IST736 Text Mining - Homework 1

Martin Alonso

## Introduction

The objective of this exercise is to perform sentiment analysis on a data set of texts regarding how people feel about artificial intelligence (AI) on social media; and whether we can accurately categorize whether people have a positive or negative sentiment towards AI.

## Methodology

For this exercise, I will mainly employ the use of the sklearn and nltk packages to understand the data set and predict the outcomes. I will also use pandas, numpy, seaborn, and matplotlib.pyplot to do initial data analysis, attempting to find any patterns in the data prior to modeling it. I’ll also import the re package to clean the tweets.

import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
import re   
# sklearn package.  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.svm import LinearSVC  
from sklearn.metrics import accuracy\_score  
# nltk package.  
import nltk  
from nltk.classify import SklearnClassifier

As for the dataset, it was obtained from Twitter using the twitterscape package [1]. This was done prior to building this report as the data collection requires the use of the command prompt. The data set, however, includes 200 tweets that feature either ‘AI’ or ‘Artificial Intelligence’ within their text. The data was also transformed prior to loading the data, dropping columns regarding the url, html, likes, retweets, and fullname of the user; while also adding an id and label column. The label column is a categorical column that indicates the strengh of the sentiment from 0 to 2. In order: 0 = Negative, 1 = Neutral, 2 = Positive. This label was manually coded, so expect some slight bias in it.

# Load the data set.   
df = pd.read\_csv('ai\_tweets.csv')  
print(df.shape)

## (201, 5)

print(list(df.columns.values))

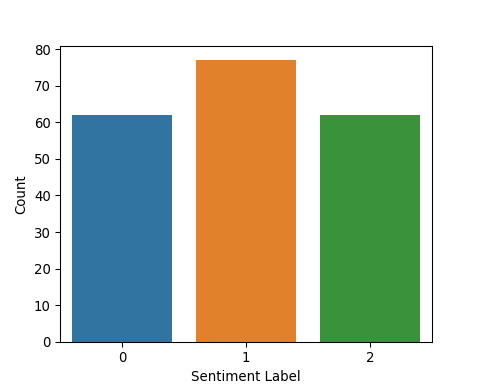
## ['id', 'label', 'user', 'timestamp', 'text']

Having loaded the data, I’ll clean it, removing special characters and converting uppercase letters into lowercase. This will allow me to easily vectorize all words in the data set.

# Transform the data.  
df['label'] = df['label'].astype(object)  
cleanText = []  
for f in df['text']:   
 text = re.sub('(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])', '', f).lower()  
 cleanText.append(text)  
cleanText = pd.DataFrame(cleanText)  
df2 = pd.merge(df, cleanText, how='outer', left\_index=True, right\_index=True)  
df2.rename(columns={0:'cleanText'}, inplace=True)

With the data clean, let’s check how the sentiment labels are distributed.

# Check label distribution.   
sns.countplot('label', data=df)  
\_ = plt.xlabel('Sentiment Label')  
\_ = plt.ylabel('Count')  
plt.show()



The counts for each sentiment label are almost equally distributed. With that, let’s start performing some sentiment analysis. I’ll first start with building both a training and testing set for the classifiers. After that, I’ll use the sklearn package to build a first classifier, and then move to nltk package to build a second model to compare results.

# Convert label back to int.  
df2['label'] = df['label'].astype('int')  
# Split the data into train and test sets.  
np.random.seed(42)  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(df2['cleanText'], df2['label'], test\_size=0.33, random\_state=3)

With tha data sets split, time to use sklearn.

# Because we have such a small data set, I'm setting a minimum of 2 repeats per word.  
tf = TfidfVectorizer(min\_df=2, max\_df=0.8, sublinear\_tf=True, use\_idf=True, stop\_words='english')  
# Count and transform the training and testing sets.  
x\_train\_tfidf = tf.fit\_transform(x\_train)  
x\_test\_tfidf = tf.transform(x\_test)  
# Now to use an svm to create the model.   
model1 = LinearSVC()  
model1.fit(x\_train\_tfidf, y\_train)  
result1 = model1.predict(x\_test\_tfidf)  
print(result1)

## [1 2 1 1 2 0 0 2 1 1 0 0 0 1 2 1 1 0 0 2 1 0 0 0 1 2 1 2 1 0 1 0 1 1 0 0 1  
## 1 1 1 1 1 1 1 1 2 1 1 2 0 2 2 0 2 1 0 1 2 1 2 1 2 1 0 1 1 1]

And now for nltk

# Rebuild the sets using the x\_train index to maintain comparability.  
indexTrain = x\_train.index.values  
indexTest = x\_test.index.values  
train = df2.iloc[indexTrain, [5, 1]]  
test = df2.iloc[indexTest, [5, 1]]  
# Clean the words in the training set tweets and create a list with words and labels.  
tweets = []  
for index, row in train.iterrows():  
 # Lower all words and keep those that appear 2 or more times.   
 wordsFiltered = [e.lower() for e in row.cleanText.split() if len(e) >= 2]  
 wordsCleaned = [word for word in wordsFiltered  
 if 'http' not in word  
 and not word.startswith('@')  
 and not word.startswith('#')  
 and word != 'RT']  
 tweets.append((wordsCleaned, row.label))  
# Obtain words and frequency distributions for every word in tweet.   
def getWordsTweets(tweets):  
 all = []  
 for (words, label) in tweets:  
 all.extend(words)  
 return all  
def getWordFeatures(wordlist):  
 wordlist = nltk.FreqDist(wordlist)  
 features = wordlist.keys()  
 return features  
wFeatures = getWordFeatures(getWordsTweets(tweets))  
def extractFeatures(document):  
 docWords = set(document)  
 features = {}  
 for word in wFeatures:  
 features['contains(%s)' % word] = (word in docWords)  
 return features   
# Splits the test set into positive, neutral, and negative to predict within their categories.  
testPos = test[test['label'] == 2]  
testPos = testPos['cleanText']  
testNrl = test[test['label']==1]  
testNrl = testNrl['cleanText']  
testNeg = test[test['label'] == 0]  
testNeg = testNeg['cleanText']  
# Build the second model.  
trainingSet = nltk.classify.apply\_features(extractFeatures,tweets)  
model2 = nltk.NaiveBayesClassifier.train(trainingSet)

## Results

Having built the two models, it is time to compare the results and decide which one is more appropriate.

## SVM Accuracy: 0.43283582089552236

## NLTK Accuracy: 0.47761194029850745

## 4

## 25

## 3

We can see by the results that both models have low accuracy scores. However, this may be due to either the size of the dataset or the data not be clean enough to accurately predict the sentiment of the text. Nevertheless, we can safely say that the NLTK model produces a superior accuracy score, correctly predicting 47.8 percent of tweet sentiment.

## Conclusions

We can conclude that the tools are suitable for the task. Albeit, we would need more data and pay closer attention to cleaning the data to build a model more capable of determining the overall sentiment of social media towards AI. We also need to pay close attention to what people are tweeting about when talking about AI and our bias when categorizing the sentiment of these tweets. This plays a strong factor as to how we reckon people feel towards AI.

## References

1. Twitterscrape package: <https://github.com/taspinar/twitterscraper>