**AI BASED STOCK PRICE PREDICTION USING BERT MODEL**

**Program Code for Loading and Preprocessing Dataset:**

**(By BERT Transformer Model)**

import numpy as np

import pandas as pd

import tensorflow as tf

import matplotlib.pyplot as plt

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

path = 'embedding\_files/'

max\_embedding = pd.read\_json(path+'max\_embedding.json')

min\_embedding = pd.read\_json(path+'min\_embedding.json')

mean\_embedding = pd.read\_json(path+'mean\_embedding.json')

sum\_embedding = pd.read\_json(path+'sum\_embedding.json')

djia = pd.read\_csv('data/DJIA\_table.csv')

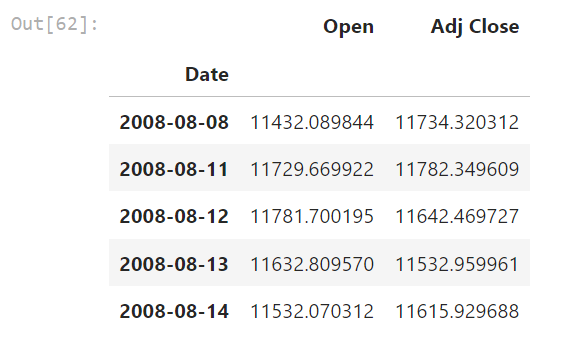
djia = djia.loc[:, ['Date', 'Open', 'Adj Close']].sort\_values('Date').set\_index('Date')



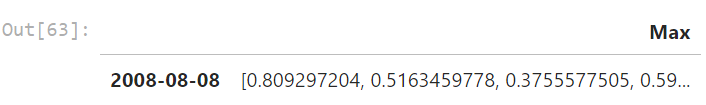
open\_price = djia[['Open']]

adj\_close\_price = djia[['Adj Close']]

djia.head()



max\_embedding.head(1)



**Program Code for Feature-Extraction and Classification:**

def transform\_data(tbl):

tbl = pd.DataFrame(tbl.iloc[:, 0].tolist())

tbl = tbl.set\_index(djia.index)

return tbl

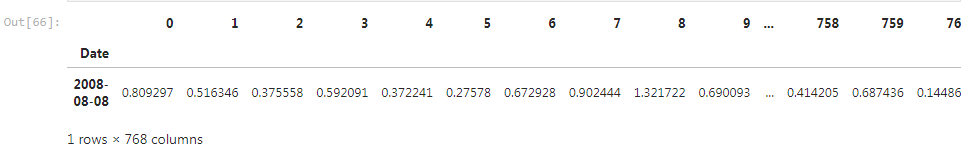
max\_embedding = transform\_data(max\_embedding)

min\_embedding = transform\_data(min\_embedding)

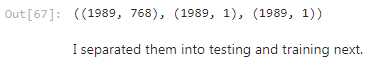
sum\_embedding = transform\_data(sum\_embedding)

mean\_embedding = transform\_data(mean\_embedding)

max\_embedding.head(1)



max\_embedding.shape, open\_price.shape, adj\_close\_price.shape



def split\_test(embedding, test\_size):

embedding\_test = embedding.iloc[-test\_size:, :]

embedding = embedding.iloc[:-test\_size, :]

return embedding\_test, embedding

test\_size = 300

max\_embedding\_test, max\_embedding = split\_test(max\_embedding, test\_size)

min\_embedding\_test, min\_embedding = split\_test(min\_embedding, test\_size)

sum\_embedding\_test, sum\_embedding = split\_test(sum\_embedding, test\_size)

mean\_embedding\_test, mean\_embedding = split\_test(mean\_embedding, test\_size)

combined\_embedding = pd.concat((mean\_embedding, max\_embedding, min\_embedding, sum\_embedding), axis=1)

combined\_embedding\_test = pd.concat((mean\_embedding\_test, max\_embedding\_test, min\_embedding\_test, sum\_embedding\_test), axis=1)

open\_test, open\_price = split\_test(open\_price, test\_size)

adj\_close\_test, adj\_close\_price = split\_test(adj\_close\_price, test\_size)

max\_embedding.shape, combined\_embedding.shape, open\_price.shape, adj\_close\_price.shape



def data\_loader(data, batch\_size, num\_iter=100):

# x : Embedding Values

# y : Open Price

# z : Close Price

x = data[0]

y = data[1]

z = data[2]

# num\_iter iterations per epoch

# mini batch

for \_ in range(num\_iter):

idx = np.random.choice(np.arange(x.shape[0]), size=batch\_size, replace=False)

batch\_x = x.iloc[idx, :]

batch\_y = y.iloc[idx]

batch\_z = z.iloc[idx]

yield batch\_x, batch\_y, batch\_z

class get\_model():

def \_\_init\_\_(self, learning\_rate=1e-3, dropout\_rate=.5):

self.learning\_rate = learning\_rate

self.dropout\_rate = dropout\_rate

# BERT Embedding

self.x = tf.placeholder(tf.float32, shape=(None, 768))

