By: Darren Pierre (101015833) & Jordan King (101043160)

COMP 4900: Assignment 2

NLP, Sentiment Analysis & Machine Learning

Abstract:

We were tasked with the job of determining whether or not a movie review was a positive or negative review based on the language used in it. We were given 2 different datasets: one for training and one for testing. The training dataset is a few tens of thousand lines long, while the testing dataset is a few thousand lines long. Using sentiment analysis, NLP, our own Bernoulli Naive Bayes (which was implemented ourselves), and different models from the SciKit-learn module available in python, we experimented to determine which model would give us the best accuracy in terms of predicting reviews correctly. The final consensus through testing was that using the TF-IDF vectorizer with logistic regression from the SciKit-learn module gave us the best possible accuracy in comparison to some of the other methods that we tested. The testing to determine this was done through the built in k-fold method, using 10-fold cross-validation

Introduction:

For this assignment, we were given two data sets - a test and a train dataset - that contained movie reviews and whether or not those reviews were positive or negative reviews/ratings. Our goal was to use different models to determine a way to predict whether or a review was a positive review or a negative review. This was done using natural language processing, sentiment analysis, and machine learning with SciKit-learn modules. One model that we used was Bernoulli's Naive Bayes model implemented from scratch in python. Then we were also told to use another model from the SciKit-learn package and compare the results to our newly implemented Naive Bayes model. With the use of SciKit-learn's method, we determined through experimentation that the most accurate model used logistic regression as well as a TF-IDF vectorizer, both of which were already built in methods. This method provided us with around 86% accuracy in terms of accurately predicting if the movie review was positive or negative. The experiments were tested using the built in k-fold method (specifically the 10-fold cross-validation).

Datasets:

We are working with two different datasets - one for testing and one for training. Both datasets consist of 2 columns.

The training data (located in a file called "*train.csv*") contains 30,001 rows where the first row is the "review" and "sentiment" column headers, therefore it contains 30,000 rows of training data. The first column (labelled "review") is an uncleaned, plain text movie review and placed by an unknown individual about an unknown movie. The second column (labelled "sentiment") is essentially a binary choice described in plain text as to whether or not the movie review was a positive or negative review/rating, using the words "positive" and "negative" exclusively.

The testing data (located in a file called "*test.csv*") contains 10,001 rows where the first row is the "id" and "review" column headers, respectively. Therefore it contains 10,000 rows of data to use for testing. The first column (labelled "id") is a unique integer value given to each movie review starting at 0. Since we have 10,000 rows of testing data, we have unique identifiers ranging from 0-9,999. The second column (labelled "review") is an uncleaned, plaintext movie review much like in the training data file. These are ultimately what will be evaluated by our models and interpreted as either "positive" or "negative".

Through the research of common text preprocessing practices, a recurring theme was stop words removal. Stop words are commonly used words in a language. Using the nltk module we remove stop words from movie reviews. Another text preprocessing technique that we used was removing punctuation. We hypothesized that punctuation doesn't have a significant impact on whether a movie review is positive or negative. Therefore we removed punctuation from movie reviews using the string. punctuation module. We did some checking with the movie reviews and noticed that some reviews had some HTML tags.Html tags shouldn't be part of the movie reviews so we remove them using a regular expression. All three of the proposed techniques served as a base for our text preprocessing.

In addition, we've also used advanced text preprocessing techniques called stemming and lemmatization. Stemming is a process that strips a word into its root or base by removing either its prefix and suffix. We decided to use the SnowBallStemmer in nltk module to complete the stemming process. Lemmatization is a similar process to stemming but does it in a fashion that brings more context into the word. To perform the lemmatization process we use the Wordnet module found in nltk module.

Proposed Approach:

Our proposed approach was to use 3 different vectorizers: a bag of words, TF-IDF, and N-grams. Our motivation was as simple as we were trying to figure out the best method for vectorizing the information and coming up with the most accurate movie rating sentiment prediction model. Using a bag of words was fairly a good approach since there wasn't a significant difference between the best performing vectorizer. TF-IDF was used because it normalizes a word across the whole entire dataset. Overall TF-IDF produced the best results for our model. N-grams were used because it's another common way to represent text and perhaps could lead to great performance for a model. This vectorizer, in the end, gave the worst results out of the vectorizers. We had settled on using only unigrams and bigrams since our experimentation had proved that any situation involving trigrams or more was more detrimental to the overall accuracy of the prediction model than it was beneficial. A 10-fold cross validation was used because it's common practice to use relatively large value of folds for large datasets. For Bernoulli's Naive Bayes implementation, we had based our implementation off of the lecture slides and an online source cited below.

