Game of Thrones sentiment analysis

Information Retrieval

By Itai Danielli

# Introduction:

Text classification is one of the subjects covered by text retrieval systems, while word count and TfIdf methods can be applied for sentiment analysis they can-not handle sarcasm Ex; “The episode was great ☺ in your dreams!”. When trying to analyze sentiments in tweets there are issues of shorthand and deliberate miss use of language for example “9ger” might be used instead of “nigger” to overcome lexical analyzers. Another problem is the presence of emoji’s not necessarily present in lexical analyzers.

## The method

These problems can be addressed my machine learning methods using data labeled by human which are likely to overcome such problems and best capture the tweet sentiment. For all of these reasons I chose to address the task of sentiment analysis using machine-learning algorithm trained by data available on the WEB. Machine learning based classifier requires two steps:

Calibrating:

During calibration the classifier passes a set of Train/Test cycles until it’s weights are best tuned for tweets evaluation.

Data from the WEB

ML classifier

Train/test

### Classifying:

Once the learning step is completed is the classifier can be used for classifying the operational text as if the system is classifying tweets coming from user’s.

G.O.T Data with classification

ML classifier

Raw G.O.T Data

## Existing methods:

In order to avoid “Inventing what is already common practice” I started by searching for available resources and existing methods already used for sentiment analysis. One such project is presented by Bert Carremans,

Code is available on <https://github.com/bertcarremans/TwitterUSAirlineSentiment>

Documented at <https://towardsdatascience.com/sentiment-analysis-with-text-mining-13dd2b33de27>.

Training data from <https://www.kaggle.com/crowdflower/twitter-airline-sentiment>.

## Issues with existing code/data:

The original source code was designed for the specific task of Kagel air flight tweets sentiment analysis as a note file rather than reusable code. Reusing this code for the G.O.T data had proven to be a major challenge because of outdated python code, design for small data size.

It is custom to have a classification challenges accompanied by a labeled data with which we can compare to, this task is provided without G.O.T tweets labels there for the system could not be evaluated properly, I tried to analyze a sub-set of 100 tweets but since I am a single judge none English speaker without the ability to see no emoji characters this evaluation may prove to be in accurate.

## Outcome:

During this project I have redesigned the original code in to a reusable up-to-date framework for classification of text stored in cvs files. The Framework requires a training file with a text column and a labels column for training the ML classifier which is tuned using by Keras machine learning framework. This project has opened my eyes to new methods for text analysis, to problems occurring when dealing with very large number of records, I have experienced using python Keras framework for integrating multiple text learning algorithms.

# Project Steps

This part describes the project work steps and the outcome of every step.

Ramp-up: The tutorial code was developed for Jupiter which is best for tutorials but less so for running applications, so I had to export the code to standard python code. The original code is not up to date with latest python version there for I had to update the code to fit latest python and libraries, download all peripheral libraries and ramp-up the project for PyCharm IDE.

Design: Redesign the code to enable separate Training and Prediction steps, the original code is provided with a single data and does not support the usage of separate operational step using the algorithm for unfamiliar tweets file. There for I have redesigned the source code to be modular and usable. The data was also changed for development, it is performed with a sub-set of tweets in the file ‘TweetsSmall.cvs’ from the file ‘TweetsFull.cvs’

The project architecture is multi-layer where base libraries contains the basic code and the TwitterSentiment file contains the classier and the major operational Programming interface.

The tweets sentiment analysis main file is stored at : <https://github.com/Danielli-Itai/PyTwitterSentiment>

This repository is based on two libraries: <https://github.com/Danielli-Itai/PyBase>

And the text processing library: <https://github.com/Danielli-Itai/PyBaseNlp>

Execution: Running the application with the operational file ‘ItaiGOT\_7\_B\_before\_SA-482000.excel.csv’ revealed the problem of Handling very large tweet files: When trying to load the G.O.T file in to memory and run the system, the application crashed.

The solution for the problem was a three step process: Split, Classify and merge.

Split: At first the G.O.T file is split in to 482 small size files each containing 1000 tweets (saved in temporary ‘split\_tmp’ folder)

Classify: Each file is split file is classified and the moved to the merge temporary folder ‘merge\_tmp’ containing the class for each tweet.

Merge: During this step the classified files are merged in to a single output file containing all tweets and classification.

# Text Processing Framework

The text processing framework has three major steps: Training, Prediction and Reporting. Each of these steps can be executed separately.

### Text Normalization:

The first step of training is preparing the text by performing the following filters.

Mentions: Mentions are removed to keep generalization of the model.

Hash tags: Removal of the hash tag sign (#) but keeping the tag text.

No informative: Removal of punctuations and digits.

URLs: Removal of the URLs (there is no a difference in the number of URLs between classes)

Emoji’s: Convert emojis into one a word.

Stop words: Removal of stop words

Stemming : Stemming using PorterStemmer.

### Classifier:

The classifier is based on three classifiers; Naïve Base, TfIdf and Word2Vec, during training each classifier is tuned using multinomial and logistic regression. The best performing algorithm is the used for the classification process.