# Open Price

self.y = tf.placeholder(tf.float32, shape=(None, 1))

# Adj Close Price

self.z = tf.placeholder(tf.float32, shape=(None, 1))

self.pred = self.run\_model()

self.loss = tf.sqrt(tf.losses.mean\_squared\_error(self.z, self.pred), name='loss')

self.optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate, name='optimizer').minimize(self.loss)

self.saver = tf.train.Saver()

def run\_model(self):

# Dense model

layer1 = tf.contrib.layers.fully\_connected(self.x, 1000)

layer1 = tf.nn.dropout(layer1, rate=self.dropout\_rate)

layer1 = tf.layers.batch\_normalization(layer1)

layer2 = tf.contrib.layers.fully\_connected(layer1, 500)

layer2 = tf.nn.dropout(layer2, rate=self.dropout\_rate)

layer2 = tf.layers.batch\_normalization(layer2)

# This would be the value of coefficient indicating how much it impacts on a day's open price

layer3 = tf.contrib.layers.fully\_connected(layer2, 1)

layer4 = layer3 \* self.y

layer5 = tf.contrib.layers.fully\_connected(layer4, 100)

layer5 = tf.nn.dropout(layer5, rate=self.dropout\_rate)

layer5 = tf.layers.batch\_normalization(layer5)

output = tf.contrib.layers.fully\_connected(layer5, 1)

return output

def get\_data(embedding):

X = pd.concat((embedding, open\_price), axis=1)

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, adj\_close\_price, test\_size=.2)

return [X\_train.iloc[:, :-1], X\_train.iloc[:, -1:], y\_train], [X\_valid.iloc[:, :-1], X\_valid.iloc[:, -1:], y\_valid]

mean\_data\_train, mean\_data\_valid = get\_data(mean\_embedding)

max\_data\_train, max\_data\_valid = get\_data(max\_embedding)

min\_data\_train, min\_data\_valid = get\_data(min\_embedding)

sum\_data\_train, sum\_data\_valid = get\_data(sum\_embedding)

combined\_data\_train, combined\_data\_valid = get\_data(combined\_embedding)

# Train

data\_name = {'mean\_embedding':[mean\_data\_train, mean\_data\_valid],

'max\_embedding':[max\_data\_train, max\_data\_valid],

'min\_embedding':[min\_data\_train, min\_data\_valid],

'sum\_embedding':[sum\_data\_train, sum\_data\_valid]}

def train\_model(embedding\_name, learning\_rate=1e-5, epochs=300, batch\_size=16, dropout\_rate=.5, load\_params=True,

verbose=True, save\_model=True):

data\_train, data\_valid = data\_name[embedding\_name]

tf.reset\_default\_graph()

model = get\_model(learning\_rate=learning\_rate, dropout\_rate=dropout\_rate)

# For plots

train\_losses = []

valid\_losses = []

with tf.Session() as sess:

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

if load\_params:

# Load Model

try:

print(f'------------- Attempting to Load {embedding\_name} Model -------------')

model.saver.restore(sess, f'./model/{embedding\_name}\_model.ckpt')

print(f'------------- {embedding\_name} Model Loaded -------------')

except:

print('Training New Model')

else:

print('Training New Model')

# Train Model

print('\n------------- Training Model -------------\n')

for epoch in range(epochs):

for x, y, z in data\_loader(data\_train, batch\_size=batch\_size):

train\_loss, \_ = sess.run([model.loss, model.optimizer], feed\_dict={model.x:x,

model.y:y,

model.z:z})

# x : embedding, y : open price, z : close price

valid\_loss = sess.run(model.loss, feed\_dict={model.x:data\_valid[0],

model.y:data\_valid[1],

model.z:data\_valid[2]})

# print losses

if verbose:

print(f'Epoch {epoch+1}/{epochs}, Train RMSE Loss {train\_loss}, Valid RMSE Loss {valid\_loss}')

# Save Model at every 20 epochs

if save\_model:

if (epoch+1) % 20 == 0 and epoch > 0:

if not os.path.exists('./model'):

os.mkdir('./model/')

model.saver.save(sess, f"./model/{embedding\_name}\_model.ckpt")

print('\n------------- Model Saved -------------\n')

train\_losses.append(train\_loss)

valid\_losses.append(valid\_loss)

return model, train\_losses, valid\_losses

# Possible Names : mean\_embedding, max\_embedding, min\_embedding, sum\_embedding

epochs = 300

learning\_rate = 1e-4

#New Model for Combined Dataset

class combined\_model():

def \_\_init\_\_(self, learning\_rate=1e-3, dropout\_rate=.5):

self.learning\_rate = learning\_rate

self.dropout\_rate = dropout\_rate

# BERT Embedding

self.x = tf.placeholder(tf.float32, shape=(None, 3072))

