DS 6030 Final - Sentiment Analysis

Ben Wilson | Jason Zhang | Shreyas Adiyodi

Due: Wed Dec 16 | 12:00pm

Contents

---

1. Sentiment Analysis

1.1 Required R Packages

We will be using the R packages of: * lexicon for determining common sentiment * spacyr for leveraging its natural language process (NLP) library * SentimentAnalysis for determining the sentiment of words and phrases * rjson converts r object into JSON objects and vice versa * ggplot2 for creating graphs of our results * zoo for ordering indexed observations * tm for text mining applications * snowballc for implementing Porter's word stemming algorithm to collapse words into the comparable vocabulary * textcat for text categorization * catools for basic utility functions in our modeling * rpart for recursive partitioning of our tree models * randomforest for developing our tree based models to learn the sentiment * caret for our predictive models * e1071 for latent class analysis * wordcloud for developing word clouds from our input

```
library(lexicon)
library(spacyr)
library(SentimentAnalysis)
```

```
##
## Attaching package: 'SentimentAnalysis'

## The following object is masked from 'package:base':
##
##     write
```

```
library(rjson)
library(ggplot2)
library(zoo)
```

```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(tm)
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(SnowballC)
library(textcat)
library(caTools)
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
library(e1071)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

1.2 Definition

Sentiment analysis is a subset of natural language processing (NLP) techniques performed on unstructured data to assess the view of, attitude toward or opinion of a situation. Specifically, through extraction of the text state and information provided within the text body, an algorithm will breakdown a message into topic chunks and assign a sentiment score to each chunk which is used to assess the polarity (significance of positive or negative feelings) directed toward subjects. Extending beyond simply the polarity of expression, modern day algorithms may also detect emotions expressed within the text including happiness, sadness, or anger.

Try an example for yourself using the open source sentiment analysis demo provided by Komprehend **here**.

1.3 Importance of Sentiment Analysis

It is estimated that 2.5 quintillion bytes of data are generated every day, of which 80% of is unstructured. For organizations attempting to compile and respond to customer feedback in real-time or people looking to assess trends in text or even institutions attempting to understand their brand, reading through data of this size is unfathomable. Utilizing sentiment analysis allows users to:

- Monitor customer or user preferences
- Tailor chat bots to user moods
- Assess customer feedback
- Escalate improvements rapidly
- Decrease customer turnover
- Quantify customer satisfaction

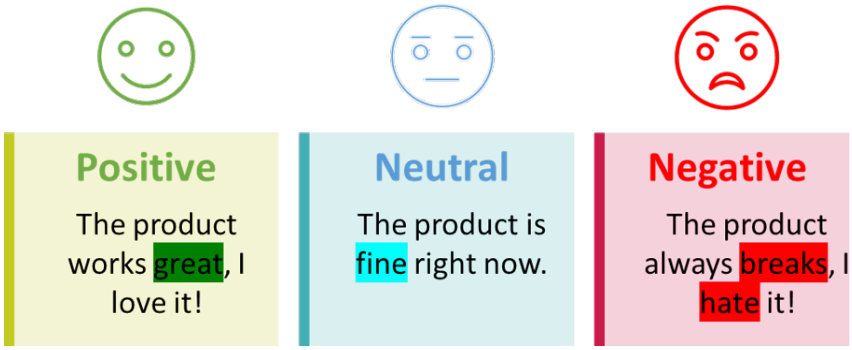


Figure 1: Text State and Sentiment

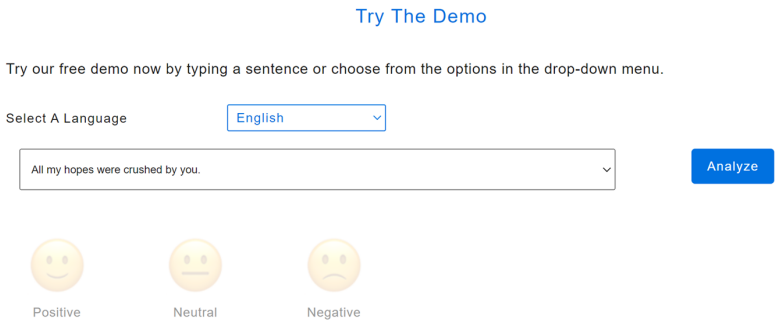


Figure 2: Komprehend Demo

1.4 Common challenges experienced

Common challenges with performing sentiment analysis include:

- **Text tone** - Statements written in a neutral sentiment yet meant to express a stronger polarity are often misinterpreted and incorrectly labelled.
  - Example - A customer complaint for an incorrect package after receiving a phone instead of a computer may read 'I received a phone' is neutral, yet is meant to express a negative emotion given that they did not receive their computer.
- **Phrases, idioms & cliches** - Expressing sentiment through common phrases specific to a culture or language need to be understood by the training system, otherwise phrases that appear positive by context and word scores are truly negative. Some common cliches have been listed in the R data set 'cliches' for training.
  - Example - An email response inquiring as to a users thoughts on a new feature may read 'it was pretty good' which the system read as having a high positivity score yet it should actually be considered neutral.

