Author: **M. Kolaksazov**

SOCIAL MEDIA SENTIMENT

A task for analyzing the sentiment of the messages in the social media. Analyzis was carried out, based on the content of words in the individual messages. The probability of occurence of words was calculated, followed by the calculation of the pointwise mutual information, or PMI, giving information about how probable is to find inside the text a combination of two words together. After that, the PMI of words from the text, as well as special words from vocabularies was calculated to find out how "positive" or "negative" the message is.

For this purpouse, data from the social media "Twitter" was used, as well as from the Internet Movie Data Base (IMDB).

In [2]:
```
from textblob import TextBlob
import numpy as np
import pandas as pd
import re
import json
import operator
from collections import Counter
from nltk.corpus import stopwords
import string
from collections import defaultdict
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from numpy import *
from sklearn.cluster import SpectralClustering
from sklearn.cluster import DBSCAN
```

Importing data from Twitter:

In [3]:
```
class ReadWriteJson():
    def __init__(self, json_file = None):
        self.json_file = json_file

    def read_from_json(self, json_file):
        self.json_file = json_file
        with open(json_file) as f:
            data_json = json.load(f)
        data = pd.read_json(data_json, typ='series')
        json_to_dataframe = pd.DataFrame(data = data['data'], index = data['index']
, columns = data['columns'])
        return json_to_dataframe
```

Text pre-processing:

Tokenization, cleaning up the tweets from special symbols, hashtags, @-mentions, short words and stop words.

There was also a Python algorithm, called sentiment analyzis, which can sort the messages on the basis of their positive or negative sentiment.

It was necessary to remove all of special symbols, but to preserve the symbols, specific for the language, such as accents, umlaut, etc. In addition, the own personal names, or names of places, such as "Los Angelis", "San Francisco", that were more likely to be found together were necessary to be removed, as well.

In [4]:
```
class TweetPreprocess():
    def clean_tweet(self, tweet):
        tweet = str(tweet)
        return ' '.join(re.sub('([…])|([@#][\w]*)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)',
' ', tweet).split())

    def remove_stop_words(self, corpus):
        removed_stop_words = []
        for text in corpus:
            removed_stop_words.append(' '.join([word.lower() for word in text.split
()
                                                if word not in stopwords.words('eng
lish')
                                                and not word.startswith('@') # excl
ude @-mentions
                                                and not word.startswith('#') # excl
ude hashtags
                                                and not word.startswith('&') # excl
ude special symbols
                                                and not word[0].isupper() # exclude
personal names
                                                and len(word) > 2])) # exclude very
short words
        return removed_stop_words

    def analyze_sentiment(self, tweet):
        analysis = TextBlob(self.clean_tweet(tweet))
        if analysis.sentiment.polarity > 0:
            return 1
        elif analysis.sentiment.polarity == 0:
            return 0
        else:
            return -1

    # extracts the data from tweets in columns
    def tweets_to_data_frame(self, tweets):
        df = pd.DataFrame(data = [tweet.text for tweet in tweets], columns = ['twee
ts'])
        df['sentiment'] = np.array([self.analyze_sentiment(tweet) for tweet in df['
tweets']])
        return df
```

Analyzis of the text by the means of the calculation of pointwise mutual information (PMI):

$$\mathrm{PMI}(t_1, t_2) = \log\left(\frac{P(t_1 \wedge t_2)}{P(t_1) \cdot P(t_2)}\right)$$

It is based on the probability for the individual occurence of two words in text, as well as the mutual co-occurence of the two words.

For this purpouse, a matrix of co-occurence, containing all possible combinations of words was created. After that, probability and the PMI were calculated.

To sort the words by their semantic orientation, the PMI of a word and a word from positive and negative vocabularies was calculated. Only two words in each dictionary were used ("good", "bad", and "happy" and "sad"), in order to decrease the subjective criteria for differentiation.

Difficulties for implementation of this algorithm were related to the use of different types of data (e.g.: lists, arrays, dictionaries and "defaultdict"). A bug, that was very difficult to eliminate was the mismatch of keys in the different dictionaries. After attention was given to set the dictionary keys, in order to originate from the same source, the bug was removed.

