**DOCUMENTATION**

***Sentiment Analysis : Task-1***

By- Krishna Pal Deora

[**Sentiment analysis**](https://monkeylearn.com/sentiment-analysis/) (a.k.a *opinion mining*) is the automated process of identifying and extracting the subjective information that underlies a text. This can be either an opinion, a judgment, or a feeling about a particular topic or subject. The most common type of sentiment analysis is called ‘polarity detection’ and consists in classifying a statement as ‘positive’, ‘negative’ or ‘neutral’.

For example, let’s take this sentence*: “I don’t find the app useful: it’s really slow and constantly crashing”.* A sentiment analysis model would automatically tag this as *Negative*.

STEPS INVOLVED IN COMPLETING THIS TASK:

1. Extracting of tweets from Twitter: **Extract\_TwitterTweets.ipynb**

First of all, I extracted about 3000 tweets from twitter using Twitter API credentials obtained after making a Twitter Developer Account. These 3000 tweets were obtained using 3 hashtags namely- #Corona, #BJP and #Congress. To connect to Twitter’s API, I have used a Python library called Tweepy, which is an excellently supported tool for accessing the Twitter API. It supports Python 2.6, 2.7, 3.3, 3.4, 3.5, and 3.6.

This [**Python script**](https://github.com/feconroses/gather-tweets-from-search) allows you to connect to the Twitter Standard Search API, gather historical tweets from up to 7 days ago that contain a specific keyword, hashtag or mention, and save them into a CSV file.  
  
This involves:

* Tweet content: text of the tweet
* Date: date and hour of the tweet
* User: name of the author of the tweet
* Tweet ID
* Tweet URL

Then, all the emojis and links were removed from these tweets. Because that’s a must, now-a-days people don’t tweet without emojis, as in a matter of fact it became another language, especially between teenagers so have to come up with a plan to do so.

1. Exploratory Data Analysis: **TESTDATACleaning.ipynb**

Once we have captured the tweets we need for our sentiment analysis, it’s time to [prepare the data](https://monkeylearn.com/text-analysis/#data-preparation). As social media data is unstructured. That means it’s raw, noisy and needs to be cleaned before we can start working on our sentiment analysis model. This is an important step because the quality of the data will lead to more reliable results.

Preprocessing a Twitter dataset involves a series of tasks like removing all types of irrelevant information like special characters, and extra blank spaces.

Data cleaning involves the following steps:

* **Convert tweets to lowercase using .lower() function**, in order to bring all tweets to a consistent form. By performing this, we can assure that further transformations and classification tasks will not suffer from non-consistency or case sensitive issues in our data.
* **Remove ‘RT’**, **UserMentions** and **links:** In the tweet text, we can usually see that every sentence contains a reference that is is a retweet (‘RT’), a User mention or a URL. Because it is repeated through a lot of tweets and it doesn’t give us any useful information about sentiment, we can remove them.
* **Remove numbers:**Likewise, numbers do not contain any sentiment, so it is also common practice to remove them from the tweet text.
* **Remove punctuation marks and special characters:**Because this will generate tokens with a high frequency that will cloud our analysis, it is important to remove them.
* The Twitter handles are already masked as @user due to privacy concerns. So, these Twitter handles are hardly giving any information about the nature of the tweet.
* Most of the smaller words do not add much value. For example, ‘pdx’, ‘his’, ‘all’. So, we remove all the stop-words as well from our data.
* Once we have executed the above three steps, we can split every tweet into individual words or tokens which is an essential step in any NLP task.
* Stemming & Lemmatization: We might also have terms like loves, loving, lovable, etc. in the rest of the data. These terms are often used in the same context. If we can reduce them to their root word, which is ‘love’, then we can reduce the total number of unique words in our data without losing a significant amount of information.

**Then, I have predicted the sentiment of these tweets using TextBlob library of Python.**

TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

The core of sentiment analysis is to use TextBlob in order to extract the **polarity** & **subjectivity** from tweet texts, which is actually done by the data preprocessing for better data storage. Negative tweets represent as -1, positive tweets represent as +1, and neutral tweets represent as 0 .

