# Capstone Project 2
# "Sentiment analysis of Tweets"

# Milestone Report

## Problem Statement

*What is a Sentiment Analysis and why is it useful?*
Sentiment Analysis represents the use of Natural Language Processing to determine the attitude, opinions and emotions of a speaker, writer or other subject within an online mention. In other words, it is the process of determining whether a piece of writing is positive or negative. A human is able to recognize and classify a text into positive or negative. However, a computer not but can learn to do so.

*On which topic?*
I would like to perform a sentiment analysis on Tweets about the late blockbuster release: *Joker*, with Joaquin Phoenix. Out in the USA beginning of October 2019 and throughout theaters in the world later on, no doubt that *Joker* has divided its audience. Whether people love or hate it, the reactions were numerous. I saw it myself at the movies in Switzerland as soon as it came out, and felt exactly this way: divided. The movie received a Golden Lion win at the Venice Film Festival, I had high expectations. The complex personality of the Joker was one of the reasons I wanted to see the movie. But after visioning it, I realized it had a strong impact on me, more than expected and not only positive. The critics I read afterwards were also two-folds, both positive and negative. Therefore, I thought it would be interesting to train a classification model on Tweets on the topic of *Joker*.

*Who is interested?*
Different types of people could be interested to know the proportion of positive vs negative tweets on this topic: fans of Joaquin Phoenix and of *The Dark Knight Rises*, owners of movies, producers, script writers, psychologists and psychiatrics.

## Methodology

I will be using the Twitter API to collect a Test set based on keywords. A function will return a list of tweets that contain our keywords selected. Each tweet's text will see itself attributed a label ('positive' or 'negative') to classify each tweet as positive or negative. The Training set will be downloaded because it has to be labelled into 'positive' or 'negative' on a big amount of tweets. The Training set is critical to the success of the model since our model will "learn" how to do create a sentiment analysis based on the Training set.

The steps that we will follow to perform the sentiment analysis are:

- Section A: Preparing the Test set
  - Step A.1: Getting the authentication credentials
  - Step A.2: Authenticating our python script
  - Step A.3: Creating the function to build the Test set
- Section B: Preparing the Training set
- Section C: Pre-processing Tweets in the Data Sets (both Test and Training)

- Section D: Naive Bayes Classifier
  - Step D.1: Build a vocabulary/list of words in our training data set
  - Step D.2: Match tweet content against our vocabulary
  - Step D.3: Build our word feature vector
  - Step D.4: Training the classifier
- Section E: Testing the model
- Section F: Analyzing the performance of the model
  - Step F.1: Concatenate pre-processed Tweets and labels
  - Step F.2: Check 10 pre-processed Tweets and their attributed label
  - Step F.3: Model performance
  - Step F.4: Wrap up observations
  - Step F.5: Possible improvements

## Section A: Preparing the Test set

### Step A.1: Getting the authentication credentials

To do so, I visited the Twitter Developer website, applied for an API access and waited for Twitter to accept. After getting the approval email, I went back to the website and clicked on "Create an app". I filled in the information asked and finally got Consumer API keys, as well as Access token & access token secret.

### Step A.2: Authenticating our python script

I then imported the twitter library, and created a twitter.Api object containing the credentials mentioned previously.

### Step A.3: Creating the function to build the Test set

I build a function that takes a search keyword as in input, searches for tweets that include this keyword and returns them as twitter.Status objects that we can iterate through. The problem is that Twitter limits the number of requests we can make through the API for security purposes. This limit is 180 requests per 15-minute window. For the sake of simplicity, we will limit the search to 100 tweets for now, not exceeding the allowed number of requests. The function name is **buildTestSet** and will return a list of tweets that contain our search keyword.
Every tweet's text is associated with a label that is NULL for now. The reason for this is that we are going to classify each tweet as Positive or Negative later on, in order to determine whether the sentiment on the search term is positive or negative, based on the majority count.

## Section B: Preparing the Training set

As suggested by examples of Sentiment Analysis, the training set is critical to the success of the model. Data needs to be labeled properly with no inconsistencies or incompleteness, as training will rely heavily on the accuracy of such data and the manner of acquisition.
Therefore, I will be using a downloadable training set, and the tweets contained in it were all labeled as positive or negative depending on the content. This is exactly what a training set is for.