We decided to use logistic regression because of the simplicity and our familiarity with the model. This model was introduced to us earlier in the course and is commonly used for binary classification problems. We experimented with one of logistic regression hyperparameters the inverse regularization parameter to see which value would yield the best result. After our experiment for hyperparameter optimization, it revealed that the best value for inverse regularization was 1.

Results:

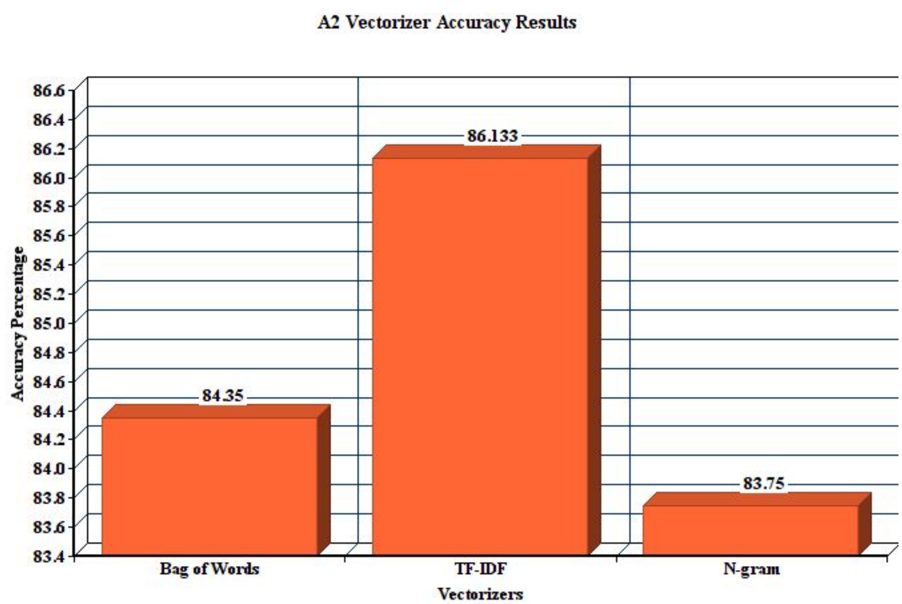

Figure 1: Results from our experiments on the vectorizer.

Ultimately, our model using logistic regression and SciKit-learn's TF-IDF vectorizer yielded the best experimental accuracy for predicting whether or not a movie review was positive or negative. It's average accuracy was ~86.133%. Using the count vectorizer was the next best model, yielding an average accuracy of ~84.35%. Using N-grams, specifically unigrams and bigrams (a parameter fed into the count vectorizer function), the average accuracy was around ~83.75%.

```
CountVectorizer accuracy score 0.8435
TfidfVectorizer accuracy score 0.8613333333333333
N-gram CountVectorizer accuracy score 0.8375
```

Figure 2: Results directly from the output of our code for the vectorizers.

All these results were generated using k-fold (10-fold) cross validation techniques on a uncleaned/unmodified version of the training set provided to us ("*train.csv*" file), with a maximum of 5,000 features set as a parameter to the vectorizers.

When running Bernoulli's Naive Bayes, the average accuracy score was ~84.643%. In comparison to our best method using logistic regression and TF-IDF, Naive Bayes falls flat by around 2%.

As of the last known update (updated at 9:45pm, Feb 27th, 2020, 2.25 hours before submission), our current leaderboard position on kaggle is 12th place with 87.133% accuracy.

Discussion & Conclusion:

Some important information to take away from this assignment is that all of our models, although slightly different, only yielded slightly different results in the end in terms of overall accuracy percentages. Clearly the TF-IDF method came out on top with the logistic regression model, but it's interesting to see that TF-IDF seemed to outperform the bag of words method every time in our experiments. This could be due to the fact that TF-IDF includes common words (such as "I", "we", "No", etc.) from the corpus where as the bag of words method does not.

It's also interesting to note that although the Naive Bayes does slightly beat the N-gram method, and it is technically slightly better than the bag of words method in terms of raw numbers (accuracy score), but there is very little to no statistical significance in that finding. Therefore, for argumentative sake, this implementation of Bernoulli's Naive Bayes could be classified as being equally as accurate as the bag of words method.

A possible direction is to use word embeddings as a way to represent features for text. Word embedding is a method of word representation that captures words that are similar in semantics in a neat way. There's various open-source word embedding models available to use such as wordtovec to experiment with.

