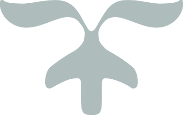


Classification of Various Sentiments

A Report for Potential Business Opportunities & Applications



**OPIM 5671 Team #1**

Ravi Kolla

Tathagata Basu

Venkatasai Varada

Vikas Singh

Marc Nazareth

Contents

[Executive Summary 2](#_Toc451912111)

[Statement of the Problem 2](#_Toc451912112)

[Background 3](#_Toc451912113)

[Methodology 3](#_Toc451912114)

[Results 4](#_Toc451912115)

[Conclusion and Recommendations 6](#_Toc451912116)

[Appendix 7](#_Toc451912117)

[References 7](#_Toc451912118)

# Executive Summary

-Might be best to summarize rest of paper at the end.

# Statement of the Problem

Social media has become an increasingly pervasive presence in the day to day life of individuals, groups, and companies. A quick browse on any retailers site will show thousands upon thousands of reviews sorted and averaged into a concise number score. Scores of blogs exist that review all sorts of content by professional critiques and passionate individuals alike. Aggregator sites that compile a vast collections of reviews serve as prominent and powerful indicators of a global audience’s reception. But what if we wanted to extract more value from this level of customer feedback? With such a vast amount of textual data I the world there has to be more than just an average numeric score to provide us with insights. Could there be value in searching for any type of pattern in the textual responses of consumer reactions?

