الغاية من الجلسة:

تطبيق مفهوم logistic regression لتحليل المشاعر ضمن النص.

مقدمة:

في هذه الجلسة سنقوم بتطبيق عدة عمليات وهي:

- Extract features
- Implement logistic regression form secratch
- Apply logistic regression on an NLP task
- Test using logistic regression

في نبدأ مع استدعاء المكتبات والتابع preprocessing المعني بتطبيق أدوات المعالجة المسبقة والتي تعلمناها في الجلسة السابقة.

```
import re
import string
import numpy as np
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import TweetTokenizer
def process_tweet(tweet):
 """Process tweet function.
 Input: tweet: a string containing a tweet
Output: tweets_clean: a list of words containing the processed tweet """
 stemmer = PorterStemmer()
stopwords_english = stopwords.words('english')
# remove stock market tickers like $GE
tweet = re.sub(r'\$\w*', '', tweet)
# remove old style retweet text "RT"
```

```
tweet = re.sub(r'^RT[\s]+', '', tweet)

# remove hyperlinks

tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)

 # remove hashtags # only removing the hash # sign from the word

tweet = re.sub(r'#', '', tweet)

 # tokenize tweets

tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
tweet_tokens = tokenizer.tokenize(tweet)

tweets_clean = []

for word in tweet_tokens:

 if (word not in stopwords_english and # remove stopwords word

not in string.punctuation): # remove punctuation

# tweets_clean.append(word)

 stem_word = stemmer.stem(word) # stemming word


tweets_clean.append(stem_word)

return tweets_clean
```

بعد عملية المعالجة المسبقة نحن بحاجة لاستخراج الميزات من النص "تعتبر هذه الميزات الدخل ذو المعنى والذي يعبر عن العلاقات أو طبيعة النص المدخل الى عملية التصنيف logistic regression

خرج التابع الذي سنستخدمه لاستخراج الميزات من النص هو قاموس يحتوي أزواج تمثل الكلمة وتواجدها في جملة إيجابية أو سلبية. كمثال على ذلك:

{(bad,1):2,(bad,0):20} المقصود هنا أن كلمة bad تكررت في جملة سلبية 20 مرة وفي جملة إيجابية مرتين

عند الاعتماد على كلمة bad من أجل تحليل المشاعر في جملة ما وفقا للتابع المستخدم فإن الميزة هنا تشير الى تصنيف سلبي للجملة.

```
def build_freqs(tweets, ys):

 """Build frequencies.

Input: tweets: a list of tweets ys: an m x 1 array with the sentiment label of each tweet (either 0 or 1)
Output: freqs: a dictionary mapping each (word, sentiment) pair to its frequency """

 # Convert np array to list since zip needs an iterable.

# The squeeze is necessary or the list ends up with one element.
```

```
# Also note that this is just a NOP if ys is already a list.
 yslist = np.squeeze(ys).tolist()
 # Start with an empty dictionary and populate it by looping over all tweets # and over all processed words in each tweet.
 freqs = {}
for y, tweet in zip(yslist, tweets):
 for word in process_tweet(tweet):
 pair = (word, y)
 if pair in freqs:
 freqs[pair] += 1
else: freqs[pair] = 1
 return freqs
```

بالاستفادة من التابعين السابقين يمكننا الان من تطبيق logistic regression لتحليل المشاعر أي بتصنيف الجمل إلى إيجابية وسلبية لذلك حاليا نحن بحاجة الى تعريف تابعين sigmoid وهو تابع خرجه بين 0 و 1 يمثل التصنيف وتابع Gradient descent والذي يمثل عملية تدريب المصنف .

مجموعة البيانات المستخدمة هنا هي مجموعة البيانات ذاتها في الجلسة السابقة.

```
def sigmoid(z):
''' Input: z: is the input (can be a scalar or an array)
 Output: h: the sigmoid of z '''
 ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code)
### # calculate the sigmoid of z
 h = 1 / (1 + np.exp(-z))
return h
```

```
def gradientDescent(x, y, theta, alpha, num_iters):
 ''' Input: x: matrix of features which is (m,n+1)
 y: corresponding labels of the input matrix x, dimensions (m,1)
theta: weight vector of dimension (n+1,1)
alpha: learning rate
num_iters: number of iterations you want to train your model for
Output: J: the final cost theta: your final weight vector Hint: you might want to print the cost to make sure that it is going down. '''
 ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code)
### # get 'm', the number of rows in matrix x
 m = len(x)
for i in range(0, num_iters):
 # get z, the dot product of x and theta
 z = np.dot(x, theta)
 # get the sigmoid of z
h = sigmoid(z)
 # calculate the cost function
 J = (-1 / m) * (np.dot(y.T, np.log(h)) + np.dot((1 - y).T, np.log(1 - h)))
 # update the weights theta
 theta = theta - (alpha / m) * (np.dot(x.T, (h - y)))
 ### END CODE HERE ###
 J = float(J)
return J, theta
```

سيتم الان الاستفادة من التوابع السابقة من أجل تحليل المشاعر في جملة مدخلة عن طريق التابع predict

```
def predict_tweet(tweet, freqs, theta):
    ''' Input: tweet: a string freqs: a dictionary corresponding to the frequencies of each tuple (word, label)
    theta: (3,1) vector of weights
     Output: y_pred: the probability of a tweet being positive or negative '''
     ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code)
     ### # extract the features of the tweet and store it into x
     x = extract_features(tweet, freqs)
    # make the prediction using x and theta
     y_pred = sigmoid(np.dot(x, theta))
     ### END CODE HERE ###
    return y_pred


# Run this cell to test your function
for tweet in ['I am happy', 'I am bad', 'this movie should have been great.', 'great', 'great great', 'great great great',
     print( '%s -> %f' % (tweet, predict_tweet(tweet, freqs, theta)))
```

I am happy -> 0.518562

I am bad -> 0.494329

this movie should have been great. -> 0.515312

great -> 0.515449

great great -> 0.530868