ass-4

November 16, 2023

[ ]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score,␣
 ↪precision_score
# TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

[ ]:
```
dataset = pd.read_csv(r"C:\Users\DELL\Desktop\DL\DL\PR 4 DL\creditcard.csv")
print("Any nulls in the dataset ",dataset.isnull().values.any() )
print("Label values ",dataset.Class.unique())
print(pd.value_counts(dataset['Class'], sort = True) )
count_classes = pd.value_counts(dataset['Class'], sort = True)
```

[ ]:
```
raw_data = dataset.values
labels = raw_data[:, -1]
data = raw_data[:, 0:-1]
train_data, test_data, train_labels, test_labels = train_test_split(data,␣
 ↪labels, test_size=0.2)
```

[ ]:
```
#Normalize the data to have a value between 0 and 1
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)
train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)
train_data = tf.cast(train_data, tf.float32)
test_data = tf.cast(test_data, tf.float32)
```

[ ]:
```
# Use only normal transactions to train the Autoencoder.
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
#creating normal and fraud datasets
normal_train_data = train_data[~train_labels]
```

```
normal_test_data = test_data[~test_labels]
fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print(" No. of records in Fraud Train Data=",len(fraud_train_data))
print(" No. of records in Normal Train data=",len(normal_train_data))
print(" No. of records in Fraud Test Data=",len(fraud_test_data))
print(" No. of records in Normal Test data=",len(normal_test_data))
```

[ ]:
```
# Create the Autoencoder
input_layer = tf.keras.layers.Input(shape=(30, ))

# Encoder
encoder = tf.keras.layers.Dense(units=14, activation="tanh",␣
 ↪activity_regularizer=tf.keras.regularizers.l2(0.02))(input_layer)
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(units=7, activation='relu')(encoder)
encoder = tf.keras.layers.Dense(units=4, activation=tf.nn.leaky_relu)(encoder)

# Decoder
decoder = tf.keras.layers.Dense(units=7, activation='relu')(encoder)
decoder = tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(units=14, activation='relu')(decoder)
decoder = tf.keras.layers.Dense(units=30, activation='tanh')(decoder)

# Autoencoder
autoencoder = tf.keras.Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

[ ]:
```
# Define the callbacks for checkpoints and early stopping

cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.
 ↪h5",mode='min', monitor='val_loss', verbose=2,
                                                                    ␣
 ↪        save_best_only=True)
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',min_delta=0.
 ↪0001,patience=10,verbose=1, mode='min',
                                                                    ␣
 ↪         restore_best_weights=True)
```

[ ]:
```
autoencoder.
 ↪compile(metrics=['accuracy'],loss='mean_squared_error',optimizer='adam')
```

[ ]:
```
history = autoencoder.fit(normal_train_data,␣
 ↪normal_train_data,epochs=50,batch_size=64,shuffle=True,
                           validation_data=(test_data,␣
 ↪test_data),verbose=1,callbacks=[cp, early_stop]).history
```

[ ]:
```
# Detect Anomalies on test data
test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2), axis=1)
error_df = pd.DataFrame({'Reconstruction_error': mse,'True_class': test_labels})
```

[ ]:
```
threshold_fixed =52
pred_y = [1 if e > threshold_fixed else 0 for e in error_df.
 ↪Reconstruction_error.values]
error_df['pred'] =pred_y
conf_matrix = confusion_matrix(error_df.True_class, pred_y)
print(conf_matrix)
```