## Dan Popoutanu

## Reg Number: 1704805

## Task 3

## CE807 – Text Analytics

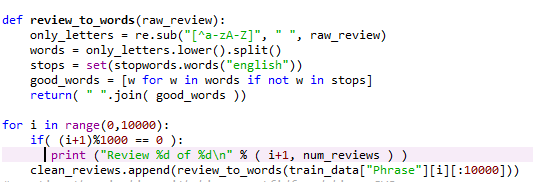
## Implementation Report

According to analysis in task 1 I have learned that it is important to explore and try out as many options as possible. As different researchers showed, there is no best way to do a sentiment analysis because it always differs depending on the dataset and the parameters used. As a beginner to anything related to machine learning and sentiment analysis, my first choice was using a Random Forest classifier because it felt like an easy implementation.

## Step 1 – Importing data

Even though I have seen different ways of doing it, one of the easiest ways was importing pandas and using the read\_csv method for both training and testing data. After importing the data, had some time playing around, counting how many of each sentiment is there and looking at different parts of the data.

## Step 2 – Cleaning data

As following my research the first obvious step at the time seemed cleaning the dataset. Creating a function that lowers all letters so capitalized letters would make no difference, removes all html elements that may be left or mistaken. Removing everything that is not a letter from a to z such as numbers, punctuation and other characters and finally removing all stopwords. 

In order to clean the data a for loop was used that was supposed to iterate over each element and remove all unwanted data. Since it is a good practice to start short I was only using 10.000 lines of the document to try out the classifier.

## Step 3 – Vectorizer

The two obvious options that a lot of people use and have created examples of are Tfidf Vectorizer and Count Vectorizer. Count Vectorizer takes each text and sets the number the specific word can be found. Term Frequency Inverse Document Frequency increases proportionally by count .

First example was using the CountVectorizer playing with different optimization options such as changing the analyzer string to either “ word” or “char” so that the features should be made of either words or character n-grams, setting different values for max\_features so that a vocabulary can be built. Setting unigrams and bigrams proved to be a success in increasing the score my algorithm could reach, finaly setting upon a unigram and bigram classifier with an n-gram range (1,2)



## Step 4 – Classifier

Sentiment analysis relies heavily upon trying out different possibilities. All the test that I have done can be found under the Testing Data chapter that contains details about the classifier and processing tools used, the time it took, and the score that was achieved.

One of the first baselines that I started with was using a Random Forest classifier with 100 estimators. What I did not understand at the time was the speed of Random Forest which is really low. Training a random forest classifier will take hours depending on the data that you provide, but the more and the better data you have, the better your classifier will be, and more time will be required. As an exercise it can be seen from my test cases that I only started with the first 10k lines, trying out different possibilities in order to assess the validity of Random Forest. Even though a slight improvement was obtained when using the tfidfvectorizer the time spent increased as well, and when compared to the obtained score it proved to be a wasted effort. As an example, a RandomForest classifier that is training on 20.000 lines took almost an hour to complete and only got a score lower than 0.6.

After a wasted effort on optimizing the Random Forest a basic test with Multinomial Naïve Bayes proved to get a better result compared to all the work previously done in a record time of 3 seconds over the entire dataset.

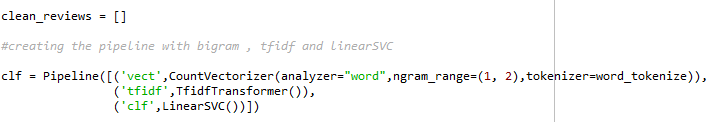
The final change was trying out the LinearSVC which provided great results in comparison with everything before. A classification method that is supposed to scale better for large numbers of samples

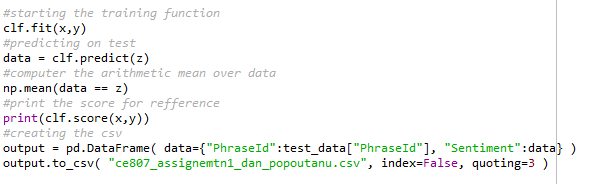
## Step 5 – Pipeline

Pipelines provide an easy way of adding steps and tweaking what you have already under a clear and concise part of code.

## Step 6 – Final Result

Taking it a step ahead I started experimenting with a basic pipeline with default variables for Count Vectorizer, tfidfTransformer and LinearSVC that proved to score higher than everything else so far getting a whooping score of 0.61. Further optimizing the Count Vectorizer with different parameters proved to work out for my dataset, making the features word-based, removing stopwords, lowercasing all letters, adding a max\_features option and so on. The final result was using the analyzer=”word”, an n-gram ranging from 1 to 2 and a tokenizer. Both the tfidfTransformer and LinearSVC were best used with their default optimizations in order get a high score.





## Testing data

A lot of the tests were simply trying out parameter optimization based on research. Tests contain the name of the classifier, the number of lines tested on, the time it took, and the score obtained on Kaggle.

