of model testing were performed which, for the sake of brevity, cannot be covered exhaustively in this report. The primary observation however was that despite having roughly 1.2 million tuples to train a model from, there was little observable increase in model performance after the first several thousand observations, and in some cases model performance began to suffer due to severe overfitting, which combined with a quadratically increasing runtime, strongly dissincentivised scaling up the size of the usable training set.

With this in mind, the model shown here was built from a minor subset of all observations, but was built using as many attributes as might be relevant: all 3 vote metrics for each review, aggregate vote metrics for each user, sentiment score, and total number of reviews.

```r
#Subset data to amount that will finish execution in reasonable time
sub <- head(training,1000)
#Build model on selected attributes using random forests
model <- train(stars ~ text$score + votes.x$funny + votes.x$useful + votes.x$cool + votes.y$funny + vot
#Predict values based on model from training set
trainPred <- predict(model,sub)
#Clean values to resolve to integers between 1 and 5
predClean <- function(pred) {
for(i in 1:length(pred)) {
  if(pred[i] > 5) { pred[i] <- 5 }
  if(pred[i] < 1) { pred[i] <- 1 }
  pred[i] <- round(pred[i])
}
pred}
trainPred <- predClean(trainPred)
```

```r
#Evaluate model on training set
confusionMatrix(trainPred,sub$stars)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   5.210000e-01   2.828332e-01   4.895145e-01   5.523616e-01   3.640000e-01
## AccuracyPValue  McnemarPValue
##   3.632701e-24            NaN
```

Results: To evaluate the results of the procedure outlined above, the model was applied to the test data set. To the following results.

```r
#Predict values based on model from test set
testPred <- predict(model, test)
#Clean values to resolve to integers between 1 and 5
testPred <- predClean(testPred)
```

```r
#Evaluate model on test set
confusionMatrix(testPred,test$stars)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.30524357     0.06757159     0.30363267     0.30685807     0.36929944
## AccuracyPValue  McnemarPValue
##     1.00000000     0.00000000
```

Perhaps the most immediately signficant observation is the discrepancy between the accuracy of the model as applied to the training data versus the model when applied to the test data, a gap of almost 20%. This was most likely attributed to overfitting and poor technique attribute selection, but without much more thorough examination it would be difficult to precisely determine the cause.

To briefly address the predictive task described in the introduction: **the results indicate that analysis of text sentiment has some degreee of predictive use, however, building useful models with Natural Language Processing techniques requires significant work and study, and as such is an ongoing process.**

Discussion: This report is truly only an initial analysis of how Natural Language Processing techniques might be applied to various types of user data. As with any type of predictive task, there is never truly a "correct answer" only "better" answers. In this case, I do not believe the model that was created to have significant predictive value on its own, however, I believe that it serves an adequate representation of the fact that analysis of text data can be useful in making up for inadequacies in overly sparse numerical data.

The accuracy of approximatley 50% that was found in the model created from the test set would indicate that text sentiment does have statistcal validity, scoring roughly 1.5 times more accurately than random chance, and I believe that when applied to an existing model that sentiment would provide a strong tuning measure to any predictive or descriptive task.

In future efforts, it would be interesting to extract more numerical data from the text, such as length, capitalized versus non-capitalized characters, distinctive punctuation, and so on. I believe that if applied as factors/categorical data, such an endeavor might provide the basis for building useful association rules, perhaps with the R "apriori" library.