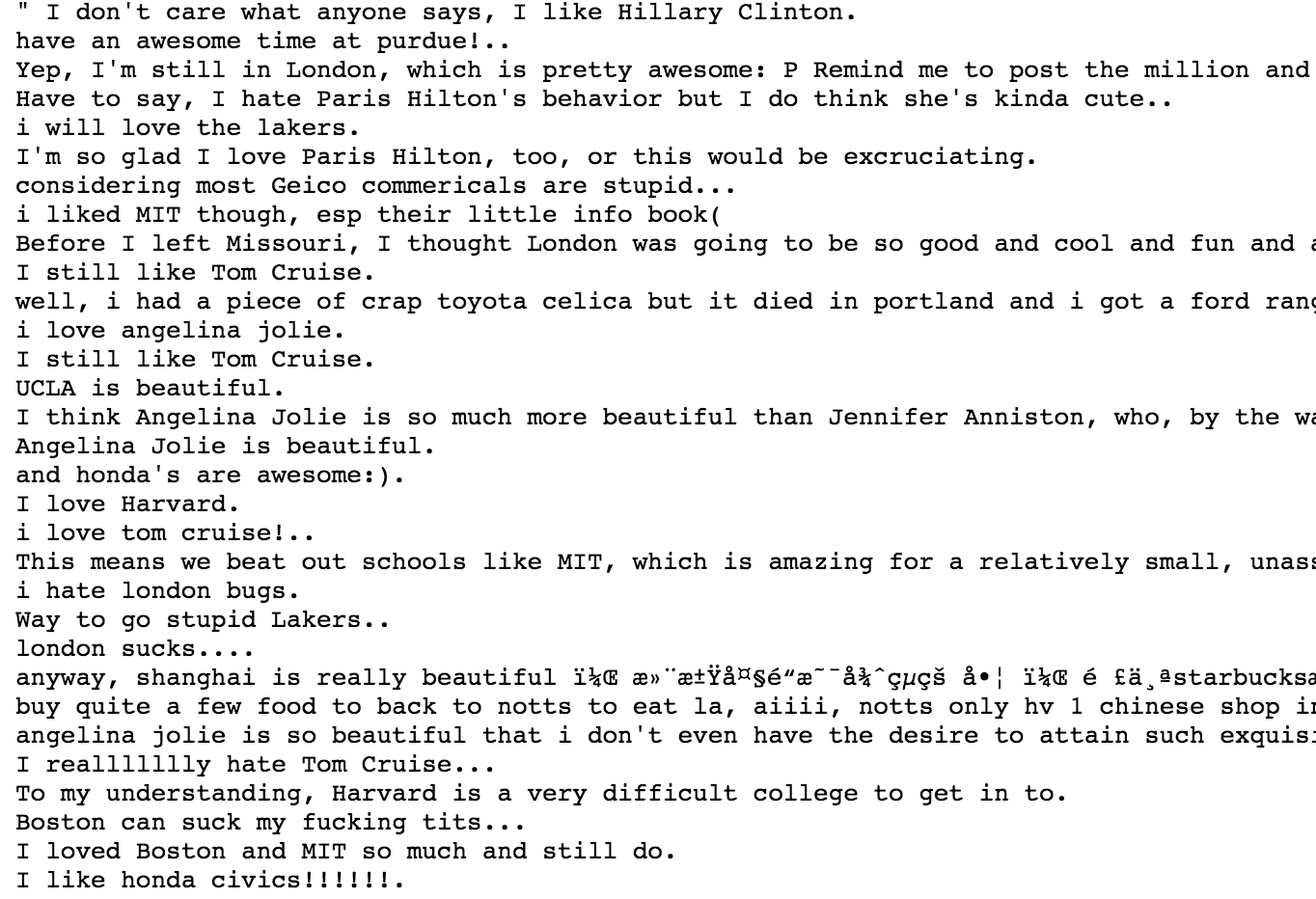
Background

The goal of this project is to extract business value from the data set in a manner that can be both expanded and applied to future data sets. By analyzing the customer reviews of a particular product or company, we can suggest methods for a company to improve its products and related customer satisfaction. We can achieve this objective by leveraging what we have learned in the data mining class. We would like to approach the problem by using various text mining techniques and constructing models. We would utilize SEMMA analysis (Sample, Explore, Modify, Model, Assess) as a roadmap in approaching the dataset.

The data was obtained from Kaggle, a competition website that is kind enough to share its data sets for academic use. This particular data set contains 7086 sentences, each one coded with a 1 or 0 meaning positive or negative sentiment, respectively. A link to the actual data set can be found in the references section of this report.

# Methodology

We set out to try and classify the various sentiments from a data series containing a pertinent sentence extracted from a document obtained from various forms of social media. By classifying the sentiment of each sentence we believe we could classify each one into either “positive” or “negative” reaction. Text parsing and filtering tools would allow us to find the insights about the most dominant words and see how these sentiments relate. Analyzing the data in R and Python in addition to SAS might provide us with alternate insights and greater potential future applications. After completing analysis in each program, a model comparison would be utilized to asses and contrast the results. For the unsupervised portion of the analysis, tools like clustering in SAS will need to be used where as tools like regression will be useful for the supervised portion. A sample of the raw data can be seen below.



A brief glance at the raw data above tells a bit about what kind of data we are dealing with and lead us to a few preliminary insights. As previously discussed, the data contains sentences taken from blogs, comments, and more, all describing various sentiments towards particular current events, products, places, people, and more. Some sentences are quite obvious in their admiration or distaste about a particular subject while some might be a little more ambivalent and therefore tougher to categorize.

SAS

R



The above images show preprocessing in R using word clouds. The first step was to narrow down the data to make it more useful for us. To do this, we removed words that only appeared in .5% or less of documents since we did not think these would contribute to building a good sentiment model.