## Training:

The M.L model includes two classification algorithms MulitnominalNB and logistic regression. Each algorithm uses three types of words representations: CountVectorizer (Term frequency), TfIdf and Word2Vec. Each of the classifiers is tuned for all three representations. The train and test data are selected using cross-validation (multiple deviations of test/train data used at multiple training sessions).

The model is trained on an airline data founded on: Airline tweets file from the Kagel competition which includes tweets labels ‘TweetsFull.csv’. The file contains 14,873 lines of tweets and a total size of 3,436,296 bytes.

### Training Performance:

The training takes about 10min and the class prediction for the same data takes merely 33sec. During training a small test case of prediction and reporting is executed.

Train - Total Time = 0:10:33.574155

Predict - Total Time = 0:00:33.949181

Report: Total Time = 0:00:34.910193

### Training results:

The following text describes the classifiers time consumption and scores. The cross validation score reflects the best score achieved during the cross validation training and the Test best estimator score reflects the best estimation score of the test data followed by the different scores regarding Negative, Natural and Positive tweets. The differences in performance are expected due to the fact that different of dada-size is available for each class.

\*\*\*\*\* BestContVect MultinomialNB done in 56.847s

Best Cross Validation score: 0.777 Test score with best\_estimator\_: 0.774

Classification Report Test Data precision recall f1-score

Negative 0.83 0.89 0.86

Neutral 0.59 0.47 0.53

Positive 0.71 0.69 0.70

\*\*\*\*\* BestContVect LogisticRegression done in 164.245s

Best Cross Validation score: 0.797 Test score with best\_estimator\_: 0.802

Classification Report Test Data precision recall f1-score

Negative 0.85 0.92 0.88

Neutral 0.67 0.52 0.58

Positive 0.71 0.70 0.70

\*\*\*\*\* BestTfIdf MultinomialNB done in 52.177s

Best Cross Validation score: 0.752 Test score with best\_estimator\_: 0.756

Classification Report Test Data Precision recall f1-score

Negative 0.76 0.96 0.85

Neutral 0.66 0.30 0.42

Positive 0.78 0.50 0.61

\*\*\*\*\* BestTfIdf LogisticRegression done in 150.895s

Best Cross Validation score: 0.787 Test score with best\_estimator\_: 0.801

Classification Report Test Data precision recall f1-score

Negative 0.82 0.95 0.88

Neutral 0.74 0.48 0.58 Positive 0.74 0.60 0.66

\*\*\*\*\* BestWord2Vec LogisticRegression done in 69.480s

Best Cross Validation score: 0.711 Test score with best\_estimator\_: 0.713

Classification Report Test Data Precision recal f1-score

Negative 0.75 0.93 0.83

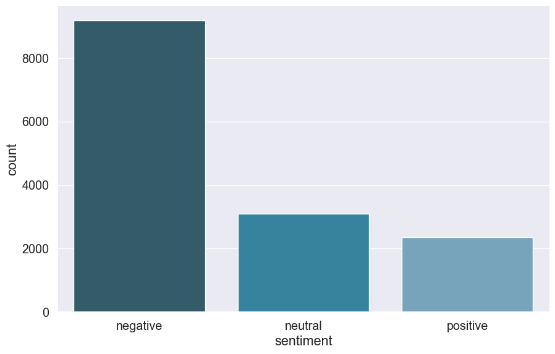
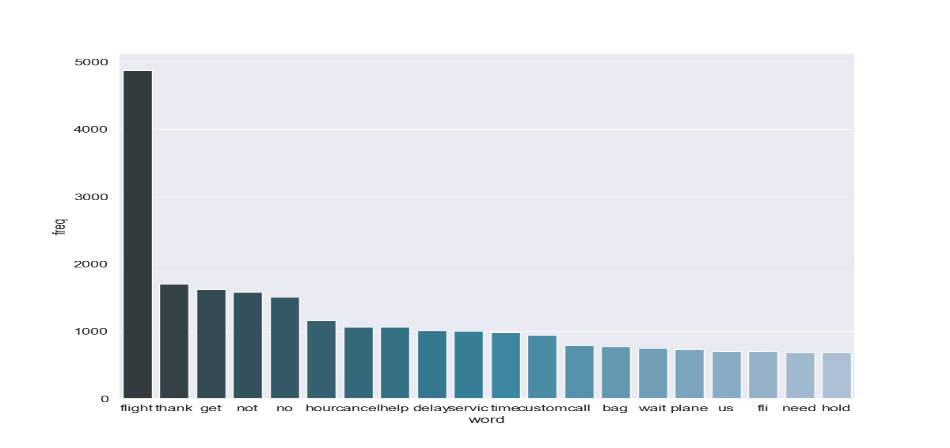
Neutral 0.58 0.33 0.42