```
data(cliches)
head(cliches)
```

```
## [1] "a chip off the old block" "a clean slate"
## [3] "a dark and stormy night"  "a far cry"
## [5] "a fine kettle of fish"    "a loose cannon"
```

- **Sarcasm** - Use of sarcasm ironically in conversations can lead to a positive score given the words used, yet with context is interpreted negatively.
  - Example - A social media post from a conservative account responding to an anti-gun advertisement may read 'This advertisement is SO true, good job!' The algorithm may have a difficult time understanding the context to determine it is a sarcastic comment.
- **Emojis & GIFs** - Pictures or videos expressing various emotions are unable to be assessed by the NLP techniques used for sentiment analysis, and removing them from data may remove necessary context for the data to be correctly understood. Utilizing the R 'emoji sentiment' data, you can begin to assess the emoji sentiment.
  - Example - Assessing a message chain between two individuals, if one user inquires as to meeting at a specific time and the second user responds with a thumbs down emoji, removal of the emoji changes our understanding of the response.

```
data(emojis_sentiment)
head(emojis_sentiment)
```

```
##               byte                              name                          id
## 1 <f0><9f><98><80>                     grinning face lexiconnhlscfhhqlqocpjlbgkv
## 2 <f0><9f><98><81>  beaming face with smiling eyes lexiconwpujksvgujncexktvyrn
## 3 <f0><9f><98><82>           face with tears of joy lexiconhfasgeamcujygmeplmkn
## 4 <f0><9f><98><83>     grinning face with big eyes lexiconacmkvtolcfbnqnbojygf
## 5 <f0><9f><98><84> grinning face with smiling eyes lexiconwmfqfauheemspnjvgota
## 6 <f0><9f><98><85>        grinning face with sweat lexiconibjkxxzmsluvensxgups
##   sentiment polarity         category frequency negative neutral positive
## 1 0.5717540 positive smileys & people       439       37     114      288
## 2 0.4499772 positive smileys & people      2189      278     648     1263
## 3 0.2209684 positive smileys & people     14622     3614    4163     6845
## 4 0.5580431 positive smileys & people      1206       86     361      759
## 5 0.4220315 positive smileys & people      1398      191     426      781
## 6 0.1796537 positive smileys & people       462      135     109      218
```

- **Double negated phrases** - Double negated phrases can cause confusion with the algorithms used, misinterpreting the phrase given a high usage of otherwise negative words.
  - Example - A document may read a thought from a character saying 'I wasn't not excited about the event' which the algorithm may choose to read as a positive emotion about the upcoming event yet may be more neutral to negative.
- **Contradictions** - Statements that contradict one another between positive and negative comments can be difficult to parse into a single result, leaving the system scoring how many positive or negative phrases collectively were used rather than determining total context.
  - Example - An Amazon review for head phones may read 'The product is fantastic. It does not work well in the rain. I also broke it in the car. It does not have great volume quality outdoors.' It is unclear whether this is positive or negative yet may be perceived negatively given the over use of negative comments even if the user enjoys the product.

2. Bag of Words & Embeddings

2.1 Bag of Words

The bag of words approach utilizes tokenization which is a means of separating text into units (called tokens) where three types of groupings can be achieved:

- **Word** which identifies the individual word (ex. smarter)
- **Character** which separates each character into unique parts (ex. smarter becomes s-m-a-r-t-e-r)
- **Subword (n-gram characters)** which breaks the word into smaller word components (ex. smarter becomes smart-er)

The tokens are further prepared into a vocabulary which is the set of unique tokens in the corpus. Each word or token is counted to determine the number of times each word or character appears within the text being analyzed. Counting each word within the phrase, your algorithm would assess the sentiment using a lexicon or dictionary of known words and their associated sentiment to produce a sentiment score (per word) as to how much of a particular sentiment was present in the word. The score of the text or phrase is then fully assessed by producing a sum or average of the generated word sentiment scores.