Python

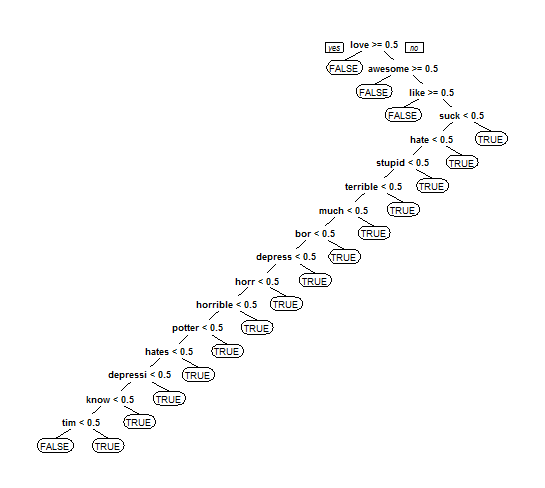
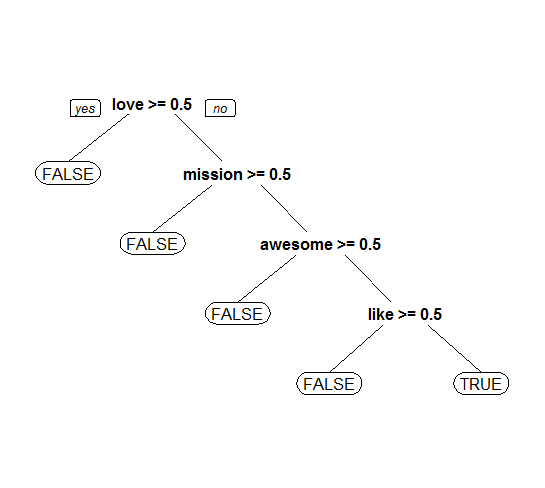
Using the “pandas library” available in Python, we loaded the train and test data in the Python dataframes. The training data is classified in a format that gives both text and sentiment, where as the test data is classified into just the text format. Initial clean up steps included using the stemmed English words, removing the prefixes and suffixes in words by utilizing the “PorterStemmer” libraries available in the NLTK (Natural Language Toolkit) package, and converting the collection of text documents to a matrix of token counts used to create a bag-of-words set of features by utilizing the “CountVectorizer” libraries. This last step makes analysis easier by breaking down the text into smaller pieces, utilizing a process called tokenization. Finally, punctuations, lowercase, and stem words were removed by using the “Regex” libraries available in Python.

The Term Frequencies/Inverse Document Frequency (tf-idf), a numerical statistic intended to reflect the importance of a word in relation to a document in a collection or corpus, is often used as a weighting factor in information retrieval and text mining. Put more simply, the tf-idf value increases with the frequency of a given word in a document but is offset by the word’s frequency in the corpus, which adjusts for some words appearing more frequently in general. This exploratory approach was used to find the frequent words.

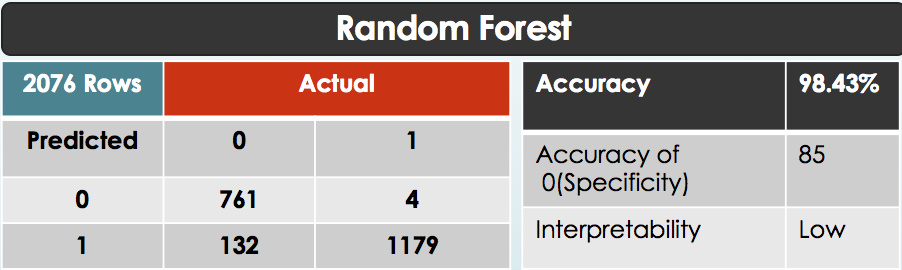
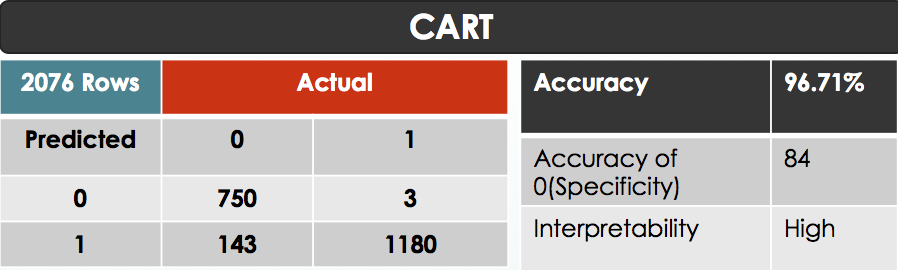
# Before processing the data, a quick look at a weighted cluster of the source material produced the images above.