# Open Price

self.y = tf.placeholder(tf.float32, shape=(None, 1))

# Adj Close Price

self.z = tf.placeholder(tf.float32, shape=(None, 1))

self.pred = self.run\_model()

self.loss = tf.sqrt(tf.losses.mean\_squared\_error(self.z, self.pred), name='loss')

self.optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate, name='optimizer').minimize(self.loss)

self.saver = tf.train.Saver()

def run\_model(self):

# Dense layers

layer1 = tf.contrib.layers.fully\_connected(self.x, 1000)

layer1 = tf.nn.dropout(layer1, rate=self.dropout\_rate)

layer1 = tf.layers.batch\_normalization(layer1)

layer2 = tf.contrib.layers.fully\_connected(layer1, 500)

layer2 = tf.nn.dropout(layer2, rate=self.dropout\_rate)

layer2 = tf.layers.batch\_normalization(layer2)

# Coefficient of impact values

layer3 = tf.contrib.layers.fully\_connected(layer2, 1)

layer4 = layer3 \* self.y

layer5 = tf.contrib.layers.fully\_connected(layer4, 100)

layer5 = tf.nn.dropout(layer5, rate=self.dropout\_rate)

layer5 = tf.layers.batch\_normalization(layer5)

output = tf.contrib.layers.fully\_connected(layer5, 1)

return output

tf.reset\_default\_graph()

model = combined\_model(learning\_rate=1e-4, dropout\_rate=.5)

epochs = 300

combined\_train\_losses = []

combined\_valid\_losses = []

with tf.Session() as sess:

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

try:

print(f'------------- Attempting to Load Combined Model -------------')

model.saver.restore(sess, f'./model/combined\_model.ckpt')

print(f'------------- Combined Model Loaded -------------')

except:

print('Training New Model')

# Train Model

print('\n------------- Training Model -------------\n')

for epoch in range(epochs):

for x, y, z in data\_loader(combined\_data\_train, batch\_size=16):

train\_loss, \_ = sess.run([model.loss, model.optimizer], feed\_dict={model.x:x,

model.y:y,

model.z:z})

valid\_loss = sess.run(model.loss, feed\_dict={model.x:combined\_data\_valid[0],

model.y:combined\_data\_valid[1],

model.z:combined\_data\_valid[2]})

if epoch % 20 == 0:

print(f'Epoch {epoch+1}/{epochs}, Combined Train RMSE Loss {train\_loss}, Combined Valid RMSE Loss {valid\_loss}')

if not os.path.exists('./model'):

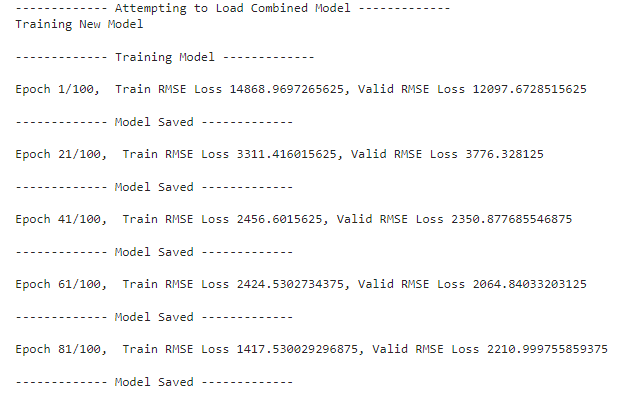
os.mkdir('./model/')

model.saver.save(sess, f"./model/combined\_model.ckpt")