The training set used comes from Niek Sanders' Corpus of over 5000 hand-classified tweets, which makes it quite reliable. The corpus includes a keyword (topic of the tweet), a label and a tweet ID number of every tweet.
The file was obtained here: https://github.com/karanluthra/twitter-sentiment-training/blob/master/corpus.csv

```
tr_set.head()
```

| | apple | positive | 126415614616154112 |
|---|---|---|---|
| 0 | apple | positive | 126404574230740992 |
| 1 | apple | positive | 126402758403305474 |
| 2 | apple | positive | 126397179614068736 |
| 3 | apple | positive | 126395626979196928 |
| 4 | apple | positive | 126394830791254016 |

Characteristics of the Corpus:

- 5512 lines
- 3 columns
- Column Topic contains 4 topics

```
: tr_set[tr_set.columns[0]].unique()

: array(['apple', 'google', 'microsoft', 'twitter'], dtype=
  object)
```

- Column Label contains 4 labels

```
tr_set[tr_set.columns[1]].unique()

array(['positive', 'negative', 'neutral', 'irrelevant'],
dtype=object)
```

I will be using the API to get the actual tweet text through each tweet's ID number included in the Corpus we have. Since Twitter API has a limit mentioned above, I will make the code rest for 15 minute to download 5000 tweets.
The function to do so does the following:

- two inputs: "corpusFile" – a string path to the Niek Sanders' corpus file I downloaded (containing tweet's **topic**, **label** and **id**), and "tweetDataFile" – a string path to the file we would like to save the full tweets in (containing tweet's **text**, **topic**, **label** and **id**)
- started with empty list *corpus*
- then opened file "corpusFile" and appended every tweet from the file to the list *corpus*
- get the text of tweets based on the ID
- loop through the tweets in *corpus* calling the API on every tweet to get the Tweet.Status object of the particular tweet.

- use status object to get the text associated with it and push it into the "trainingDataSet"
- sleep for a couple of minutes to avoid the API limit mentioned

The tweets retrieved through the API are saved in a pickle file "trainingData".

```
In [13]: trainingData

Out[13]: [{'tweet_id': '126415614616154112',
           'label': 'positive',
           'topic': 'apple',
           'text': 'Now all @Apple has to do is get swype on the i
         phone and it will be crack. Iphone that is'},
          {'tweet_id': '126402758403305474',
           'label': 'positive',
           'topic': 'apple',
           'text': "Hilarious @youtube video - guy does a duet wit
         h @apple 's Siri. Pretty much sums up the love affair! ht
         tp://t.co/8ExbnQjY"},
          {'tweet_id': '126397179614068736',
           'label': 'positive',
           'topic': 'apple',
           'text': '@RIM you made it too easy for me to switch to
         @Apple iPhone. See ya!'},
```

## Section C: Pre-processing Tweets in the Data Sets

Before moving on to the classification section, there is some cleaning up to do. For a Sentiment Analysis, words are the most important input used. However, punctuation, images, videos, URLs, usernames, emojis do not contribute to the analysis of tweets, and will have to be removed. This data cleaning step will be applied on both the Training set and the Test set.
A word about the importance of normalizing/pre-processing. Normalization in the NLP context is the process of converting a list of words to a more uniform sequence. By transforming the words into a standard format, later operations can be done on the data without compromising the process. Many pre-processing steps can be taken including: lowercasing, stemming (example: troubled, troubles go into 'troubl'), lemmatization (example: troubled, troubles to into 'trouble'), stopword removal, normalizing, noise removal...

Creating the function to perform data cleaning includes the following steps:
- import necessary librairies:
  - *re*, to parse strings
  - *ntlk*, most commonly used Python library, Natural Processing Toolkit
- create a function, **processTweets**, that loops through all the tweets input into it, calling the function **processTweet** on every tweet in the list
- **processTweet** makes all text in lower-case, removes URLs and usernames, removes '#', gets rid of duplicate characters, tokenizes text into words, and removing "stop words" (useless words such as 'the', 'a', 'in').

