LAB 7

AIM: Manual version of Logistic Regression with TF based version.

```
import numpy as np
import pandas as pd
import tensorflow.compat.v1 as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from nltk.corpus import twitter_samples
import nltk
tf.disable_v2_behavior()


nltk.download('twitter_samples')
nltk.download('stopwords')

import re
import string
import numpy as np

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

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

```
#process tweet function
def process_tweet(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  word not in string.punctuation):
                  stem_words=stemmer.stem(word)

                  tweets_clean.append(stem_words)
    return tweets_clean


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


#prepare data
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')

data=all_positive_tweets+all_negative_tweets
d=np.append(np.ones((len(all_positive_tweets), 1)), np.zeros((len(all_negative_tweets), 1)), axis=0)

train_x,test_x,train_y,test_y=train_test_split(data,d,test_size=0.25,random_state=142)


freqs = build_freqs(train_x,train_y)

# check the output
print("type(freqs) = " + str(type(freqs)))
print("len(freqs) = " + str(len(freqs.keys())))
```

```
type(freqs) = <class 'dict'>
len(freqs) = 10859
```

```
#Logistic regression using TF

#extract feature function
def extract_features(tweet,freqs):
  word1=process_tweet(tweet)
  x=np.zeros((1,2))
  for word in word1:
    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))
```

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

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

```
[ 312. 3949.]
```

```
#predict function
def predict_tweet(tweet):
  with tf.Session() as sess:
    saver.restore(sess,save_path='TSession')
    data=[]
    for t in tweet:
      data.append(extract_features(t,freqs))
    data=np.asarray(data)
    return sess.run(tf.nn.sigmoid(tf.add(tf.matmul(a=data,b=W,transpose_b=True),b)))
  print("fail")
  return

#bias and weight
b=tf.Variable(np.random.randn(1),name="Bias")
W=tf.Variable(np.random.randn(1,2),name="Bias")


#extract all features
data=[]
for t in train_x:
    data.append(extract_features(t,freqs))
data=np.asarray(data)


#sigmoid function and cost function
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_8:0", shape=(7500, 1), dtype=float64)
Tensor("logistic_loss_4:0", shape=(7500, 1), dtype=float64)
```

```
#Gradient Descent Optimizer
optimizer=tf.train.GradientDescentOptimizer(learning_rate=1e-4,name="GradientDescent").minimize(cost)
init=tf.global_variables_initializer()

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 [0.45161577]
 Weight [[ 0.13756078 -0.72522957]]
 Accuracy 0.8928
```

```
#predict all tweet
preds=predict_tweet(test_x)
print(preds,len(test_y))
```

```
INFO:tensorflow:Restoring parameters from TSession
[[1.]
 [1.]
 [0.]
 ...
 [1.]
 [1.]
 [1.]] 2500
```

```
#calculate accuracy
def calculate_accuracy(x,y):
  if len(x)!=len(y):
    print("dimnsion are different")
    return
  return ((x==y).sum())/len(y)

print(calculate_accuracy(preds,test_y))
```

```
0.9172
```