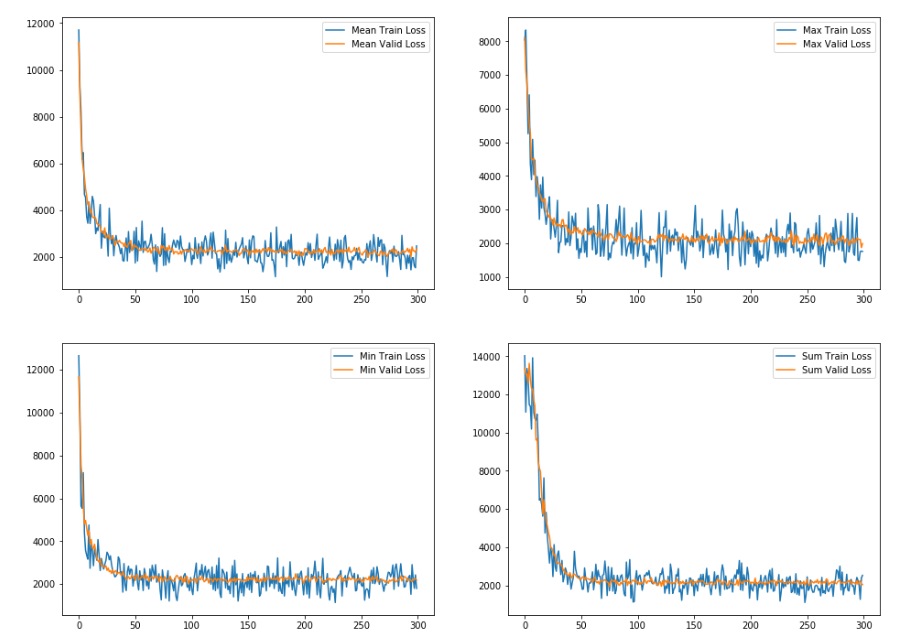
print('\n------------- Model Saved -------------\n')

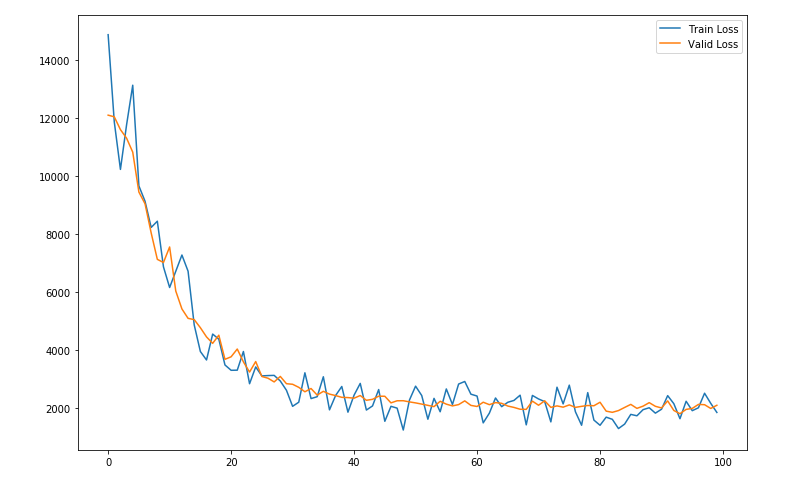
combined\_train\_losses.append(train\_loss)

combined\_valid\_losses.append(valid\_loss)



# Losses of each model





embedding\_data = {'mean\_embedding':mean\_embedding\_test,

'max\_embedding':max\_embedding\_test,

'min\_embedding':min\_embedding\_test,

'sum\_embedding':sum\_embedding\_test}

def predict\_model(embedding\_name):

tf.reset\_default\_graph()

data = embedding\_data[embedding\_name]

model = get\_model(learning\_rate=1e-5)

with tf.Session() as sess:

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

# Load Model

try:

print(f'------------- Attempting to Load {embedding\_name} Model -------------')

model.saver.restore(sess, f'./model/{embedding\_name}\_model.ckpt')

print('------------- Model Loaded -------------')

except:

pass

pred = sess.run(model.pred, feed\_dict={model.x:data,

model.y:open\_test})

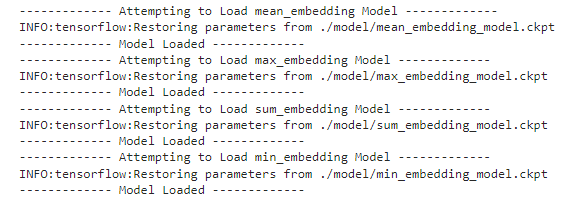
return model, pred

mean\_model, mean\_pred = predict\_model('mean\_embedding')

max\_model, max\_pred = predict\_model('max\_embedding')

sum\_model, sum\_pred = predict\_model('sum\_embedding')

min\_model, min\_pred = predict\_model('min\_embedding')



tf.reset\_default\_graph()

model = combined\_model(learning\_rate=1e-5)

with tf.Session() as sess:

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

# Load Model

try:

print(f'------------- Attempting to Load Combined Model -------------')

model.saver.restore(sess, f'./model/combined\_model.ckpt')

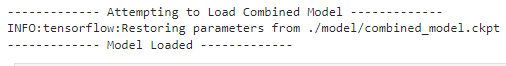
print('------------- Model Loaded -------------')

except:

pass

combined\_pred = sess.run(model.pred, feed\_dict={model.x:combined\_embedding\_test,

model.y:open\_test})



mean\_pred = mean\_pred.flatten()

max\_pred = max\_pred.flatten()

min\_pred = min\_pred.flatten()

sum\_pred = sum\_pred.flatten()

combined\_pred = combined\_pred.flatten()

