V3_Project Proposal

I. Definition

I've decided to create a Twitter Sentiment Analysis as a mean of analyzing textual data. It is no secret that nowadays social networks influence people's perception of a given company(for example Trump's tweets). The perception of a given company could potentially influence buyer decision; as a result, a company could lose profit if a company looks unreliable or untrustworthy in customers' eyes or, in contrast, gain more profit if customers' perception is positive towards a company. This data could be accessed through Twitter tweets.

Based on this notion, I've decided to perform a simple Tweet Sentiment Analysis which could return the result for a given tweet whether it is positive or negative.

I've made a bit of research on the topic of the Sentiment Analysis and it's presence in the Academia as well as in the market.

Here is what I found:

I've used provided link and took this research paper (Feb 2016) as an example. I would like to cite a following piece of the text from the introduction:

> Sentiment analysis (SA) has become one of the most active and progressively popular areas in information retrieval and text mining due to the expansion of the World Wide Web (WWW). SA deals with the computational treatment or the classification of user's sentiments, opinions and emotions hidden within the text. Aspect extraction is the most vital and extensively explored phase of SA to carry out the classification of sentiments in precise manners. During the last decade, enormous number of research has focused on identifying and extracting aspects.

This quote resonates with my way of thinking about a given problem. Also, it's important to mention that NLP algorithms have been on the rise lately and will continue to be on the rise in the future, according to this article (Jul 2019).

Also, it's important to notice that this is not a new issue to be tackled, so there's a reliable source of data (dataset) which is provided by NLTK library - open source library, consisting of one the top-notch algorithms to be used for the Sentiment Analysis. I will provide the links later in reports. For now it's suffice to say that the dataset will consist of 20k of tweets already pre-defined as positive or negative.

Problem Statement

As I stated above the problem is that the trustworthy information about the perception of a company \ individual could be found in social networks. For this project, I will use NLTK's data corpus of tweets.

In a nutshell, the idea is the following: take a dataset of the tweets and based on sentiment analysis understand which words are leading a negative tweet and which to a positive tweet.

This type of information could useful when applied a certain company's Twitter. For example, we can take the Twitter of, say, Ryanair and based on the given amount of Tweets in the account of this company for a given amount of time (e.g just download tweets), Ryanair data scientist could understand the overall impression about the company by search the keywords indicating a positive tweet vs a negative tweet in the dataset.

This information then could be used by marking or PR teams to address a certain pain points (airport issues) or, as a contrast, to push the sales of a given product (e.g. flights).

How I will approach this problem:

I will prepare a dataset of sample tweets from the NLTK package for NLP using some cleaning methods. Once the dataset is ready for processing, I will train a model on pre-classified tweets (by NLTK) and use the model to classify the sample

tweets into negative and positives sentiments.
The process step by step(detailed explanation will be provided below):

- To take a given NLTK's tweet corpus
- Data tokenization step
- Data normalization step
- To clean the dataset (remove noise)
- To determine words density
- Optimize the data to feed the ML model
- (Build and Test the ML model) Find the most common words in positive tweets
- (Build and Test the ML model) Find the most common words in negative tweets

The solution is: to return to the user the result of a tweet whether positive or negative. Also, the project will provide an intuition how a certain words leading to a positive or negative tweet.

I will aim for the ⇒ 80% of accuracy on a test set.

This task could be solved by Supervised Learning algorithm. Here is why:

- We have two labels: negative and positive
- Hence, we are aware that this dataset should contain either positive tweets or negative. Therefore, this task could be classified as Supervised Learning case.

Metrics

A bit of information about the metrics I will use:

The problem is quantifiable: it is possible to quantify the number of tweets;

The problem is measurable: it is possible to measure whether a tweet is positive or negative;

The problem is replicable: it is possible to replicate this task because the solution to this problem could be used to understand the sentiment of the posts or comments.

Since we have only two classes: positive and negative, I assume that the accuracy metric will be the best bet here.

Accuracy will be defined as the percentage of tweets in the testing dataset for which the model was correctly able to predict the sentiment whether positive or negative.

I will be target at =>80% of the accuracy that the given tweet is positive or negative.