```
sentiment_score <- analyzeSentiment("I love your cupcakes!")
sentiment_score
```

```
##   WordCount SentimentGI NegativityGI PositivityGI SentimentHE NegativityHE
## 1         2         0.5            0          0.5           0            0
##   PositivityHE SentimentLM NegativityLM PositivityLM RatioUncertaintyLM
## 1            0           0            0           0                  0
##   SentimentQDAP NegativityQDAP PositivityQDAP
## 1           0.5              0            0.5
```

```
convertToBinaryResponse(sentiment_score)$SentimentGI
```

```
## [1] positive
## Levels: negative positive
```

This is made possible through the use of a sentiment dictionary which contains individual words or phrases with an associated emotion and polarity that is commonly expressed with it. Once the score per word or phrase is determined, we can compute the resulting sentiment of the entire text by aggregating the collection of scores. The below example text is assessed using the Sentiment Analysis package which an associated dictionary.

This approach generates strong insights although remains limited by context. If you introduce 'do not' to the statement above, it cannot determine any difference in sentiment as it simply identifies the frequency of words and the words used rather than their order.

```
sentiment_score <- analyzeSentiment("I do not love your cupcakes!")
sentiment_score
```

```
##   WordCount SentimentGI NegativityGI PositivityGI SentimentHE NegativityHE
## 1         2         0.5            0          0.5           0            0
##   PositivityHE SentimentLM NegativityLM PositivityLM RatioUncertaintyLM
## 1            0           0            0           0                  0
##   SentimentQDAP NegativityQDAP PositivityQDAP
## 1           0.5              0            0.5
```