Random Forest: 10k lines, coutvectorizer features 5k, no stopwords filtering – 0.531

Random Forest: 10k lines, countvectorizer features 1k, 113 seconds – 0.532

Random Forest: 10k lines, countvectorizer features 5k, 278 seconds – 0.541

Random Forest: 10k lines, tfidfvectorizer, 317 seconds – 0.547

Random Forest: 10k lines, tfidfvectorizer, set\_max\_features:5, 500 estimators 1474 seconds – 0.547

Random Forest: 20k lines, vectorizer features 5k, no stopwords filtering, 2465 seconds– 0.545

Radom Forest: 50k lines, vectorizer features 1k, no stopwords filtering 9870 seconds – 0.57

Basic Multinomial NB :3 seconds - 0.57

MNB ,Pipeline, countvectorizer with unigram, 27 seconds – 0.578

LinearSVC, Pipeline basics, countvectorizer, tfidftransformer, 10 seconds - 0.619

Pipeline tfidfvectorizer, 10 seconds, - 0.619

Pipeline, countvectorizer with unigram, features = 1k, 34 seconds – 0.583

Pipeline, countvectorizer with unigram, features = 5k, 34 seconds – 0.613

Pipeline, countvectorizer with unigram, features = 5k, preprocessor=lambda text, 33 seconds – 0.611

Pipeline, countvectorizer with brigram, ngram range (2,2), 33 seconds – 0.563

**Best result:**

Pipeline, countvectorizer with unigram and bigram classifier, ngram range (1,2) 35 seconds – 0.631

# Reflection

Most importantly what I have learnt from this exercise was paying close attention to my training and testing dataset. Having a dataset comprised of a sentence where each word of that sentence can be found on a separate line underlines the importance of stemming and stop words. Since sometimes in the training data a sentiment must be attributed to a stop words, removing them puts the classifier at a disadvantage which was reflected by the score of the classifier. Since the test data sometimes has stop words or even punctuation marks on a single line that need to be attributed a sentiment. With more time available, I am sure a better solution would arise. Even though the final score of 0.63 is amongst top 200 it is still a low score even with all the optimizations that have been considered.

In comparison to state of the art solutions, I have tried most of the code I could find online trying to adapt it as good as possible, but nothing scored better than the LinearSVC. I am sure there are better solutions out there and what I have may not be the best optimized for this scenario but with time I will develop a better understanding of sentiment analysis and what are the best choices predicting movie reviews. As documented in Task 1, I have tried both the NB approach as well as the SVM testing out different optimization settings but also tested out several approaches, as listed in the bibliography, from websites such as python for engineers or python spot.

Research Bibliography

Mihaela Sorostinean, Katia Sana, MOHAMED Mohamed, Amal Targhi (2017) Sentiment Analysis on Movie Reviews

Pang and L. Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In ACL, pages 115–124.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Chris Manning, Andrew Ng and Chris Potts. Conference on Empirical Methods in Natural Language Processing (EMNLP 2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank,

Sida Wang, Crhristopher D. Manning Baselines and Bigrams (2008): Simple, Good Sentiment and Topic Classification

Sida Wang

Saif M. Mohammad, Svetlana Kiritchenko, Xhiaodan Zhu (2013) : NRC-Canada, Building the State of the art in Sentiment Analysis of Tweets

# Website Bibliography

|  |  |
| --- | --- |
| [1] | P. Tutorials, "Python Sentiment Analysis," Python Tutorials, 1 jan 2017. [Online]. Available: https://pythonspot.com/python-sentiment-analysis/. [Accessed 15 Feb 2018]. |
| [2] | Dataquest, "Naive bayes: Predicting movie review sentiment," Dataquest, 17 March 2015. [Online]. Available: https://www.dataquest.io/blog/naive-bayes-tutorial/. [Accessed 21 Feb 2018]. |
| [3] | PyLing, "Movie Reviews Sentiment Analysis with Scikit-Learn," 7 Feb 2017. [Online]. Available: http://www.pitt.edu/~naraehan/presentation/Movie+Reviews+sentiment+analysis+with+Scikit-Learn.html. [Accessed 15 Feb 2018]. |
| [4] | J. Shaikh, "Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK.," 23 July 2017. [Online]. Available: https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a. [Accessed 15 Feb 2018]. |
| [5] | "PYTHON PIPELINES FOR SENTIMENT ANALYSIS," ANSWR, 16 July 2016. [Online]. Available: https://blog.answr.com/2016/07/18/python-pipelines-for-sentiment-analysis/. [Accessed 14 Feb 2018]. |
| [6] | R. Cranfill, "Building a Sentiment Analysis Pipeline in scikit-learn Part 2: Building a Basic Pipeline," 10 October 2016. [Online]. Available: https://ryan-cranfill.github.io/sentiment-pipeline-sklearn-2/. [Accessed 16 Feb 2018]. |
| [7] | B. Ivanov, "Getting started with Sentiment Analysis," 21 January 2017. [Online]. Available: https://nlpforhackers.io/sentiment-analysis-intro/. [Accessed 15 Feb 2018]. |
| [8] | Quierozf, "Scikit-Learn Pipeline Examples," 30 Jan 2018. [Online]. Available: http://queirozf.com/entries/scikit-learn-pipeline-examples. [Accessed 15 Feb 2018]. |
| [9] | Shantnu, "Natural Language Processing and Sentiment Analysis with Python," Python for Engineers, 2018. [Online]. Available: http://pythonforengineers.com/natural-language-processing-and-sentiment-analysis-with-python/. [Accessed 21 Feb 2018]. |