II. Analysis

Data Exploration

For train dataset, I will use NLTK's Twitter corpus, which currently contains a sample of 20k Tweets (named 'twitter_samples') retrieved from the Twitter Streaming API, together with another 10k which are divided according to sentiment into negative and positive.

I will use the negative and positive tweets to train the model on sentiment analysis(I will provide examples below). The tweets with no sentiments will be used to test your model.

The link to the source

The dataset will be splitted as shown below (for the project):

```
# 70/30
train_data = dataset[:7000]
test_data = dataset[7000:]
```

Also, it is important to note that initial segmentation of the test dataset has already been done NLTK (link).

I will import three datasets from NLTK that contain various tweets to train and test the model:

- `negative_tweets.json`: 5000 tweets with negative sentiments
- `positive_tweets.json`: 5000 tweets with positive sentiments
- `tweets.20150430-223406.json`: 20000 tweets with no sentiments

Using a Tweet Corpus

NLTK's Twitter corpus currently contains a sample of 20k Tweets (named '`twitter_samples`') retrieved from the Twitter Streaming API, together with another 10k which are divided according to sentiment into negative and positive.

```
from nltk.corpus import twitter_samples
twitter_samples.fileids()

# output
['negative_tweets.json', 'positive_tweets.json', 'tweets.20150430-223406.json']
```

I will pretty much follow the process explained in the TWITTER HOWTO section provided by NLTK's documentation.

The link

The example of a positive tweet from the initial dataset:

```
print("Positive tweet example:", positive_tweets[0])
# Positive tweet example: #FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this we
```

The example of a negative tweet from the initial dataset:

```
print("Negative tweet example:", negative_tweets[0])
# Negative tweet example: hopeless for tmr :(
```

Exploratory Visualization

I decided that the most simple and, therefore, easy-to-understand visualization will be the table view.
Here is the example from the code:

```
                    :) = True           Positi : Negati =   1002.1 : 1.0
                   sad = True           Negati : Positi =     55.8 : 1.0
              follower = True           Positi : Negati =     33.6 : 1.0
                   bam = True           Positi : Negati =     22.3 : 1.0
                  glad = True           Positi : Negati =     18.3 : 1.0
              followed = True           Negati : Positi =     15.8 : 1.0
               welcome = True           Positi : Negati =     15.8 : 1.0
               forward = True           Positi : Negati =     14.3 : 1.0
                  blog = True           Positi : Negati =     14.3 : 1.0
                   ugh = True           Negati : Positi =     13.7 : 1.0
```

How to interpret:
In the table that shows the most informative features, every row in the output shows the ratio of occurrence of a token in positive and negative tagged tweets in the training dataset.
The first row in the data signifies that in all tweets containing the token **:(**, the ratio of **negative** to positives tweets was 1002.1 to 1. Interestingly, it seems that there was one token with **:(** in the positive datasets. You can see that the top two insightful items in the text are the emoticons. Next, words such as **sad** and, again, quite interestingly, **followed**, lead to **negative sentiments**, whereas **follower** and **bam** are associated with **positive sentiments**.

Algorithms and Techniques

The technique to prepare the dataset will be the following:

- Data Tokenization
  Explanation: A token is a sequence of characters in text that serves as a unit. Based on how we will create the tokens, they may consist of words, emoticons, hashtags, links, etc.
  I will use the way of breaking language into tokens by splitting the text based on whitespace and punctuation.
- Data normalization step (stemming and lemmatization)
  Explanation: Normalization helps group together words with the same meaning but different forms. Without normalization, “cook”, “cooking”, and “cooked” would be treated as different words, even though you may want them to be treated as the same word.
- Data stemming step
  Explanation: Stemming is a process of removing affixes from a word. Stemming, working with only simple verb forms, is a heuristic process that removes the ends of words.
- Data lemmatization step
  Explanation: lemmatization normalizes a word with the context of vocabulary and morphological analysis of words in text. The lemmatization algorithm analyzes the structure of the word and its context to convert it to a normalized form. Basically, a comparison of stemming and lemmatization ultimately comes down to a trade off between speed and accuracy.
- To clean the dataset (remove noise)
  Explanation: Essentially, the "noise" is any part of the text that does not add meaning or information to data.
  Noise is specific to each project, so what constitutes noise in one project may not be in a different project. For example, the most common words in a language are called stop words. Some examples of stop words are “is”, “the”, and “a”. They are generally irrelevant when processing language, unless a specific use case warrants their inclusion.