Another possible direction would be to furthur play around with the parameters available to us in the SciKit-learn vectorizer. Currently we have tested different parameters such as the C value, max_features, ngram_range, binary, etc. Any parameter that was value-based was tested multiple times using a for loop with to iterate over different values in order to get the best result. For loop iterations started out large and progressively were narrowed down to a smaller subset that returned the best result.

Appendix:

COMP4900_A2.py

```
# -*- coding: utf-8 -*-
"""COMP4900A2Final.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1x3QWEltLmFhtiWiiLfOB8zYzyxdrGCz5
"""

from google.colab import drive
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
drive.mount('/content/drive')
TRAIN_DATA="/content/drive/My Drive/Colab Notebooks/Data Sets/train.csv"
TEST_DATA="/content/drive/My Drive/Colab Notebooks/Data Sets/test.csv"


# Loading up datasets into dataframes
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
trainDS=pd.read_csv(TRAIN_DATA)
testDS=pd.read_csv(TEST_DATA)

# Feature engineering test
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(trainDS.review, trainDS.sentiment,
train_size=0.80, test_size=0.20)
set_of_vectorizers=[CountVectorizer(max_features=5000),
                    TfidfVectorizer(max_features=5000),
                    CountVectorizer(max_features=5000,ngram_range=(1,2))]
```

```
names=["CountVectorizer","TfidfVectorizer","N-gram CountVectorizer"]


for i in range(len(set_of_vectorizers)):
  X=set_of_vectorizers[i].fit_transform(X_train)
  clf = LogisticRegression(max_iter=90000, C=1).fit(X,y_train)
  X_test_v=set_of_vectorizers[i].transform(X_test)
  pred_y=clf.predict(X_test_v)
  print(f"{names[i]} accuracy score {accuracy_score(pred_y,y_test)}")

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import      WordNetLemmatizer
import re

def removeStopwords(sentence):
  stop_words = set(stopwords.words('english'))
  filtered_sentence = [w for w in word_tokenize(sentence)  if not w in stop_words]
  return " ".join(filtered_sentence)

def remove_punctuation(sentence):
  return sentence.translate(str.maketrans('', '', string.punctuation))

def remove_html_tags(sentence):
  cleanr = re.compile('<.*?>')
  cleantext = re.sub(cleanr, '', sentence)
  return cleantext

# Normalization of words

def stemmize(sentence):
  stemmer = SnowballStemmer("english")
  stemmize_tokens=[stemmer.stem(token) for token in word_tokenize(sentence)]
  return " ".join(stemmize_tokens)

def lemmatize(sentence):
  wordnet = WordNetLemmatizer()
  lem_words=[wordnet.lemmatize(w) for w in word_tokenize(sentence)]
  return " ".join(lem_words)


def baseCleaner(sentence):
```

```
  func_cleaners=[removeStopwords,remove_html_tags,remove_punctuation]
  sentence=sentence
  for clean_func in func_cleaners:
    sentence=clean_func(sentence)
  return sentence
# normalize and use base cleaners to clean text
def stemBC(sentence):
  return(stemmize(baseCleaner(sentence)))
def lemBC(sentence):
  return(lemmatize(baseCleaner(sentence)))

# use both normalization techiniuqes considering the order
def both1(sentence):
  return(stemmize(lemBC(sentence)))
def both2(sentence):
  return(lemmatize(stemBC(sentence)))

# create datasets with clean data
stemmize_DS=trainDS.review.apply(stemBC)
lemmized_DS=trainDS.review.apply(lemBC)
both1_DS=trainDS.review.apply(both1)
both2_DS=trainDS.review.apply(both2)

from sklearn.model_selection import KFold
import numpy as np
from sklearn.model_selection import cross_val_score

kf = KFold(n_splits=10)
vectorizer = TfidfVectorizer(max_features=5000)
experimentDS=[trainDS.review,stemmize_DS,lemmized_DS,both1_DS,both2_DS]

for ds in experimentDS:
  vectors_train = vectorizer.fit_transform(ds)
  logReg=LogisticRegression(max_iter=90000000)
  scores = cross_val_score(logReg, vectors_train, trainDS.sentiment, cv=kf)
  avg_score = np.mean(scores)
  print(avg_score)

'''
# forloop iteration testing
from sklearn.metrics import classification_report


max = -1
```

```
index = -1
for i in range(0, 0):
