LAB 7

Replace Manual version of Logistic Regression with TF based version.

In [41]:
```
import nltk
from nltk.corpus import twitter_samples
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

In [42]:
```
nltk.download('twitter_samples')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package twitter_samples to
[nltk_data]     /home/nihar/nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
[nltk_data] Downloading package stopwords to /home/nihar/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[42]: True

In [43]:
```
import re
import string
import numpy as np

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import TweetTokenizer
```

In [44]:
```
def process_tweet(tweet):
    stemmer = PorterStemmer()
    stopwords_english = stopwords.words('english')

    tweet = re.sub(r'\$\w*', '', tweet)

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

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

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



    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 word not in string.punctua
tion):
                stem_word=stemmer.stem(word)
                tweets_clean.append(stem_word)

    return tweets_clean
```

In [45]:
```
def build_freqs(tweets, ys):
    yslist = np.squeeze(ys).tolist()
    freqs = {}
    for y, tweet in zip(yslist, tweets):
        for word in process_tweet(tweet):
            pair = (word, y)
            if pair not in freqs:
              freqs[pair]=0
            freqs[pair]+=1
    return freqs
```

In [46]:
```
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

In [47]:
```
data=all_positive_tweets+all_negative_tweets
la=np.append(np.ones((len(all_positive_tweets), 1)), np.zeros((len(all_negat
ive_tweets), 1)), axis=0)
train_x,test_x,train_y,test_y=train_test_split(data,la,test_size=0.30,random
_state=129)
```

In [48]:
```
freqs = build_freqs(train_x,train_y)

print("type(freqs) = " + str(type(freqs)))
print("len(freqs) = " + str(len(freqs.keys())))
```

```
type(freqs) = <class 'dict'>
len(freqs) = 10401
```

In [49]:
```
print('This is an example of a positive tweet: \n', train_x[0])
print('\nThis is an example of the processed version of the tweet: \n', proc
ess_tweet(train_x[0]))
```

```
This is an example of a positive tweet:
 Yeah I screwed up again :-( and this time I thought I did something good go
od

This is an example of the processed version of the tweet:
 ['yeah', 'screw', ':-(', 'time', 'thought', 'someth', 'good', 'good']
```

In [50]:
```
def extract_features(tweet, freqs):
    word_l = process_tweet(tweet)
    x = np.zeros((1, 2))
    for word in word_l:
        if((word,1) in freqs):
          x[0,0]+=freqs[word,1]

        if((word,0) in freqs):
          x[0,1]+=freqs[word,0]

    assert(x.shape == (1, 2))
    return x[0]
```

In [51]:
```
tmp1 = extract_features(train_x[0], freqs)
print(tmp1)
```

```
[486. 698.]
```

In [52]:
```
def predict_tweet(tweet):

    with tf.Session() as sess:
      saver.restore(sess,save_path='TSession')
      data_i=[]
      for t in tweet:
        data_i.append(extract_features(t,freqs))
      data_i=np.asarray(data_i)
      return sess.run(tf.nn.sigmoid(tf.add(tf.matmul(a=data_i,b=W,transpose_
b=True),b)))
    print("Fail")
    return
```

In [53]:
```
b=tf.Variable(np.random.randn(1),name="Bias")
W=tf.Variable(np.random.randn(1,2),name="Bias")
```

In [54]:
```
data=[]
for t in train_x:
  data.append(extract_features(t,freqs))
data=np.asarray(data)
```

In [55]:
```
Y_hat = tf.nn.sigmoid(tf.add(tf.matmul(np.asarray(data), W,transpose_b=True
), b))
print(Y_hat)
ta=np.asarray(train_y)
cost = tf.nn.sigmoid_cross_entropy_with_logits(
                    logits = Y_hat, labels = ta)
print(cost)
```

```
Tensor("Sigmoid_4:0", shape=(7000, 1), dtype=float64)
Tensor("logistic_loss_2:0", shape=(7000, 1), dtype=float64)
```

In [56]:
```
optimizer = tf.train.GradientDescentOptimizer(learning_rate = 1e-4,name="Gra
dientDescent").minimize(cost)
init = tf.global_variables_initializer()
```

In [57]:
```
saver = tf.train.Saver()
with tf.Session() as sess:
  sess.run(init)
  print("Bias",sess.run(b))
  print("Weight",sess.run(W))
  for epoch in range(400):
    sess.run(optimizer)
    preds=sess.run(Y_hat)
    acc=((preds==ta).sum())/len(train_y)
    accu=[]
    repoch=False
    if repoch:
      accu.append(acc)
    if epoch % 1000 == 0:
      print("Accuracy",acc)
    saved_path = saver.save(sess, 'TSession')
```

```
Bias [-0.50074477]
Weight [[ 0.28123355 -0.49477198]]
Accuracy 0.9008571428571429
```

In [58]:
```
preds=predict_tweet(test_x)
print(preds,len(test_y))
```

```
INFO:tensorflow:Restoring parameters from TSession
[[1.00000000e+000]
 [0.00000000e+000]
 [0.00000000e+000]
 ...
 [1.00000000e+000]
 [1.34292262e-124]
 [1.00000000e+000]] 3000
```

In [59]:
```
def calculate_accuracy(x,y):
  if len(x)!=len(y):
    print("dimensions are different")
    return
  return ((x==y).sum())/len(y)
```

In [60]:
```
print(calculate_accuracy(preds,test_y))
```

```
0.9266666666666666
```