```
convertToBinaryResponse(sentiment_score)$SentimentGI
```

```
## [1] positive
## Levels: negative positive
```

Additionally, bag of words views the statement as unique, independent words rather than a single entity. If your statement included multiple negative words, the analysis would look at them independently.

2.2 Phrase Modeling

Phrase modeling aims to learn the combinations of tokens previously developed in order to represent multi-word concepts for determining sentiment (among other reasons). This is performed using the below equation which loops over word combinations to determine which set of words occur more often than they otherwise would with random chance.

$$\frac{count(A\ B) - count_{min}}{count(A) * count(B)} * N > threshold$$

Figure 3: Phrase Modeling Equation

- *count(A)* which is the count of A tokens in the document corpus
- *count(B)* which is the count of B tokens in the document corpus
- *count(A B)* which is the count of times tokens A B appear in the corpus in order
- N which is the size of the corpus vocabulary
- $count_{min}$ which is the parameter input by a user to determine the minimum times a phrase should occur to be accepted
- *threshold* which is the parameter input by a user to determine the strength of the relationship between tokens A and B to be accepted as a phrase for analysis

Following the model training across the document corpus, the model may be applied to new text to assess future phrases.

2.3 Embedding

With the growth of deep learning, natural language processing (NLP) has shifted from a bag of words approach to leveraging word and document embeddings.

2.3.1 Embedding Approach

The goal of the embedding algorithm is to learn numerical vector representations of each term in a corpus vocabulary. Such embeddings allow you to view words based on their proximity to other words which would provide a semantic meaning, meaning it relates the word to the meaning in the language or logic. In data science, this is done by assigning a series of numbers to each word in the text. Those words which are related to one another are considered 'near' whereas those words unrelated are 'far' from one another within the word space. By doing so, models learn not just the sentiment of individual words but those related to one another which greatly increases the model efficiency for document embeddings.

Chris Bail from Duke University provides an excellent visual on how this embedding functions.

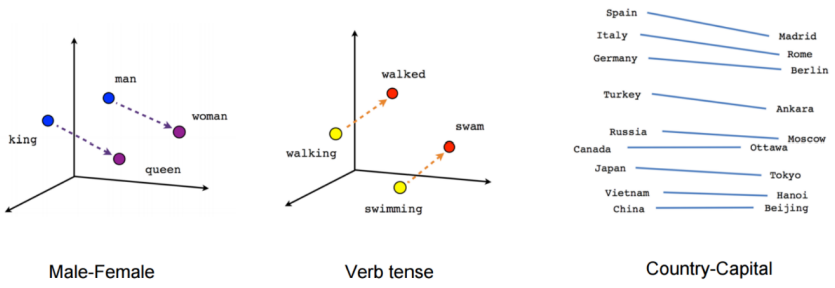


Figure 4: Word Embedding Visual

These models are performed in an unsupervised manner, learning the relationships by analyzing the text of the corpus without a prior target provided. One of the most-used resources for word vector modelling is **word2vec** which uses a neural network to learn word associations from the text corpus. This is done by using a sliding window technique where it evaluates smaller portions of text before and after the focal word only a few tokens long at a time.

2.3.2 Embedding Types

Word embedding algorithms will initially remove those words used as fillers or that are otherwise unimportant to the context of the discussion. The words that remain are apart of your 'context window', or a set of words built around a focal word or words used to train the model. These focal words tend to be those found in your sentiment dictionary with a historical sentiment attached (ex. happy, sad, bad, etc.). These word embedding models are often also referred to as word vector embedding models or word vector models.

There are two primary types of embedding models used:

- **Continuous Bag of Words (CBOW)** reads the context window words identified and provides a prediction on the most likely focal word
- **Skip-Gram Models** predicts the context words given the focal word

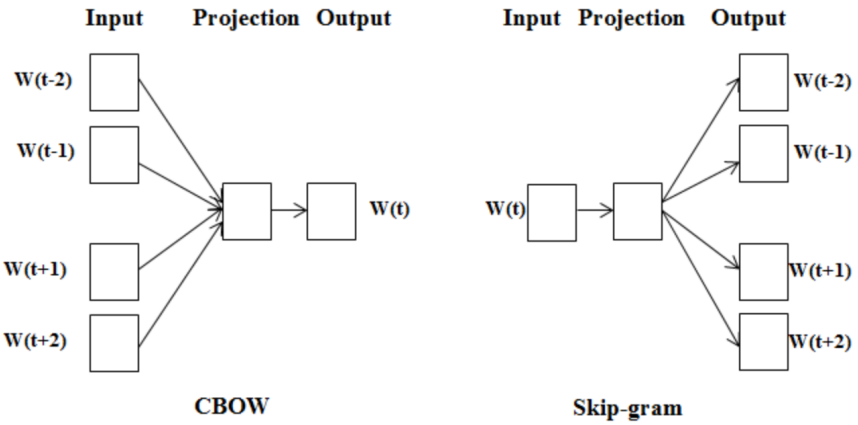


Figure 5: CBOW vs. Skip-Gram

3. Field of Sentiment Analysis

The most common sentiment analysis techniques used across data science currently include:

- **Fine-Grained or Graded Analysis** which determines a detailed assessment of the polarity expressed within the statement, returning precise levels of the polarity. The term 'Graded' refers to the assessed score returned, for instance expressing the polarity on a scale of 1-5 or from very negative to very positive. The refined scoring into more precise categories does come with a trade-off requiring greater cost-intensive and time-intensive training of the model.

Figure 6: Graded Polarity

- **Emotion Detection** focuses on the emotion of the text summary leveraging lexicons or groups of words that are proven to be positive or negative under specific contexts. Doing so allows the algorithm to separate combinations of words into positive and negative polarity followed by determining the probability of the total polarity based on a culminated score. Utilizing emotion detection allows the end-user to identify why the phrase received the polarity labeling. Utilizing the R Lexicon databases can assist with initial training of models. A list of available Lexicons can be found under the R Package Documentation **here**.
- **Aspect-Based Analysis** determines the specific topic of the text which the sentiment is directed toward. For example, applying aspect-based sentiment analysis to a product review stating 'I loved the weight of the product, it is perfect for my child!' would determine that it is a positive review specifically about the weight of the product, rather than about the product as a whole.

Figure 7: Aspect-Based Analysis Example

4. Sentiment Analysis Algorithms

4.1 Algorithms Available

There are three primary families of methods used in Sentiment Analysis Algorithms. These are: Rule-based approaches, Automatic approaches, and hybrid/ensemble approaches.

- **Rule-based approaches** (utilizing bag of words) use a set of manually created rules to identify polarity, emotion, or subjectivity. Common computational NLP techniques used among this type of algorithms include stemming, part-of-speech tagging, tokenization, and Lexicons. The following steps are an example of how this approach may work:
  - Define two (or more, if analysis is more than positive vs. negative) lists of polarizing words: one for negative words such as bad (process), terrible (service), poor (quality), and another one for positive words such as great, nice, and excellent.

```
lst_neg = list('hate', 'bad', 'terrible', 'poor')
lst_pos = list('love', 'great', 'nice', 'excellent')
```

- Count the number of negative and positive words that appear in a given text, such as a review for a restaurant.