Positive 0.57 0.29 0.39

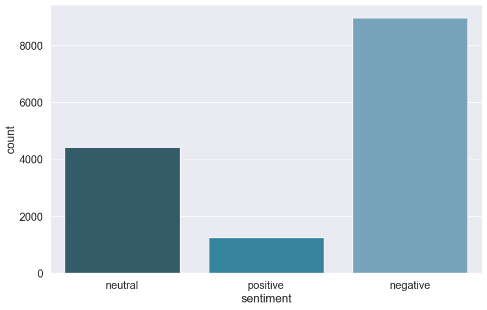
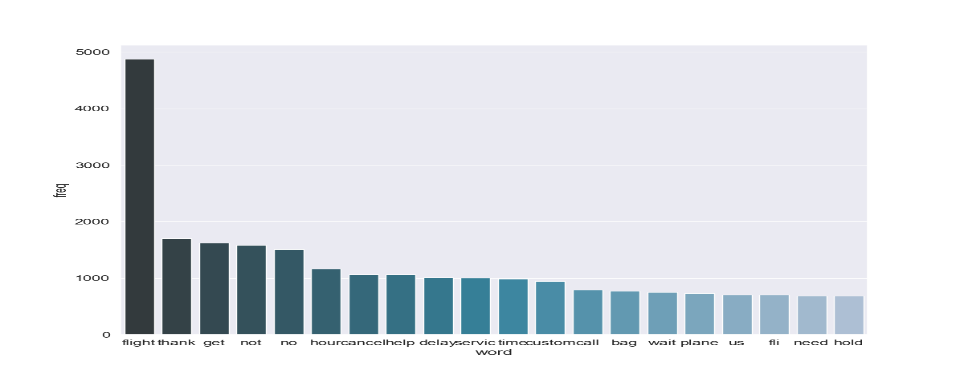
# Results

In order to evaluate performance I divided the report in to three steps:

Training data:

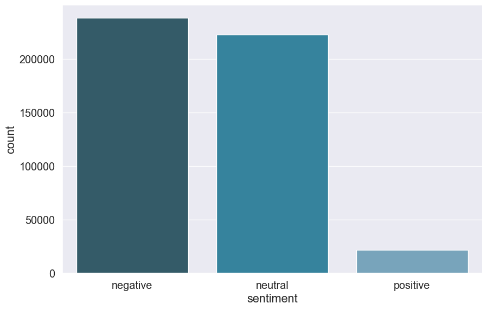
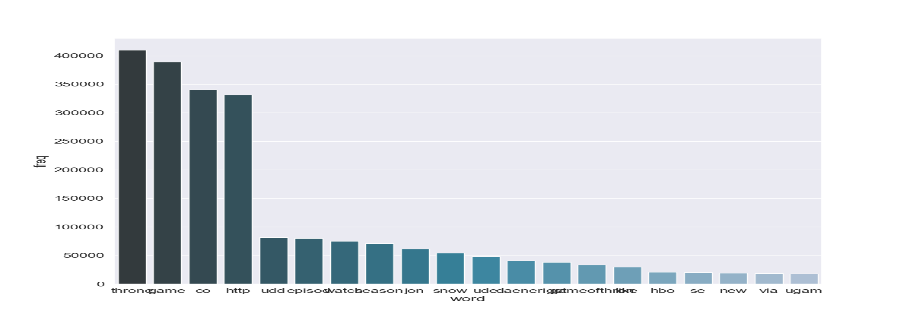
In the following graph’s we can see the distribution of the classes and the word count for a major words used in the database used before text normalization; Negative 9178.0, neutral 3099.0, positive 2363.0

## Classification of training data:

In this phase I used the model for classifying the training data on which it was trained, this step is done for software engineering reasons I order to validate that the software is working. The expected result is a good similarity between the training and training classification (unless under fitting had occur). Negative 8965 positive 1252 Neutral 4423

Although columns order has changed we can see Negative number remains around 9000 while the classifier has problems with positive which might be evaluated as neutral. We cannot make a definitive conclusion since classification errors may be different but based on the law of large number statistics we can assume this is what happened.

## G.O.T data

The game of thrown classification also shows a majority of negative but it is no longer far from neutral sentiment. The positive tweets are very rare as it appears on the training data; Negative 238,049, neutral 222,471.0, positive 21968.0

Word counts:

In the word count graphs we can see the resemblance of the distribution shapes but there is a big difference between the usages of words from the vocabularies, while AirLine tweets most frequent words are flight, thank… the G.O.T frequent words are thrown, game, co and URL’S. It is noticeable that word frequencies are not shared by the two document corpuses.

## Results Evaluation:

In order to evaluate the method performance I have manually categorized 50 tweets from the G.O.T collection and then compared my labels with the labels automatically generated by the M.L algorithm. Although the learning algorithm reported average precision of 0.6 during the learning process the actual precision I calculated is 18/50≈36% which is close to a random success of 1/3=33%. On the other hand this result cannot be trusted since the sample of 100 from 482,282 cannot in any case be a good representation of the data, added to that I cannot interpret Emoji’s.

This experiment has shown the disadvantages of using machine learning:

1. The training data may not represent the target data - From the words distribution it is clear that the terms used in the AirLine data is different than what is generally used in G.O.T.
2. The data from which the model was trained on is much smaller than that which it was used and that increases the risk of under fitting.
3. Good training data is hard to come by – I could not find a training data for G.O.T sentiment analysis.
4. Evaluation is also hard to come by – I could not evaluate performance since it will require about 10% which is 48,000 tweets out of the scope of this project.