# Results

# R

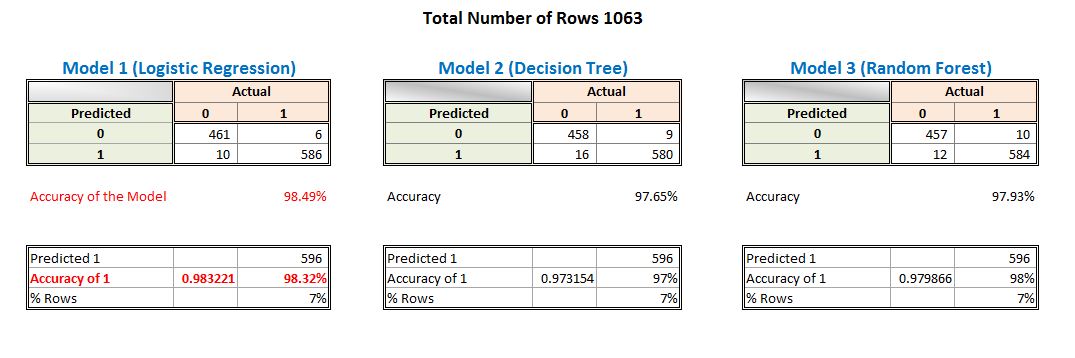


Using R to try and alternatively predict negative sentiment indicated that words like “love” “awesome” and “hate” we most likely to be indicative of sentiment.

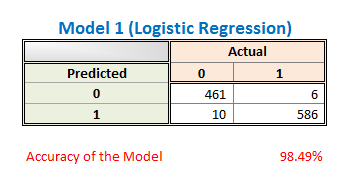


The accuracy of the random forest (RF) model is better in R. The value in the CART model tells us specifically what words help to determine sentiment as opposed to the RF model.

# Python



A comparison of the model accuracies indicates the Logistic Regression model as highest, with an accuracy of 98.49% while the accuracy of the RF model is 97.93%. The Logistic Regression model also had the lowest type 2 error as seen below (model predicted 1 when it was actually 0)



|  |  |  |
| --- | --- | --- |
| Accuracy | (True Positive + True Negative)/Total | 0.98494826 |
| Misclassification Rate | (False Positive + False Negative)/Total | 0.01505174 |
| Specificity | (True Negative)/Actual NO | 0.98715203 |
| Sensitivity | (True Positive)/Actual Yes | 0.98322148 |
| Precision | (True Positive)/Predicted Yes | 0.98986486 |

A Decision Tree model was also considered (97.65% accuracy), but is prone to over-fitting so we have alternatively considered the RF classifier to predict the accuracy of the model.





# Conclusion and Recommendations

The business use of sentiment analysis is analyzing negative sentiments for the purpose of both individual and future improvement. By focusing on what went wrong with negative sentiment expressing customers, a business can benefit by restoring that customers experience, potentially retaining their business in the future, and not repeating the mistake in the future. A mistake in the model where a negative review is classified as a positive review would fail to achieve any of these three benefits and could therefore result in a loss of business. Regarding R, although both models had similar metrics, interpretability is extremely important. The CART model clearly tells you which words help to determine the sentiment while the RF model does not.

# Appendix- Codes

# Python

Py#Sentimental Analysis using Python

# importing and loading the test and train data in python dataframes using the pandas api

import pandas as pd

test\_data\_file\_name = "c:/datamining/testdata.txt"

train\_data\_file\_name = "c:/datamining/training.txt"

test\_data\_df = pd.read\_csv(test\_data\_file\_name, header=None, delimiter="\t", quoting=3)

test\_data\_df.columns = ["Text"]

train\_data\_df = pd.read\_csv(train\_data\_file\_name, header=None, delimiter="\t", quoting=3)

train\_data\_df.columns = ["Sentiment","Text"]

# stemming the english words with the help of PorterStemmer api available in NLTK package, removing the prefixes-suffixes in words