```
txt = 'I loved the appetizer. The service is great.
Our server was really nice. But the food took a while to arrive'
```

- If the number of word appearances for the negative words is greater than that for the positive words, then the algorithm will return a negative sentiment for the given text. In the restaurant review example, we can see that the reviewer left three positive instances and 1 negative instance of a review about the restaurant. Our algorithm would then return +2 and deem it a positive sentiment.

One key advantage of Rule-based approaches is that they are easy to create and very easy to understand or interpret. But because of their naive nature, they cannot take into account the real sentiment of combinations of words. And with increasing number of rules, the complexity of the algorithm increases.

- **Automatic approaches** (utilizing embedding) use machine learning techniques and convert sentiment analysis into a classic classification problem. Similar to a regression problem, sentiment analysis can be done with training a model and using the model to make predictions.
  - The first step in a machine learning text classification is to transform the text extraction, also called text vectorization. The classical approach has been bag-of-words or bag-of-ngrams using frequencies of word appearances. Several examples of classification algorithm include Naive Bayes, LDA/QDA, Linear Regression, Support Vector Machines, and Deep Learning, each with their own unique advantages and disadvantages.
- **Hybrid approaches** simply take the advantages from the rule-based approaches and the automatic approaches for greater results, analogous to an ensemble model.

4.2 Sentiment Library's

- **Sentiment Library** is essential because it serves as a central bible for sentiment analysis. Sentiment libraries are collections of dictionaries that are manually scored. These libraries often contain phrases and adjectives, and are created before hand as a pre-processing step.

  - For example, an effective sentiment analysis will need to distinguish the level of positiveness between the word “beautiful” and the word “gorgeous”.
  - Customizations / personalizations can often be done in the creation of these libraries to make the results more accurate for certain topics, business, or type of personalities.

4.3 POS Tagging

- **POS Tagging** stands for “Part Of Speech Tagging”. It breaks down texts into natural language elements such as nouns, verbs, adjectives, adverbs, and prepositions so that these pieces can be processed similar to ways human process them. Grammars are essentially rules that can be translated into programming.
  - However, different languages have different levels of complexity, rules, and levels of exceptions. And as languages evolve, such as the case with English, rules often need to be modified or exceptions added. Thus, the scoring quality is dependent upon the POS tagging performed.

5. Use Cases in Business

The insights gained from sentiment analysis can transform businesses. It can be in areas that are creating either the most negative sentiment such as product features, customer service, prices, or more positive sentiment such as price matching and discounts. The strategic changes that sentiment analysis can bring about can aide businesses to:

- **Become competitive**
- **Attract and retain customers**
- **Improve customer service**
- **Improve marketing messages and campaigns**

Almost every industry is going through some form of digital transformation that results in large quantities of both structured and unstructured data being leveraged. The biggest challenge that companies have is turning their unusable, unstructured data into useful insights that can help them make data driven decisions, create operational efficiency's, and gain overall competitive advantages.

The following are primary examples of how sentiment analysis is currently transforming industries:

- **Banking** - Customers are becoming more vocal on social media about the services that are offered to them. Banks may face customer churn if they cannot provide the expected service. It is essential to stay in touch with your customers and analyze their demands. Banks can run sentiment analysis by listening to different hashtags on Twitter and extracting sentiment around their services. See the below example from Repustate regarding sentiment for varying features of the bank.

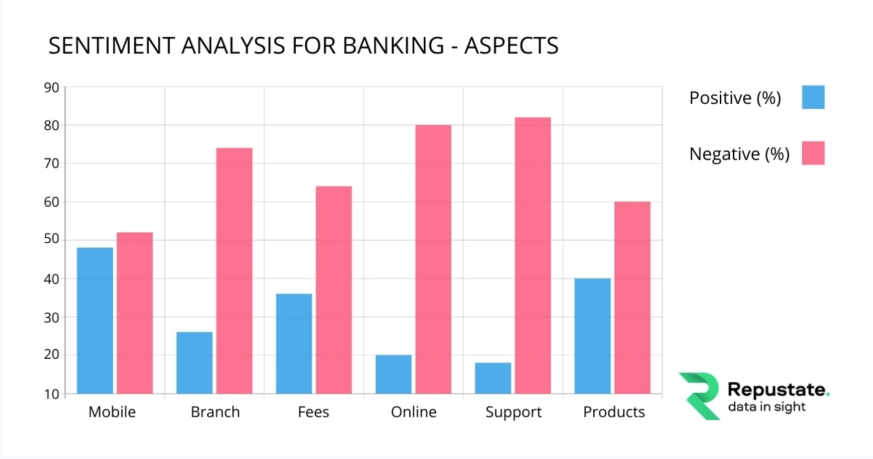


Figure 8: Repustate Sentiment on Banking Sector

- **Call Centers:** - Call centers could track and analyze all customer service representative (CSR) interactions to get a sense as to what the tipping point of a certain number of negative interactions was, that was causing customer attrition. They can first convert speech to text and analyze the sentiment related to the products and services offered to the customer. Once understood the company can change its script to create more positive customer interaction. It can be hard to understand customer interactions in a call center without having this kind of automation in place.
- **Government Sector:** - Governments are enhancing the efficiency of their public services with proactive monitoring of their citizens sentiment for public services (ex. waiting time at the DMV). Governments can analyze the heaps of feedback data they collect from citizens using different web portals and identify areas of pain points in real-time using sentiment analysis. Being able to make good decisions based on historical data allows governments to be more agile, efficient, and approachable.

- **Stock Market:** - High volatility in the financial market means the need for lightning-quick reflexes to make transactions in sub-second frequencies is intense. Hedge funds can build real-time dashboards that cover market sentiment and share prices for different debt instruments and equities. Sentiment analysis can help these financial companies make sense of the scores of information on news websites and social media. This may become a source of competitive advantage if it can be combined with their automated trading algorithms (ex. monitoring sentiment for online bluds pumping meme stocks).
- **Market Research:** - Market research firms run surveys with limited options to keep them structured and make sense of the output data. However, close-ended surveys inherently create biased results and may be misleading for the firms. So, such firms can produce more open-ended surveys about different products they are trying to analyze and run sentiment analysis on top of responses to extract crucial insights. Having this information at hand allows firms to develop specific strategies regarding their product roll-out and segments to target.
- **Hospitality Industry:** - Yelp, Google, and Airbnb reviews have generated large amounts of unstructured data online. Restaurants and hotels can perform sentiment analysis to understand the issues their customers have faced and fix their business. Granular insights from sentiment analysis of the restaurant reviews can help restaurants find out if their customers were unhappy about things that ranged from burnt pizza to very late delivery times. With these actionable insights on hand, they can make the changes necessary to ensure client satisfaction.
- **Retail:** - Clothing retail giants can analyze customer sentiment for changing industry trends to stay ahead of the competition. They can gain a holistic view of their problem area and look for in-depth semantic insights from social media websites. They can analyze social media trends and identifies patterns matched with user personas through semantic clustering and natural language processing techniques. They can gain deep insights into customer behavior based on a number of factors such as age, geography, language, pricing, fabric, design, etc. and can keep a tab on trends and stay ahead of the curve.

6. Walk Through Example

In this example we will perform sentiment analysis on the yelp reviews of restaurants using the different techniques mentioned above. Each review would be categorized as either positive, neutral or negative. We will verify the effectiveness of the different algorithms by comparing it with the number of stars given to the restaurant by the user i.e. any review with >4 stars is positive, 3 stars is neutral, and <3 stars is negative(total rating possible being 5 stars)

Load the data set

