Detection of Offensive Tweets

This project is to apply Natural Language Processing in data obtained from the web. In this case, we will do web scrapping to Twitter to get the data. We will modify them and obtain a cleaned data, to finally make some predictive models to know if a specific tweet is an insult or not.

First of all, we need to connect to connect to Twitter to maniputale the data. Tweepy is a package that makes it easier to use the twitter streaming api by handling authentication, make the connection and then we will be able to extract the data we want.

In [30]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
import tweepy
from tweepy import OAuthHandler

consumer_key = 'XXXXXXXXXXXXXXXXXXXXXXXX'
consumer_secret = 'XXXXXXXXXXXXXXXXXXXXXXX'
access_token = 'XXXXXXXXXXXXXXXXXXXXXXXXXX'
access_secret = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXX'

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)

api = tweepy.API(auth)
```

In [2]:

```
for status in tweepy.Cursor(api.home_timeline).items(10):
    # Process a single status
    print(status.text)
```

```
#EnVivo Aula Magna XXII: Inteligencia Artificial, Big Data, Machine Learning y Robótica.
#AulaMagnaPUCP https://t.co/Z0L15SvNle
RT @Rudgrcom: Birthdays are best with friends, and cake. Lots of cake. Happy celebrating
@davidguetta! □□□ https://t.co/ePJZq4a1ih https://…
Súbele a tu miércoles con buena música □□ crea tu Playlist favorito en @spotify con todos mis tem
as más recientes c… https://t.co/OlTRI3FuHu
RT @NRJhitmusiconly: Les patrons sont au rendez vous !!!! 😀
David GUETTA et  Martin GARRIX raflent tout sur leur passage.
Ils remportent le…
J'adore cette journée □□□
https://t.co/mxew8SWh1s
RT @martinsolveig: Happy birthday David ✌□ https://t.co/RkR4m4wXJO https://t.co/flI9SFntr4
RT @KungsMusic: Happy birthday boss @davidguetta and congrats for this amazing year. True legend □
□□ https://t.co/f9HuzR2oyz
"Adiós a la incapacidad civil de las personas con discapacidad mental". Escribe Renata Bregaglio,
profesora del Dep… https://t.co/eEJncFR7Aq
I just can’t wait! □□□□ □□□ Love U bae! @jo__lyn ❤□ https://t.co/Ze9r72V56H
I just can’t wait! □□□□ □□□ Love U bae! jo__lyn ❤□ https://t.co/en8SK4zwkX
```

In [ ]:

```

```

We will skip the part of obtaining the different tweets. Also, we have to decide if each tweet in our dataset is an insult or not. We need to define it using a flag, this can be made creating a new column in our dataset.

We will use nltk to make the cleaning process of our data. First, we will use tokenization to separate each word. After that, After that, we will make shorter our words. Between lemmatization and stemming, we prefer to use stemming because it is simpler, smaller and usually faster. Then, we will get rid of the words that are not important, such as “the”, “a”, “an”, “in”.

In [11]:

```
import nltk
from nltk.tokenize import PunktSentenceTokenizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import sent_tokenize, word_tokenize
import re

lemmatizer = WordNetLemmatizer()   #to lemmatize
ps = PorterStemmer()     #stemming
tokenizer = nltk.tokenize.punkt.PunktSentenceTokenizer()
stop_words = set(stopwords.words('english'))
```

After creating the instances, we will make use of them. We will clean all tweets making a for-loop. Have in mind we will make use of regular expressions, which are special sequences of characters that help to match or find other strings or sets of strings. In this case, this will help us to eliminate some characters we find useless.

In [21]:

```
features=[]
comments=[]
num=0

for comment in tweets:
    word_tokens=word_tokenize(comment)   #tokenizing the comment
    filtered_sentence = []

    for w in word_tokens:           #each word of the comment
        w1=ps.stem(w)                   #stemming the word
        #w1=lemmatizer.lemmatize(w1)    #its not necessary lemmatizing