Once the function is created, we can apply it on our Training set ("trainingData") and Test set ("testDataSet").

```
In [18]: print(preprocessedTrainingSet[0:4])

[(['get', 'swype', 'iphone', 'crack', 'iphone'], 'positiv
e'), (['hilarious', 'video', 'guy', 'duet', "'s", 'siri',
'pretty', 'much', 'sums', 'love', 'affair'], 'positive'),
(['made', 'easy', 'switch', 'iphone', 'see', 'ya'], 'posi
tive'), (['16', 'strangest', 'things', 'siri', 'said', 'f
ar', 'sooo', 'glad', 'gave', 'siri', 'sense', 'humor', 'v
ia'], 'positive')]

In [36]: print(preprocessedTestSet[0:4])

[(['finally', 'saw', 'joker', 'technical', 'aspects', 'mo
vie', 'stunning', 'acting', 'incredible', 'migh…'], None)
, (['``', 'joker', "''", 'became', 'first', 'r-rated', 'm
ovie', 'hit', '1', 'billion', 'box-office', 'are\u200b',
'rumors', 'sequel'], None), (['would', 'like', 'see', 'se
quel', 'joaquin', 'phoenix', "'s", 'joker'], None), (['rt
', 'done', 'movies', 'bts', 'bring', 'soul', 'movie', 'we
athering', 'joker', 'spiderman', 'far', 'from…'], None)]
```

The data is now ready, we can proceed to building our model.

## Section D: Naive Bayes Classifier

Naïve Bayes Classifier is a classification algorithm that relies on Bayes' Theorem, which provides a way of calculating a type or probability called posterior probability, in which the probability of an event A occurring is reliant on probabilistic know background. For example, if person_X only plays tennis when it is not raining outside, then according to Bayesian statistics, the probability of person_X playing tennis when it is not raining can be given as:

`P(X plays | no rain) = P(no rain | X plays)*P(x plays)/P(no rain)`

### Step D.1: Build a vocabulary/list of words in our training data set

A vocabulary in NLP is a list of all speech segments available for the model. In our case, this includes all the words present in the Training set. We create a list of distinct words, with its frequency (number of occurrences in the set) as a key, "word_features".

### Step D.2: Match tweet content against our vocabulary

I will go through all the words in the Training set, i.e. "word_features", comparing every word against the tweet at hand, associating a number with the word:

- 1 (true): if word in vocabulary is present in tweet
- 0 (false): if word in vocabulary is not resident in tweet

Every word in the vocabulary "word-features", will have a key "contains word X", where X is the word, and associated value "True/False" according to if word in vocabulary is present in tweet or not.

### Step D.3: Build our word feature vector

Calling the last two function created, we will build our final feature vector, called "trainingFeatures", with which we can proceed on to training.

```
In [40]: trainingFeatures

Out[40]: [({'contains(get)': True, 'contains(swype)': True, 'conta
         ins(iphone)': True, 'contains(crack)': True, 'contains(hi
         larious)': False, 'contains(video)': False, 'contains(guy
         )': False, 'contains(duet)': False, "contains('s)": False
         , 'contains(siri)': False, 'contains(pretty)': False, 'co
         ntains(much)': False, 'contains(sums)': False, 'contains(
         love)': False, 'contains(affair)': False, 'contains(made)
         ': False, 'contains(easy)': False, 'contains(switch)': Fa
         lse, 'contains(see)': False, 'contains(ya)': False, 'cont
         ains(16)': False, 'contains(strangest)': False, 'contains
         (things)': False, 'contains(said)': False, 'contains(far)
         ': False, 'contains(sooo)': False, 'contains(glad)': Fals
         e, 'contains(gave)': False, 'contains(sense)': False, 'co
         ntains(humor)': False, 'contains(via)': False, 'contains(
         great)': False, 'contains(close)': False, 'contains(perso
         nal)': False, 'contains(event)': False, 'contains(tonight
         )': False, 'contains(regent)': False, 'contains(st)': Fal
         se, 'contains(store)': False, 'contains(companies)': Fals
         e, 'contains(experience)': False, 'contains(best)': False
```

### Step D.4: Training the classifier

Thanks to the library *nltk* it will take us a short time to train the model as a Naïve Bayes Classifier.
We will perform a small test of our classifier by creating fake sequences of words that are particularly positive and negative to check that the classifier is efficient.

```
test_tweet = ['rt','hilarious', 'video', 'guy', 'duet', "'s
print(NBayesClassifier.classify(extract_features(test_tweet

positive
```

```
test_tweet = ['terrible','bad', 'awful', 'negative', 'duet'
print(NBayesClassifier.classify(extract_features(test_tweet

negative
```

## Section E: Testing the model

Now, I will run the classifier on the Test set (100 tweets), getting the labels (pos/neg) and calculating the positive or negative percentage of the tweets. That's it!

```
Overall Negative Sentiment
Negative Sentiment Percentage = 52.0%
```

**Observation:** the results of the sentiment analysis on Tweets about the movie Joker with Joaquin Phoenix that came out in October 2019 is that there is the overall sentiment is negative at 52%. These classification results coincide with the critics one can read in the newspaper and on the internet, as they are balanced between positive and negative.