```
library("rjson")
json_data <- fromJSON(file='./yelp.json')
reviews.df <- data.frame(do.call(rbind, json_data))
```

Lasso Regression with Bag of Words

```
# create an actual_sentiment feature based on number of star ratings
reviews.df$actual_sentiment = ifelse(reviews.df$stars < 3, 'negative',
                                     ifelse(reviews.df$stars == 3, 'neutral',
                                            'positive'))
reviews.df$actual_sentiment = as.factor(reviews.df$actual_sentiment)

# loop through each review and calculate accuracy based on the number of
#correct predictions
correct_count = 0
for(i in 1:nrow(reviews.df)) {
  sentiment_score = analyzeSentiment(reviews.df[i,]$text[[1]])
  predicted_sentiment  = convertToBinaryResponse(sentiment_score)$SentimentGI
  if(reviews.df[i,]$actual_sentiment[[1]] == predicted_sentiment) {
    correct_count = correct_count + 1
  }
}

print(paste0('Accuracy is: ', (correct_count/nrow(reviews.df))))
```

```
## [1] "Accuracy is: 0.78"
```

Classification Trees with Bag of Words

```
# pull in larger dataset with 10K reviews
json_data <- fromJSON(file='./yelp_2.json')
reviews.df <- data.frame(do.call(rbind, json_data))

reviews.df$actual_sentiment = ifelse(reviews.df$stars < 3, 'negative',
                                     ifelse(reviews.df$stars == 3, 'neutral',
                                            'positive'))
reviews.df$actual_sentiment = as.factor(reviews.df$actual_sentiment)

create_bag_of_words = function(df) {
  #transforming text into lowercase, remove the punctuation and the common
  #English words (such as "this", "and", etc)
  text = VCorpus(VectorSource(df$text))
  text = tm_map(text, content_transformer(tolower))
```

```
  text = tm_map(text, removePunctuation)
  text = tm_map(text, removeWords,stopwords("english"))
  text = tm_map(text, stemDocument)