  In this project I will use regular expressions in Python to search for and remove these items:

- **Hyperlinks** - All hyperlinks in Twitter are converted to the URL shortener t.co. Therefore, keeping them in the text processing would not add any value to the analysis.
- **Twitter handles in replies** - These Twitter usernames are preceded by a `@` symbol, which does not convey any meaning.
- **Punctuation and special characters** - While these often provide context to textual data, this context is often difficult to process. For simplicity, I will remove all punctuation and special characters from tweets.

- To determine words density
  Explanation: The simplest form of analysis on textual data is, to my mind, to take out the word frequency. A single tweet is too small of an entity to find out the distribution of words, hence, the analysis of the frequency of words would be done on all positive tweets. However, I will provide the information about negative tweets as well.
- Optimize the data to feed the ML model
  Explanation:

  Sentiment analysis is a process of identifying an attitude of the author on a topic that is being written about. As I have stated above It is a supervised learning ML process, which requires to associate each dataset with a “sentiment” for training. In this project, the model will use the “positive” and “negative” sentiments.

  Sentiment analysis can be used to categorize text into a variety of sentiments.

  A model is a description of a system using rules and equations. It may be as simple as an equation which predicts the weight of a person, given their height. A sentiment analysis model that I will use would associate tweets with a positive or a negative sentiment. I will split the dataset into two parts. The purpose of the first part is to build the model, whereas the next part tests the performance of the model.

  - Converting Tokens to a Dictionary
    Explanation: since I will use the Naive Bayes classifier in NLTK. It's important to remember that the model requires not just a list of words in a tweet, but a Python dictionary with words as keys and True as values. The "get_tweets_for_model" function makes a generator function to change the format of the cleaned data.
  - Splitting the Dataset for Training and Testing the Model
    Explanation:I need to attach a Positive or Negative label to each tweet. It then creates a dataset by joining the positive and negative tweets.

    By default, the data contains all positive tweets followed by all negative tweets in sequence. When training the model, I need to provide a sample of the data that does not contain any bias. To avoid bias, I've added code to randomly arrange the data using the `.shuffle()` method of `random`.

    Finally, the code splits the shuffled data into a ratio of 70:30 for training and testing, respectively. Since the number of tweets is 10000, I can use the first 7000 tweets from the shuffled dataset for training the model and the final 3000 for testing the model.

    Basically, what I've done in this step is: converted the cleaned tokens to a dictionary form, randomly shuffled the dataset, and split it into training and testing data.
- Build and Test the ML model
  Explanation: I will use the NaiveBayesClassifier class (NLTK library) to build the model.
  I decided to use this model based on the documentation explanation:

  Naive Bayes classifiers are paramaterized by two probability distributions:

  - P(label) gives the probability that an input will receive each label, given no information about the input’s features.
  - P(fname=fval|label) gives the probability that a given feature (fname) will receive a given value (fval), given that the label (label).

  If the classifier encounters an input with a feature that has never been seen with any label, then rather than assigning a probability of 0 to all labels, it will ignore that feature.

  Which is, to my mind, perfect for Sentiment Analysis Task.

Benchmark

The benchmark for the model will be the accuracy ⇒ 80%

Accuracy will be defined as the percentage of tweets in the testing dataset for which the model was correctly able to predict the sentiment whether positive or negative.

I will be target at =>80% of the accuracy that the given tweet is positive or negative.

III. Methodology

Data Preprocessing

I have throughly described this step in the "Algorithms and Techniques" step.

Implementation

Here I will explain more details about the algorithm I selected for this job. As I stated above I decided to use , NaiveBayesClassifier class to build the model.
Next, I will use the methods : .train() and the .accuracy() to train the model and to test the model on the testing data respectively.

Also, I will use method "classifier.show_most_informative_features()" to determine the most relevant features (to the task), and display them.

Also, I have documented several functions: remove_noise, get_all_words and get_tweets_for_model.