## Section F: Analyzing the performance of the model

Let's have a look at a couple of Tweets and the assigned label to check if the classification is intuitive or not. This is a first approximate idea of our classifier's performance.

### Step F.1: Concatenate pre-processed Tweets and labels

**preprocessedTestSet (100, 2)**

| | Tweets pre-processed | label |
|---|---|---|
| 0 | [finally, saw, joker, technical, aspects, movi... | None |
| 1 | [`, joker, '', became, first, r-rated, movie,... | None |
| 2 | [would, like, see, sequel, joaquin, phoenix, '... | None |
| 3 | [rt, done, movies, bts, bring, soul, movie, we... | None |
| 4 | [ashishchanchalani, ashishchanchalanimemes, ol... | None |
| 5 | [think, joker, new, favorite, character/movie,... | None |
| 6 | [rt, still, thinking, sana, harley, quinn, wan... | None |
| 7 | [joker, action, movie] | None |
| 8 | [saw, ford, v, ferrari, last, weekend, great, ... | None |
| 9 | [', dope, joker, movie] | None |

**NBResultLabels (100,1)**

| | label |
|---|---|
| 0 | positive |
| 1 | negative |
| 2 | negative |
| 3 | positive |
| 4 | negative |
| 5 | positive |
| 6 | negative |
| 7 | positive |
| 8 | positive |
| 9 | positive |

```
test_labels
```

```
array([[list(['finally', 'saw', 'joker', 'technical', 'as
pects', 'movie', 'stunning', 'acting', 'incredible', 'mig
h…']),
        None, 'positive'],
       [list(['``', 'joker', "''", 'became', 'first', 'r-
rated', 'movie', 'hit', '1', 'billion', 'box-office', 'ar
e\u200b', 'rumors', 'sequel']),
        None, 'negative'],
       [list(['would', 'like', 'see', 'sequel', 'joaquin'
, 'phoenix', "'s", 'joker']),
        None, 'negative'],
       [list(['rt', 'done', 'movies', 'bts', 'bring', 'so
ul', 'movie', 'weathering', 'joker', 'spiderman', 'far',
'from…']),
        None, 'positive'],
       [list(['ashishchanchalani', 'ashishchanchalanimeme
s', 'oldvideos', 'joker', 'think', 'downloaded', 'wrong',
'joker', 'movie']),
        None, 'negative'],
```

## Step F.2: Check 10 pre-processed Tweets and their attributed label

```
# Check first 10 Tweets words and their attributed labels
test_labels[0]
```

```
array([list(['finally', 'saw', 'joker', 'technical', 'asp
ects', 'movie', 'stunning', 'acting', 'incredible', 'migh
…']),
       None, 'positive'], dtype=object)
```

```
test_labels[1]
```

```
array([list(['``', 'joker', "''", 'became', 'first', 'r-r
ated', 'movie', 'hit', '1', 'billion', 'box-office', 'are
\u200b', 'rumors', 'sequel']),
       None, 'negative'], dtype=object)
```

```
test_labels[2]
```

```
array([list(['would', 'like', 'see', 'sequel', 'joaquin',
'phoenix', "'s", 'joker']),
       None, 'negative'], dtype=object)
```

```
test_labels[3]
```

```
array([list(['rt', 'done', 'movies', 'bts', 'bring', 'sou
l', 'movie', 'weathering', 'joker', 'spiderman', 'far', '
from…']),
       None, 'positive'], dtype=object)
```

```
test_labels[4]
```

```
array([list(['ashishchanchalani', 'ashishchanchalanimemes
', 'oldvideos', 'joker', 'think', 'downloaded', 'wrong',
'joker', 'movie']),
       None, 'negative'], dtype=object)
```

```
test_labels[5]
```

```
array([list(['think', 'joker', 'new', 'favorite', 'charac
ter/movie', 'love', 'every', 'actor', '‘', 'played', 'get
', 'new', 'parts', 'him…']),
       None, 'positive'], dtype=object)
```

```
test_labels[6]
```

```
array([list(['rt', 'still', 'thinking', 'sana', 'harley',
'quinn', 'wanted', 'chaeyoung', 'joker', 'coz', 'chaeyoun
g', 'love', 'movie…']),
       None, 'negative'], dtype=object)
```

```
test_labels[7]
```

```
array([list(['joker', 'action', 'movie']), None, 'positiv
e'], dtype=object)
```

```
test_labels[8]
```

```
array([list(['saw', 'ford', 'v', 'ferrari', 'last', 'week
end', 'great', 'movie', 'would', 'top', 'list', '2019', "
'm", 'marvel', 'guy', 'avengers…']),
       None, 'positive'], dtype=object)
```

```
test_labels[9]
```

```
array([list(['‘', 'dope', 'joker', 'movie']), None, 'posi
tive'],
      dtype=object)
```