  #create bag of words
  freq = DocumentTermMatrix(text)
  sparse = removeSparseTerms(freq, 0.99)
  new.df = as.data.frame(as.matrix(sparse))
  colnames(freq) = make.names(colnames(freq))

  new.df$actual_sentiment = df$actual_sentiment
  return(new.df)
}


set.seed(1)

#use 90% of dataset as training set and 10% as test set
sample <- sample(c(TRUE, FALSE), nrow(reviews.df), replace=TRUE, prob=c(0.7,0.3))
train.df  <- reviews.df[sample, ]
test.df   <- reviews.df[!sample, ]

# create bag of words
bow_train.df = create_bag_of_words(train.df)
bow_test.df = create_bag_of_words(test.df)

train_cols = colnames(bow_train.df)
test_cols = colnames(bow_test.df)

# attach missing words to both training and test datasets to make them consistent
for (i in test_cols){
   if(!(i %in% train_cols)) {
     bow_train.df[[i]] = 0
   }
}

for (i in train_cols){
   if(!(i %in% test_cols)) {
     bow_test.df[[i]] = 0
   }
}

# create a classification tree model
ct.model = rpart(actual_sentiment~., data=bow_train.df, method="class")

# predict sentiment for restaurant
predict = predict(ct.model, newdata=bow_test.df, type="class")

# loop through each review in the test data set and calculate accuracy based
#on the number of correct predictions
correct_count = 0
for(i in 1:nrow(bow_test.df)) {
  if(predict[i] == bow_test.df[i,]$actual_sentiment) {
```

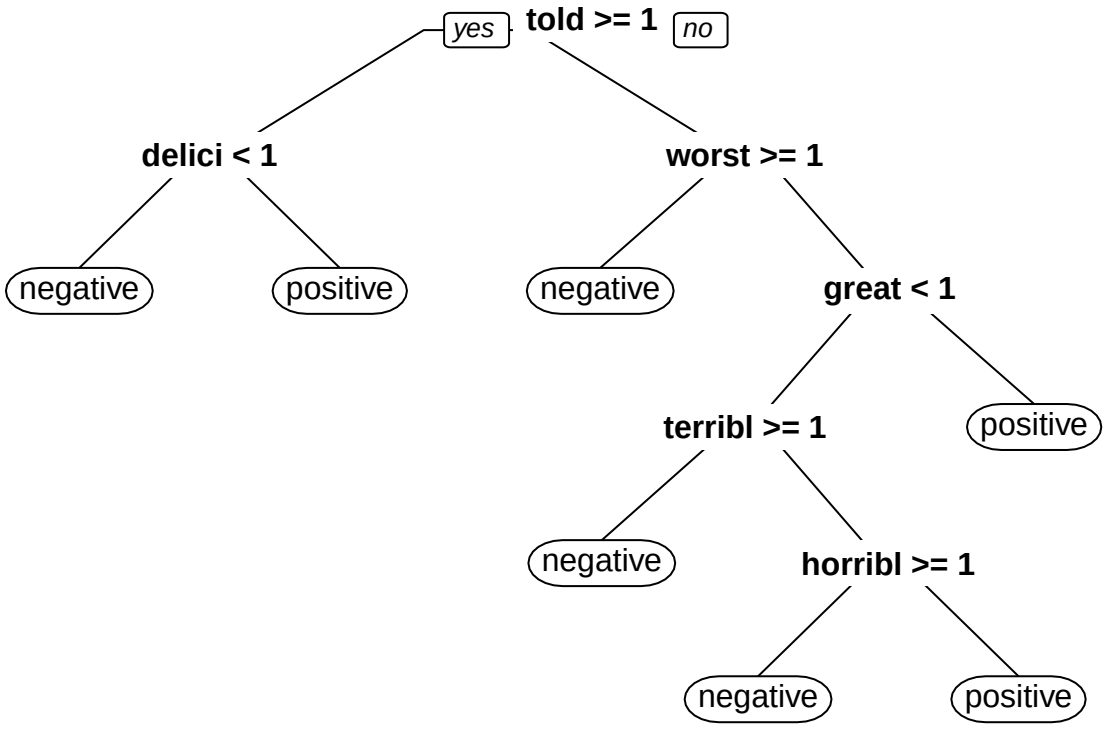
```
    correct_count = correct_count + 1
  }
}