# CountVectorizer api converts a collection of text documents to a matrix of token counts and is used to create a bag-of-words set of features

# Removing punctuations, lowercase, remove stop words, and stem words

import re, nltk

from sklearn.feature\_extraction.text import CountVectorizer

from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

def stem\_tokens(tokens, stemmer):

stemmed = []

for item in tokens:

stemmed.append(stemmer.stem(item))

return stemmed

# tokenizing the words

def tokenize(text):

# remove non letters

text = re.sub("[^a-zA-Z]", " ", text)

# tokenize

tokens = nltk.word\_tokenize(text)

# stem

stems = stem\_tokens(tokens, stemmer)

return stems

# removes non-letters and performs the stemming, together with lowercasing and removing english stop-words, number of terms in our final vectors (i.e. 85)

vectorizer = CountVectorizer(

analyzer = 'word',

tokenizer = tokenize,

lowercase = True,

stop\_words = 'english',

max\_features = 150

)

# fitting the model and learning the vocabulary, transforming our corpus data into feature vectors

corpus\_data\_features = vectorizer.fit\_transform(

train\_data\_df.Text.tolist() + test\_data\_df.Text.tolist())

# converting the result to an array

corpus\_data\_features\_nd = corpus\_data\_features.toarray()

vocab = vectorizer.get\_feature\_names()

# Sum up the counts of each vocabulary word

import numpy as np

dist = np.sum(corpus\_data\_features\_nd, axis=0)

# For each, print the vocabulary word and the number of times it appears in the data set(test+train data),

#total 85 important words considered

for tag, count in zip(vocab, dist):

print count, tag

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

# the corpus\_data\_features\_nd contains all of our

# original train and test data, so we need to exclude

# the unlabeled test entries

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

corpus\_data\_features\_nd[0:len(train\_data\_df)],

train\_data\_df.Sentiment,

train\_size=0.85,

random\_state=1234)

# Using Logistic Regression technique and training our classifier

from sklearn.linear\_model import LogisticRegression

log\_model = LogisticRegression()

log\_model = log\_model.fit(X=X\_train, y=y\_train)

#classifier to label our evaluation set

y\_pred = log\_model.predict(X\_test)

# calculating confusion matrix on a classification model

from sklearn.metrics import confusion\_matrix

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*confusion matrix report\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print(confusion\_matrix(y\_test, y\_pred))

# re-training our model with all the training data and use it for sentiment classification with the original (unlabeled) test set

# train classifier

log\_model = LogisticRegression()

log\_model = log\_model.fit(X=corpus\_data\_features\_nd[0:len(train\_data\_df)], y=train\_data\_df.Sentiment)

# get predictions

test\_pred = log\_model.predict(corpus\_data\_features\_nd[len(train\_data\_df):])

# running the model on 15 records

import random

spl = random.sample(xrange(len(test\_pred)), 15)

# print text and labels

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Model Prediction on Sample\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

for text, sentiment in zip(test\_data\_df.Text[spl], test\_pred[spl]):

print sentiment, text

# R

#“If you don’t understand their emotions, you don’t understand your customers”

sent<-read.csv("sentiment.csv", stringsAsFactors = FALSE)

str(sent)

wc<-read.csv("sentiment.csv")

write.csv(wc,"beforepreprocess.txt")

summary(sent)

#We are interested in detecting negative sentiments.

#Pre- Processing

sent$negative <- as.factor(sent$Sentiment==0)

table(sent$negative)

table(sent$Sentiment)

library(tm)

library(SnowballC)

corpus<-Corpus(VectorSource(sent$Text))

corpus

corpus[[1]]

corpus <- tm\_map(corpus,tolower)

corpus<-tm\_map(corpus,removePunctuation)

corpus<-tm\_map(corpus,removeWords,c(stopwords("english")))

stopwords("english")

corpus<-tm\_map(corpus,stemDocument)

corpus<-tm\_map(corpus,PlainTextDocument)

typeof(corpus)

writeLines(as.character(corpus), con="mycorpus.txt")

#Bag of Words

freq<-DocumentTermMatrix(corpus)

typeof(freq)

#total terms is 2156

#popular terms

#more terms means more independent variables.

freq20<-findFreqTerms(freq,lowfreq = 20)

freq20

#total terms is 2156,words that appear atleast 20 times 233 => there are words that appear very often. remove them to make model simple.