  vectorizer = TfidfVectorizer(max_features=50000)
  vectors_train = vectorizer.fit_transform(X_train)
  clf = LogisticRegression(max_iter=90000000).fit(vectors_train, y_train)
  vectors_test=vectorizer.transform(X_test)
  predicted_values=clf.predict(vectors_test)
  print("***INDEX***", i)
  print(classification_report(predicted_values,y_test))

'''

#from sklearn.metrics import classification_report
vectors_test=vectorizer.transform(X_test)
predicted_values=clf.predict(vectors_test)
print(classification_report(predicted_values,y_test))

vectorRealTest=vectorizer.transform(testDS.review)
values=clf.predict(vectorRealTest)
df = pd.DataFrame()
df["id"]=testDS.id
df["answer"]=values

print(df)

from google.colab import files
df.to_csv('filename.csv')
#files.download('filename.csv')

"""
  # Title: Bernoulli Naive Bayes Classifier
  # Author:  Robert M. Johnson
  # Date: <date>
  # Code version: 1
  # Availability: https://mattshomepage.com/articles/2016/Jun/07/bernoulli_nb/
  """
def get_features(text):
    return set([w.lower() for w in text.split(" ")])

import numpy as np
from math import log

from collections import Counter
class NaiveBayesB():
```

```
  def __init__(self):
    self._log_priors = None
    self._cond_probs = None
    self.features = None

  def fit(self, documents, labels):

      label_counts = Counter(labels)
      N = float(sum(label_counts.values()))
      self._log_priors = {k: log(v/N) for k, v in label_counts.items()}

      X = [set(get_features(d)) for d in documents]
      self.features = set([f for features in X for f in features])
      self._cond_probs = {l: {f: 0. for f in self.features} for l in self._log_priors}

      for x, l in zip(X, labels):
          for f in x:
              self._cond_probs[l][f] += 1.

      for l in self._cond_probs:
          N = label_counts[l]
          self._cond_probs[l] = {f: (v + 1.) / (N + 2.) for f, v in self._cond_probs[l].items()}

  def predict(self, text):
    x = get_features(text)
    pred_class = None
    max_ = float("-inf")

    for l in self._log_priors:
        log_sum = self._log_priors[l]
        for f in self.features:
            prob = self._cond_probs[l][f]
            log_sum += log(prob if f in x else 1. - prob)
        if log_sum > max_:
            max_ = log_sum
            pred_class = l

    return pred_class

  def predictDS(self,dataset):
    return [self.predict(instance) for instance in dataset]

from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
```

```
# Loggistic Regression section
NUM_FOLDS=10
kf = KFold(n_splits=NUM_FOLDS)
X=both1_DS
y=trainDS.sentiment
vectorizer = TfidfVectorizer(max_features=5000).fit(X)
logRegression= LogisticRegression(max_iter=90000, C=1)
scores = cross_val_score(logRegression, vectorizer.fit_transform(X), y, cv=kf)
avg_score = np.mean(scores)
print(f"Logistic Regression result: {avg_score}")

# naive Bayes
scores=list()
for train_index, test_index in kf.split(X,y):
  clf=NaiveBayesB()
  X_train, X_test = X[train_index], X[test_index]
  y_train, y_test = y[train_index], y[test_index]
  # Naive Bayes Model
  clf.fit(X_train,y_train)

  predictions=clf.predictDS(X_test)

  scores.append(accuracy_score(y_test,predictions))

result=np.mean(scores)
print(f"Naive Bayes Result:{result}")

ds=[trainDS.review]

for x in ds:
  cls=LogisticRegression(max_iter=90000)
  vectorizer=TfidfVectorizer(max_features=50000)
  X=vectorizer.fit_transform(x)
  cls.fit(X,trainDS.sentiment)
  vectorRealTest=vectorizer.transform(testDS.review)
  values=cls.predict(vectorRealTest)
  df = pd.DataFrame()
  df["sentiment"]=values

  from google.colab import files
  df.to_csv('filename.csv')
  files.download('filename.csv')
```

Resources:

Prof. Majid's lecture slides

"User Guide - SciKit-Learn 0.22.1 Documentation". https://scikit-learn.org/stable/user_guide.html

Johnson, Robert M. "Bernoulli Naive Bayes Classifier." Mattshomepage, 2016. https://mattshomepage.com/articles/2016/Jun/07/bernoulli_nb/

""Word Embedding Tutorial: word2vec using Genism [EXAMPLE]". Guru99, 2020. https://www.guru99.com/word-embedding-word2vec.html