In [14]:
```
class TweetAnalyzer():
    def co_occurrence(self, tweets, num_words=5):
        co_occur = defaultdict(lambda : defaultdict(int))
        for tweet in tweets:
            tweet = tweet.split()
            # Build co-occurrence matrix
            for i in range(len(tweet)-1):
                for j in range(i+1, len(tweet)):
                    word_1, word_2 = sorted([tweet[i], tweet[j]])
                    if word_1 != word_2:
                        co_occur[word_1][word_2] += 1
        return co_occur

    def probability_calculate(self, text, co_occur):
        p_t = {}
        p_t_com = defaultdict(lambda : defaultdict(int))
        count_words = Counter(" ".join(text).split(" "))
        for key_1, n in count_words.items():
            total_amount = len(text)
            p_t[key_1] = n / total_amount
            for key_2 in co_occur[key_1]:
                p_t_com[key_1][key_2] = co_occur[key_1][key_2] / total_amount
        return p_t, p_t_com, count_words
```

In [ ]:
```
    def pmi(self, p_t, p_t_com, co_occur, searched_word=None):
        pmi = defaultdict(lambda : defaultdict(int))
        for key_1 in p_t:
            for key_2 in co_occur[key_1]:
                multiplied_probability = p_t[key_1] * p_t[key_2]
                pmi[key_1][key_2] = np.log2(p_t_com[key_1][key_2] / multiplied_prob
ability)

        positive_vocab = ['happy','good']
            #,'great', 'recommend', 'omg', 'beautiful', 'wow', 'happy', ':)', ':-)'
            #, 'like', 'love', 'nice', 'awesome', 'outstanding'
            #,'fantastic', 'terrific', 'congratulations', 'win']
        negative_vocab = ['sad','bad']
            #,'sad', 'cried', 'terrible', 'crap', 'useless', 'hate', ':(', ':-(']
        semantic_orientation = {}
        for term, n in p_t.items():
            positive_assoc = np.sum([pmi[term][tx] for tx in positive_vocab])
            negative_assoc = np.sum([pmi[term][tx] for tx in negative_vocab])
            semantic_orientation[term] = positive_assoc - negative_assoc
        semantic_sorted = sorted(semantic_orientation.items(),key=operator.itemgett
er(1),reverse=True)
        top_pos = semantic_sorted[:5]
        top_neg = semantic_sorted[-5:]
        return top_pos, top_neg

    def calculate_accuracy(self, labels):
        #changing the index of the predicted by the means of the clustering analysi
s to be equal with
        #the original sentiment analysis
        correct = 0
        for i in range(labels.shape[1]):
            if labels[1,i]== 1:
                labels[1,i]= -1
            elif labels[1,i]== -1:
                labels[1,i]= 1
            elif labels[1,i]== 0:
                labels[1,i]= 0
            if labels[0,i]==labels[1,i]:
                correct+=1
        print("Comparing the original algorithm for sentiment analysis \nand the pr
edicted by the means of the clustering analysis: \n", labels)
        print(correct/(labels.shape[1])*100, " % accuracy")
```

Sorting of messages in positive and negative categories:

It was very difficult to find objective criteria for separation of the messages, because the sorting of messages was highly dependend on the words, that were included inside the vocabularies. Furthermore, words can have different meanings, based on the context. Thus the sentiment of the text messages was analyzed by the means of unsupervised learning (cluster analyzis). Afterwards, a comparison between the built-in algorithm for sentiment analysis and the clustering analysis was carried out and the accuracy was calculated. The accuracy of cluster analyzis, carried out on the data set from the IMDB was higher, because the data was already separated in two (positive and negative) categories with equal sizes.

It was also interesting to perform PMI analyzis on the already separated data by the means of the native Python algorithm. It was found out, that words categorized as positive and negative by PMI were differentiated in the corresponding categories, whereas the so-called "neutral" category lacks both positive and negative words.