#This is called sparsity=> various terms that appear in very less documents.

sparse<-removeSparseTerms(freq,0.995)

#keep only terms that appear in atleast 0.5% of the terms.

#0.5% of 2156 = 10 docs. trade off choose balance.

str(sparse)

#terms = 224.

#224/2156 = 10.3. we could retrieve only 10% of the documents.

#converting in to a dataframe. with all the terms as variables.

sentsparse<-as.data.frame(as.matrix(sparse))

str(sentsparse)

colnames(sentsparse)= make.names(colnames(sentsparse))

sentsparse$negative<-sent$negative

str(sentsparse)

typeof(sentsparse)

sentsparse$loved<-NULL

sentsparse$sucks<-NULL

sentsparse$sucked<-NULL

sentsparse$movie<-NULL

sentsparse$awesom<-NULL

sentsparse$vinci<-NULL

sentsparse$dudeee<-NULL

sentsparse$mission<-NULL

sentsparse$imposs<-NULL

sentsparse$impossible<-NULL

sentsparse$harry<-NULL

sentsparse$movi<-NULL

sentsparse$mountain<-NULL

sentsparse$man<-NULL

sentsparse$cock<-NULL

sentsparse$brokeback<-NULL

sentsparse$much<-NULL

sentsparse$much<-NULL

sentsparse$hated<-NULL

sentsparse$hates<-NULL

sentsparse$heard<-NULL

sentsparse$around<-NULL

sentsparse$star<-NULL

sentsparse$quiz<-NULL

sentsparse$potter<-NULL

sentsparse$know<-NULL

sentsparse$soo<-NULL

sentsparse$awards<-NULL

library(caTools)

#Modelling

set.seed(1234)

spl<-sample.split(sentsparse$negative,SplitRatio = 0.7)

trainsparse<-subset(sentsparse,spl==TRUE)

testsparse<-subset(sentsparse,spl==FALSE)

table(trainsparse$negative)

table(sentsparse$negative)

table(testsparse$negative)

accoftraining<-2925/nrow(sentsparse)

cat("abc :",accoftraining)

2082/nrow(trainsparse)

893/nrow(testsparse)

#Predicting

library(rpart)

library(rpart.plot)

sentCART<-rpart(negative ~.,data=trainsparse, method="class")

#printcp(sentCART)

#newsent<- prp(sentCART,snip=TRUE)$obj # interactively trim the tree

#prp(new.tree.1) # display the new tree

prp(sentCART)

predictCART<- predict(sentCART,newdata = testsparse, type="class")

table(testsparse$negative,predictCART)

(1148+860)/nrow(testsparse) #96.7

table(testsparse$negative)

1183/nrow(testsparse) #57

install.packages("tree")

library(randomForest)

library(tree)

set.seed(1234)

sentRF<-randomForest(negative ~., data = trainsparse)

#tr <- tree(negative ~ ., data=trainsparse)

predictRF<-predict(sentRF, newdata=testsparse)

#cforest(negative ~ ., data=trainsparse, controls=cforest\_control(mtry=2, mincriterion=0))

#getTree(sentRF, 1, labelVar=TRUE)

table(testsparse$negative,predictRF)

(1173+876)/nrow(testsparse) # -->98.4

install.packages("party")

library("party")

x <- ctree(negative ~ ., data=trainsparse)

plot(x, type="simple")

# References

Data Source: <https://inclass.kaggle.com/c/si650winter11/data>

<http://scikit-learn.org/stable/supervised_learning.html#supervised-learning>

<http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression>

<https://en.wikipedia.org/wiki/Random_forest>

<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>