## Step F.3: Model performance

Tweet[0]: At a glance, we see words such as 'stunning', 'incredible' that are positive. The label is therefore 'positive' as expected. CORRECT
Tweet[1]: This tweet seems rather neutral, but is classified as 'negative'. BIAS?
Tweet[2]: This tweet seems rather neutral, but is classified as 'negative'. BIAS?
Tweet[3]: This tweet seems rather neutral, but is classified as 'positive'. BIAS?
Tweet[4]: This tweet seems rather neutral, but is classified as 'negative'. BIAS?
Tweet[5]: This tweet seems rather positive with the word 'love', 'favorite'. The label is therefore 'positive' as expected. CORRECT
Tweet[6]: This tweet seems rather neutral, with a small positive touch with the word 'love'. However, the label is 'negative'. WRONG
Tweet[7]: This tweet seems rather neutral, but is classified as 'negative'. BIAS?
Tweet[8]: At a glance, we see words such as 'great', 'top' that are positive. The label is therefore 'positive' as expected. CORRECT
Tweet[9]: This tweet seems rather neutral, with a small positive note 'dope'. The label is therefore 'positive' as expected. CORRECT

## Step F.4: Wrap up observations

1. **50%** | It seems like neutral Tweets are more often classified as negative (4/5) which is a consequence of simplifying the classification options to two (positive and negative). More rarely (1/5), a neutral Tweet is classified as positive.
2. **40%** | Correct classification of the Tweet
3. **10%** | Wrong classification of the Tweet

This (subjective) performance evaluation task is telling us that the classification model is not optimal yet.

### Step F.5: Possible improvements

- Trying different combinations of pre-processing steps
- Adding some pre-processing steps
- Checking the impact of each pre-processing step on model performance
- Building our own Training Dataset, to control quality of the classification in the Training Dataset, as it is crucial to a good model performance.

## Conclusion

Sentiment Analysis is an interesting way to think about the applicability of Natural Language Processing in making automated conclusions about text. It is being utilized in social media trend analysis and, sometimes, for marketing purposes. Making a Sentiment Analysis program in Python is not a difficult task, thanks to modern-day, ready-for-use libraries.

With this project, we wanted to analyze Tweets about the late released movie "Joker" that attracted all kinds of comments, critics on the web. In summary, we have

- created a Test set directly from Twitter thanks to the API
- used an existing Training set that was classified into positive and negative tweets according to its content by hand
- cleaned both sets by removing any signs (punctuation, hashtags...) that do not bring anything to the sentiment analysis
- created a vocabulary of words based on the Training Set
- matched tweets against vocabulary
- trained the classifier
- test the model on the test Set

The result of the sentiment analysis is that almost half of the Tweets in the Test set are classified as negative and the other half as positive. The results show the divided critics about the movie, which is a feeling I had when scanning through articles on the web.

The result of the sentiment analysis is that almost half of the Tweets in the Test set are classified as negative and the other half as positive. The results show the divided critics about the movie, which is a feeling I had when scanning through articles on the web.

However, the quality of our model seems compromised, and several areas of improvement have been identified above.

I would not use the model per se, but most probably after implementing the different changes mentioned above.

Another area of improvement to the model is in regards to measuring the performance of our model. The way the project was conducted didn't allow to use a solid model performance tool such as a confusion matrix. What could be done to measure better the model's performance would be to:

- separate the Training Tweets Dataset between Train and Test sets
- develop and train the model on the Train set
- use Test set to test model
- run metrics (confusion matrix) to evaluate model's performance over Test set and Train set