        if w1 not in stop_words:        #if its not a stop words, enter to the if-statement
            w2 = re.sub(r'(\\xa0)', r' ', w1)      #erasing the emoticons, or dirty characters
            w3 = re.sub(r'(\xa0)', r' ', w2)
            w3=w3.replace(u'\\n', u' ')
            w3=w3.replace(u'\\xc2', u' ')
            w3=w3.replace(u'\\u', u' ')
            w3=w3.replace(u'\\x', u' ')
            w4 = re.sub(r'(\xc2)', r' ', w3)
            w5 = re.sub(r'[^s](\n)', r' ', w4)
            w6 = re.sub(r'(\n)', r' ', w5)
            w7 = re.sub(r'(\xec)', r' ', w6)
            w8 = re.sub(r'($)', r' ', w7)             #erasing dollars
            w9 = re.sub(r'[^\s]+@[^\s]+', r'', w8)      #erasing emails
            w10 = re.sub(r'@[^\s]+', r' ', w9)            #erasing userId of people i.e. @alex01
            w11 = re.sub(r'(http|https)://[^\s]*', r'', w10)     #erasing websites
            w11=w11.replace(u'\\', u' ')
            word1 = re.sub(r'[0-9]+', r'', w11)         #erasing numbers


            word_tokens_Aux=word_tokenize(word1)

            if len(word_tokens_Aux)!=1:
                for w_Aux in word_tokens_Aux:
                    w1_Aux=ps.stem(w_Aux)
                    if w1_Aux not in stop_words:
                        filtered_sentence.append(w1_Aux.lower())
                        features.append(w1_Aux.lower())
            else:
                filtered_sentence.append(word_tokens_Aux[0])
                features.append(word_tokens_Aux[0])


    comments.append( [filtered_sentence, dataset.iloc[num,0]] )
    num=num+1

comments[1]
```

Out[21]:

```
[['``',
  'realli',
  "n't",
  'understand',
  'point',
  '.',
  'It',
  'seem',
  'mix',
  'appl',
  'orang',
  '.',
  "''"],
 0]
```

Above we can see one tweet after the cleaning process. It looks pretty good in comparison how we found it at first. Now, we will use only the first 10,000 words to do the predictive analysis

In [22]:

```python
all_words = nltk.FreqDist(features)
main_words=all_words.most_common(10000)
word_features=[]
for w in main_words:
    w_list=list(w)
    word_features.append(w[0])
#word_features

word_features = list(all_words.keys())[:10]
#word_features
```

We need to create our defined format to put in our machine learning model. In this case, our table containing all the data will be defined in featuresets.

In [23]:

```python
featuresets=[]

for comment1 in comments:
    features_Aux={}
    for w in word_features:

        features_Aux[w]=w in comment1[0]
    featuresets.append((features_Aux,comment1[1]))
```

After that, we will split our dataset in training set and test set. After that, we will use some machine learning models (such as Naive Bayes, Bernoulli, Logistic Regression, SGD, SVC) to classify the tweets. Finally, we will score our models with the accuracy.

In [31]:

```python
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.naive_bayes import MultinomialNB,BernoulliNB
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.svm import SVC
training_set = featuresets[:2000]
testing_set =  featuresets[2000:]

classifier = nltk.NaiveBayesClassifier.train(training_set)
print("Original Naive Bayes Algo accuracy percent:", (nltk.classify.accuracy(classifier,
testing_set))*100)

BernoulliNB_classifier = SklearnClassifier(BernoulliNB())
BernoulliNB_classifier.train(training_set)
print("BernoulliNB_classifier accuracy percent:", (nltk.classify.accuracy(BernoulliNB_classifier,
testing_set))*100)

LogisticRegression_classifier = SklearnClassifier(LogisticRegression())
LogisticRegression_classifier.train(training_set)
print("LogisticRegression_classifier accuracy percent:",
(nltk.classify.accuracy(LogisticRegression_classifier, testing_set))*100)
```

```
(nltk.classify.accuracy(LogisticRegression_classifier, testing_set))*100)

SGDClassifier_classifier = SklearnClassifier(SGDClassifier())
SGDClassifier_classifier.train(training_set)
print("SGDClassifier_classifier accuracy percent:",
(nltk.classify.accuracy(SGDClassifier_classifier, testing_set))*100)

SVC_classifier = SklearnClassifier(SVC())
SVC_classifier.train(training_set)
print("SVC_classifier accuracy percent:", (nltk.classify.accuracy(SVC_classifier,
testing_set))*100)
```

```
Original Naive Bayes Algo accuracy percent: 72.26502311248075
BernoulliNB_classifier accuracy percent: 72.26502311248075
LogisticRegression_classifier accuracy percent: 72.26502311248075
SGDClassifier_classifier accuracy percent: 72.31638418079096
SVC_classifier accuracy percent: 72.47046738572163
```

SVC Classifier has the greatest accuracy, that's why we choose this model to go forward and deploy our model.