print(paste0('Accuracy is: ', (correct_count/nrow(reviews.df))))
```

```
## [1] "Accuracy is: 0.223"
```

Classification tree model visualization

```
prp(ct.model)
```



Note: The Classification tree model has a low accuracy because the dataset used above is not large enough and has only 10,000 reviews. The model's accuracy can be improved by training it on a larger data set and also by doing cross-validation.

7. References

The following resources were used by our team in developing the above tutorial:

- Repustate real-world business use-cases
- Repustate data in sight
- tech target
- Wikipedia
- bitext
- Towards Data Science
- Chris Bail, PhD - Duke University

8. Appendix A

8.1 Word Embedding with Continous Bag Of Words (CBOW) Approach

Given the greater development of python libraries for word embedding, the following appendix has been added to provide transparency into how it is performed.

We will be using the python packages of:

- numpy for computing and data prep
- keras for reducing cognitive load
- gensim for topic modeling

In [1]:
```
import numpy as np
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda
from keras.utils import np_utils
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
import gensim
```

We have tagged each review with either a neutral, positive or negative sentiment based on the star rating.

In [2]:
```
data = open('yelp.txt','r')
corona_data = [text for text in data if text.count(' ') >= 2]
vectorize = Tokenizer()
vectorize.fit_on_texts(corona_data)
corona_data = vectorize.texts_to_sequences(corona_data)
total_vocab = sum(len(s) for s in corona_data)
word_count = len(vectorize.word_index) + 1
window_size = 2
```

We will generate pairs of the context words and the target words.

In [3]:
```
def cbow_model(data, window_size, total_vocab):
    total_length = window_size*2
    for text in data:
        text_len = len(text)
        for idx, word in enumerate(text):
            context_word = []
            target   = []
            begin = idx - window_size
            end = idx + window_size + 1
            context_word.append([text[i] for i in range(begin, end) if
                                 0 <= i < text_len and i != idx])
            target.append(word)
            contextual = sequence.pad_sequences(context_word,
```

```
                                                    total_length=total_length)
            final_target = np_utils.to_categorical(target, total_vocab)
            yield(contextual, final_target)
```

We will build the neural network model that will train the CBOW on our sample data.

In [4]:
```
model = Sequential()
model.add(Embedding(input_dim=total_vocab, output_dim=100,
                    input_length=window_size*2))
model.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(100,)))
model.add(Dense(total_vocab, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
for i in range(10):
    cost = 0
    for x, y in cbow_model(data, window_size, total_vocab):
        cost += model.train_on_batch(contextual, final_target)
    print(i, cost)
```

```
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
```

We will perform the vector file creation.

In [5]:
```
dimensions=100
vect_file = open('vectors.txt','w')
vect_file.write('{} {}\n'.format(len(vectorize.word_index.items()), dimensions))
```

Out[5]:
```
8
```

In [6]:
```
weights = model.get_weights()[0]
for text, i in vectorize.word_index.items():
    final_vec = ' '.join(map(str, list(weights[i, :])))
    vect_file.write('{} {}\n'.format(text, final_vec))
vect_file.close()
```

We will use the vector file in gensim model with the example word "neutral".

In [8]:
```
cbow_output = gensim.models.KeyedVectors.load_word2vec_format('vectors.txt',
                                                              binary=False)
cbow_output.most_similar(positive=['neutral'])
```

Out[8]:
```
[('fit', 0.26622945070266724),
 ('very', 0.24694503843784332),
 ('outside', 0.22985662519931793),
 ('cycle', 0.22688570618629456),
 ('did', 0.20111840963363647),
 ('smile', 0.19902928173542023),
 ('his', 0.19888196885585785),
 ('young', 0.19812779128551483),
 ('have', 0.19288483262062073),
 ('where', 0.18997474014759064)]
```

The above output shows all the words that are related to the review that was tagged neutral.

In [11]:
```
cbow_output.most_similar(positive=['positive'])
```

Out[11]:
```
[('nice', 0.2773880362510681),
 ('good', 0.19985781610012054),
 ('too', 0.17834430932998657),
 ('instructors', 0.16947823762893677),
 ('ideas', 0.16248522698879242),
 ('up', 0.15913422405719757),
 ('all', 0.14724114537239075),
 ('struggles', 0.1456916332244873),
 ('over', 0.14495055377483368),
 ('here', 0.14492066204547882)]
```

The above output shows all the words that are related to the review that was tagged positive.

After generating word vectors using CBOW, we can use Convolutional Neural Network (CNN) to train through labeled training set to capture the semantic features of the text.

Note: We just tagged each review with postive, neutral and negative just for the above example purpose to demonstrate how CBOW can generated related words. To actually build a sentiment analysis model we would have to use CNN on top of the wordvector as mentioned above.

In [ ]:
```
```