In [15]:
```
if __name__ == '__main__':
    read_write_json = ReadWriteJson()
    tweet_analyzer = TweetAnalyzer()
    tweet_preprocess = TweetPreprocess()
    json_file = 'D:\\Marko\\ML\\coding examples\\twitter_data.json'
    df = read_write_json.read_from_json(json_file)
    text = [tweet_preprocess.clean_tweet(line) for line in df['tweets']]
    text = tweet_preprocess.remove_stop_words(text)

    tfidf_vectorizer = TfidfVectorizer()
    tfidf_vectorizer.fit(text)
    reviews_features = tfidf_vectorizer.transform(text)
    labels = df['sentiment']

    #62.5 % on the dataset with from twitter (3 groups)
    k_means = DBSCAN(eps = 0.99, min_samples = 10)
    #86.875 % on the dataset from IMDB (2 groups)
    #k_means = KMeans(n_clusters = 2, init = 'k-means++')

    assigned = k_means.fit_predict(reviews_features)
    labels = np.vstack((labels, assigned))
    tweet_analyzer.calculate_accuracy(labels)

    for i in range(3):
        dat = df[df.sentiment == i-1]
        text = [tweet_preprocess.clean_tweet(line) for line in dat['tweets']]
        text = tweet_preprocess.remove_stop_words(text)
        co_occur = tweet_analyzer.co_occurrence(text)
        p_t, p_t_com, count_words = tweet_analyzer.probability_calculate(text, co_o
ccur)
        top_pos, top_neg = tweet_analyzer.pmi(p_t, p_t_com, co_occur)
        if i == 0:
            print('\n\nNEUTRAL SENTIMENT')
        elif i == 1:
            print('\n\nNEGATIVE SENTIMENT')
        elif i == 2:
            print('\n\nPOSITIVE SENTIMENT')
        print("Top positive words: ", top_pos)
        print("Top negative words: ", top_neg)
```

```
Comparing the original algorithm for sentiment analysis
and the predicted by the means of the clustering analysis:
 [[ 1  0  0 -1  1  1  0  0  1  1  0  1  0  0  1  1  1  0  1  0  0  0 -1  0
   0  0  1  0  0  1  1  0  1  1  1  0 -1  0  0  1  0  0  1  1  0  0  0  0
   1  1  1  1  1  0  0  0  1  0 -1  0  0  0  0  0  0  0  1  0 -1  0  1  0
  -1  1 -1  1  0  0  0  1  1  1  1  1  1  0  0  1  1  0  0  1 -1  1  1  0
   1  1  0  1  1  0  0  1  0  0  1  1  1  1  0  0  1  0  0  0  1  0  0  0
   0  0  0  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1  0  1  1  0  1  1  0  1  1  0  0  0  1  0 -1  1  1  1
   1 -1  1  1  0 -1  1  1]
 [ 1  1  1  1  1  1  1  1  1  1  0  1  1  1  1  1  1  1  1  0  1  1  1  0
   1  1  1  1  0  1  1  1  1  1  1  1  1  1  0  1  1  1  1  1  1  1  0  0
   1  1  1  1  1  0  1  0  1  1  1  0  0  0  0  1  0  0  1  1  1  1  1  1
   1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
   1  1  0  1  0  1  1  1  1  1  0  1  1  1  1  1  1  1  1  1  1  1  1  1
   1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1  1  1  1  1  0  1  1  1  0  1  1  0  1  0  1  1  1  1  0
   1  1  1  1  1  1  1  1]]
63.0  % accuracy


NEUTRAL SENTIMENT
Top positive words:  [('made', 0), ('difficult', 0), ('decision', 0), ('renew',
0), ('fourth', 0)]
Top negative words:  [('far', 0), ('wide', 0), ('190', 0), ('countries', 0), ('g
et', 0)]


NEGATIVE SENTIMENT
Top positive words:  [('thank', 0), ('bringing', 0), ('series', 0), ('back', 0),
('television', 0)]
Top negative words:  [('never', -6.169925001442312), ('done', -6.169925001442312
), ('parts', -6.169925001442312), ('hilarious', -6.169925001442312), ('crushingl
y', -6.169925001442312)]


POSITIVE SENTIMENT
Top positive words:  [('dont', 3.7441610955704103), ('even', 3.7441610955704103)
, ('credit', 3.7441610955704103), ('critical', 3.7441610955704103), ('baby', 3.7
441610955704103)]
Top negative words:  [('overcome', 0), ('hardships', 0), ('thinking', 0), ('pers
on', 0), ('ought', 0)]
```