import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import pickle,sys,os

from scipy import stats

import tensorflow as tf

import seaborn as sns

from pylab import rcParams

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

sns.set(style="darkgrid", palette="deep", font\_scale=1.5)

rcParams['figure.figsize']= 14,8

RANDOM\_SEED = 42

columns = ["user", "activity","timestamp","x-axis", "y-axis", "z-axis"]

df=pd.read\_csv("WISDMData.txt", header=None, names=columns)

df.head()

df["activity"].value\_counts().plot(kind="bar", title="Training examples by activity types")

df["user"].value\_counts().plot(kind="bar", title="Training examples by user types")

#Removing ";" from z-axis

df=df.applymap(lambda x: str(x).lstrip(";").rstrip(';'))

df['z-axis']= df['z-axis'].str.replace(";","")

df['x-axis']=df['x-axis'].astype(str).astype(float)

df['y-axis']=df['y-axis'].astype(str).astype(float)

df['z-axis']=df['z-axis'].astype(str).astype(float)

df['timestamp']=df['timestamp'].astype(str).astype(float)

def plot\_activity(activity, df):

data=df[df['activity']==activity][['x-axis', 'y-axis', 'z-axis']][:200]

axis= data.plot(subplots=True, figsize =(16,12), title = activity)

for ax in axis:

ax.legend(loc='lower left', bbox\_to\_anchor=(1.0, 0.5) )

plot\_activity("Sitting", df)

plot\_activity("Standing", df)

plot\_activity("Walking", df)

plot\_activity("Jogging", df)

df.activity.unique()

df['z-axis'].count()

len(df.columns)

df.groupby('activity').apply(lambda x: x.sort\_values('user'))

df.count()

#How much of your data is missing

df.isnull().sum().sort\_values(ascending=False).head()

#impute missing values using imputer in sklearn.preprocessing

# from sklearn.preprocessing import Imputer

#imp=Imputer(missing\_values='NaN', strategy='most\_frequent',axis=0)

#imp.fit(df.values[:, 3:5])

#df.values[:, 3:5]=imp.transform(df.values[:, 3:5])

df.dtypes #Listing data types

N\_TIME\_STEPS =200

N\_FEATURES =3

step =20

segments = []

labels= []

for i in range(0, len(df) - N\_TIME\_STEPS, step):

xs = df['x-axis'].values[i: i+N\_TIME\_STEPS]

ys= df['y-axis'].values[i: i+N\_TIME\_STEPS]

zs= df['z-axis'].values[i: i+N\_TIME\_STEPS]

label= stats.mode(df['activity'][i: i+N\_TIME\_STEPS])[0][0]

segments.append([xs,ys,zs])

labels.append(label)

np.array(segments).shape

reshaped\_segments=np.asarray(segments, dtype=np.float32).reshape(-1, N\_TIME\_STEPS,N\_FEATURES)

labels = np.asarray(pd.get\_dummies(labels), dtype=np.float32)

reshaped\_segments.shape

labels[0]

#Spliting dataset into training and test set

x\_train,x\_test,y\_train,y\_test=train\_test\_split(reshaped\_segments,labels,

test\_size=0.2, random\_state=RANDOM\_SEED)

#Building the model

N\_CLASSES=6

N\_HIDDEN\_UNITS=64

def create\_LSTM\_model(inputs):

W={

'hidden':tf.Variable(tf.random\_normal([N\_FEATURES, N\_HIDDEN\_UNITS])),

'output':tf.Variable(tf.random\_normal([N\_HIDDEN\_UNITS, N\_CLASSES]))

}

biases={

'hidden':tf.Variable(tf.random\_normal([N\_HIDDEN\_UNITS], mean=1.0)),

'output':tf.Variable(tf.random\_normal([N\_CLASSES]))

}

x=tf.transpose(inputs, [1,0,2])

x=tf.reshape(x, [-1, N\_FEATURES])

hidden=tf.nn.relu(tf.matmul(x, W['hidden'])+

biases['hidden'])

hidden=tf.split(hidden,N\_TIME\_STEPS,0)

# Stack 2 LSTM layers

lstm\_layers=[tf.contrib.rnn.BasicLSTMCell(N\_HIDDEN\_UNITS,

forget\_bias=1.0) for \_ in range(2)]

lstm\_layers=tf.contrib.rnn.MultiRNNCell(lstm\_layers)

outputs, \_ = tf.contrib.rnn.static\_rnn(lstm\_layers, hidden, dtype=tf.float32)

#get output for the last time steps

lstm\_last\_output=outputs[-1]

return tf.matmul(lstm\_last\_output, W['output'])+biases['output']

#Creating placeholder for model

tf.reset\_default\_graph()

x=tf.placeholder(tf.float32, [None, N\_TIME\_STEPS,N\_FEATURES],

name='input')

y=tf.placeholder(tf.float32, [None, N\_CLASSES])

#Creating Model

pred\_y=create\_LSTM\_model(x)

pred\_softmax=tf.nn.softmax(pred\_y, name="y\_")

L2\_LOSS=0.0015

l2=L2\_LOSS \* \

sum(tf.nn.l2\_loss(tf\_var) for tf\_var in tf.trainable\_variables())

loss=tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(

logits=pred\_y, labels=y))+l2

LEARNING\_RATE=0.0025

optimizer=tf.train.AdamOptimizer(learning\_rate=LEARNING\_RATE).minimize(loss)

correct\_pred=tf.equal(tf.argmax(pred\_softmax,1),tf.argmax(y,1))

accuracy=tf.reduce\_mean(tf.cast(correct\_pred, dtype=tf.float32))

#training

N\_EPOCHS=50

BATCH\_SIZE=1024

saver=tf.train.Saver()

history= dict(train\_loss=[],

train\_acc=[],

test\_loss=[],

test\_acc=[])

sess=tf.InteractiveSession()

sess.run(tf.global\_variables\_initializer())

train\_count = len(x\_train)

for i in range(1, N\_EPOCHS+1):

for start, end in zip(range(0, train\_count, BATCH\_SIZE),

range(BATCH\_SIZE, train\_count+

1, BATCH\_SIZE)):

sess.run(optimizer, feed\_dict={x:x\_train[start:end],

y: y\_train[start:end]})

\_, acc\_test, loss\_test, sess.run([pred\_softmax, accuracy, loss], feed\_dict={

x: x\_test, y: y\_test})

history['train\_loss'].append(loss\_train)

history['train\_acc'].append(acc\_train)

history['test\_loss'].append(loss\_test)

history['test\_acc'].append(acc\_test)

while i !=1 and i% 10!= 0:

continue

printf("epoch: {i} test accuracy: {acc\_test} loss: {loss\_test}")

predictions, acc\_final, loss\_final, = sess.run([pred\_softmax,

accuracy,loss], feed\_dict={x: x\_test, y:y\_test} )

print()

printf('final results: accuracy: {acc\_final}, loss:{loss\_final}')

#Storing Model to disc

pickle.dump(predictions, open("predictions.p", "wb"))

pickle.dump(history, open("history.p", "wb"))

tf.train.write\_graph(Sess.graph\_def, '.', './har.pbtxt')

saver.save(sess, save\_path="./har.ckpt")

sess.close()