Polarity is float which lies in the range of [-1,1] where 1 means positive statement and -1 means a negative statement. Subjective sentences generally refer to personal opinion, emotion or judgment whereas objective refers to factual information. Subjectivity is also a float which lies in the range of [0,1].

1. Predicting sentiment using Stanford Core NLP library : **StanfordNLP.ipynb**

The Stanford CoreNLP is a Java natural language analysis library that provides statistical NLP, deep learning NLP, and rule-based NLP tools for major computational linguistics problems, that can be incorporated into applications with human language technology needs. Stanford CoreNLP integrates many NLP tools, including the Parts of Speech (POS) tagger, the Named Entity Recognition (NER), the parser, coreference resolution system, the sentiment analysis tools, and provides model files for analysis for multiples languages.

The scale for sentiment values ranges from zero to four. Zero means that the sentence is very negative while four means it’s extremely positive.

## Designing our own sentiment analysis tool : ****SentimentAnalysis\_Models.ipynb****

Predicting sentiment using my own designed models using the following Algorithms:

* LinearRegression
* LogisticRegression
* RandomForestClassifier
* LinearSVC (Support vector machine)
* Naive\_Bayes

## **While there are a lot of tools that will automatically give us a sentiment of a piece of text, it is observed that they don't always agree! Let's design our own to see both how these tools work internally, along with how we can test them to see how well they might perform.**

### Before we get started, we need to download all of the data we'll be using

**sentiment140-subset.csv: cleaned subset of Sentiment140 data - half a million tweets marked as positive or negative.**

**Training on tweets**

Let's say we were going to analyze the sentiment of tweets. If we had a list of tweets that were scored positive vs. negative, we could see which words are usually associated with positive scores and which are usually associated with negative scores.

Luckily, we have Sentiment140 - <http://help.sentiment140.com/for-students> - a list of 1.6 million tweets along with a score as to whether they're negative or positive. We'll use it to build our own machine learning algorithm to see separate positivity from negativity.

**Train our algorithm:**

**For this,we Vectorize our tweets using a TfidfVectorizer**

When we teach our algorithm about what a positive or a negative tweet looks like, this is called training. Training can take different amounts of time based on what kind of algorithm you are using.

Here we are using 5 different algorithms,namely-

* LinearRegression
* LogisticRegression
* RandomForestClassifier
* LinearSVC (Support vector machine)
* Naive\_Bayes

A difference between the two algorithms,i.e., LinearRegression and LogisticRegression

is that for logistic regression, we can also ask for the probability that the sentence is in the 1 category instead of just simply the category.

Now that we've trained our models, they can try to predict whether some content is positive or negative.

### Preparing the data

**Add a few more sentences below.** They should be a mix of positive and negative. They can be boring, they can be exciting, they can be short, they can be long.

First we need to **vectorizer** our sentences into numbers, so the algorithm can understand them.

### Predicting with our models

To make a prediction for each of our sentences, you can use .predict with each of our models.

## Testing our models

We can actually see **which model performs the best!** As we trained our models on tweets, we can ask each model about each tweet, and see if it gets the right answer.

Our original dataframe is a list of many, many tweets. We turned this into X - vectorized words - and y - whether the tweet is negative or positive.

Before we used .fit(X, y) to train on all of our data. Instead, **we can test our models** by doing a test/train split and see if the predictions match the actual labels.

### Confusion matrices

To see how well they did, we'll use a [**"confusion matrix"**](https://en.wikipedia.org/wiki/Confusion_matrix) for each one.

**To Summarize:**

Sentiment140 is a database of tweets that come pre-labeled with positive or negative sentiment, assigned automatically by presence of a :) or :(. Our first step was using a **vectorizer** to convert the tweets into numbers a computer could understand.

After that, we build four different **models** using different machine learning algorithms. Each one was fed a list of each tweet's **features** - the words - and each tweet's **label** - the sentiment - in the hopes that later it could predict labels if given a new tweets. This process of teaching the algorithm is called **training**.

In order to test our algorithms, we split our data into sections - **train** and **test** datasts. You teach the algorithm with the first group, and then ask it for predictions on the second set. You can then compare its predictions to the right answers using a **confusion matrix**.

Although **different algorithms took different amounts of time to train**, they all ended up with about 70-75% accuracy.