Refinement

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:

- *Has an initial solution been found and clearly reported?*
- *Is the process of improvement clearly documented, such as what techniques were used?*
- *Are intermediate and final solutions clearly reported as the process is improved?*

I don't see the necessity for the refinement based on the results I got. I consider this model has its job fairly well, based on the accuracy score and the test tweet I used in the end of the notebook.

IV. Results

The results are the following(I took some from the visualization segment of this report) :

```
Accuracy is: 0.9946666666666667
Most Informative Features
                     :) = True           Positi : Negati =   1002.1 : 1.0
                    sad = True           Negati : Positi =     55.8 : 1.0
               follower = True           Positi : Negati =     33.6 : 1.0
                    bam = True           Positi : Negati =     22.3 : 1.0
                   glad = True           Positi : Negati =     18.3 : 1.0
               followed = True           Negati : Positi =     15.8 : 1.0
                welcome = True           Positi : Negati =     15.8 : 1.0
                forward = True           Positi : Negati =     14.3 : 1.0
                   blog = True           Positi : Negati =     14.3 : 1.0
                    ugh = True           Negati : Positi =     13.7 : 1.0
None
```

How to interpret:

Accuracy: 99.46%

Which is way high when I initially expected (the baseline was more or equal to 80%)

Accuracy will be defined as the percentage of tweets in the testing dataset for which the model was correctly able to predict the sentiment whether positive or negative.

The interpretation:
In the table that shows the most informative features, every row in the output shows the ratio of occurrence of a token in positive and negative tagged tweets in the training dataset.
The first row in the data signifies that in all tweets containing the token **:(**, the ratio of **negative** to positives tweets was 1002.1 to 1. Interestingly, it seems that there was one token with **:(** in the positive datasets. You can see that the top two insightful items in the text are the emoticons. Next, words such as **sad** and, again, quite interestingly, **followed**, lead to **negative sentiments**, whereas **follower** and **bam** are associated with **positive sentiments**.

Accuracy will be defined as the percentage of tweets in the testing dataset for which the model was correctly able to predict the sentiment whether positive or negative.

Accuracy will be defined as the percentage of tweets in the testing dataset for which the model was correctly able to predict the sentiment whether positive or negative.

Overall, the model has performed a good correlation between the positive and the negative tweets.

Additionally, I've tested the model performance on completely unknown data e.i "custom_tweet".

```
custom_tweet = "I ordered from AliExpress last week. They delayed my order for a week! I will never this service in the future."

# Result Negative
```

And it has predicted negative result which is correct, to my mind.

V. Conclusion

Free-Form Visualization

As I stated above, I decided to choose the most simple way of the visualization here: the table view, with the tables consisting of features which lead to a certain type of a sentiment (positive vs negative). Also, this table represents the quantifiable information on how the certain word relates to a positive or negative sentiment.

```
                      :) = True           Positi : Negati =   1002.1 : 1.0
                     sad = True           Negati : Positi =     55.8 : 1.0
                follower = True           Positi : Negati =     33.6 : 1.0
                     bam = True           Positi : Negati =     22.3 : 1.0
                    glad = True           Positi : Negati =     18.3 : 1.0
                followed = True           Negati : Positi =     15.8 : 1.0
                 welcome = True           Positi : Negati =     15.8 : 1.0
                 forward = True           Positi : Negati =     14.3 : 1.0
                    blog = True           Positi : Negati =     14.3 : 1.0
                     ugh = True           Negati : Positi =     13.7 : 1.0
```

Reflection

The most interesting and difficult aspect of the project was to transform the data to meaningful information for the model to be used. It means that I have to transform the natural human language to a machine language (e.g by tokenization). Which itself could be described as a sequence of characters in text that serves as a unit. Actually, to ease the problem of tokenization I've used the punkt module. It is a pre-trained model that helps tokenize words and sentences.

Also, normalizing the data was a challenge.

Improvement

The model performing nicely for tasks to distinguish negative vs positive tweets. Nevertheless, it could be adjusted to sarcasm tweets in the future. One of the solutions could be to use additional label: sarcasm. The reason for this is simply because sarcasm itself is neither positive or negative it is *sarcastic type of expression,* which is sometimes even for humans difficult to digest.