REG NO: 212220220024

EX NO :9	MINI - PROJECT
DATE :24/11/2022	

CREDIT CARD FRAUD DETECTION

DESCRIPTION OF THE PROJECT

Our work intends to design a predictive model from the acquired dataset that contains user-sensitive information. The performance of fraud detection system is influenced by the sampling
approach on dataset, selection of variables and detection techniques used. This dataset retains numerical input because of PCA dimensionality reduction and is imbalanced highly.
The data is preprocessed, necessary packages and libraries are imported, and the credit card transactions are fed into the machine learning models as input. The output will be displayed as
either a fraudulent or valid transaction by observing the pattern. The precision and recall graph are used for representing the accuracy of the "Autoencoders" algorithm used.
Moreover, the reconstruction error is observed and it varies accordingly when drawn with fraud and without fraud class

PROGRAM CODE

```
# import packages
from keras import regularizers
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.layers import Input, Dense
from keras.models import Model, load_model
from pylab import rcParams
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_recall_fscore_support, f1_score
from sklearn.metrics import recall_score, classification_report, auc, roc_curve
from sklearn.metrics import confusion_matrix, precision_recall_curve
from sklearn.model_selection import train_test_split
import pickle
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from scipy import stats
import numpy as np
import pandas as pd
get_ipython().run_line_magic('matplotlib', 'inline')
# set random seed and percentage of test data
RANDOM_SEED = 314 # used to help randomly select the data points
TEST_PCT = 0.2 # 20% of the data
# set up graphic style in this case I am using the color scheme from xkcd.com
rcParams['figure.figsize'] = 14, 8.7 # Golden Mean
LABELS = ["Normal", "Fraud"]
col_list = ["cerulean", "scarlet"] # https://xkcd.com/color/rgb/
sns.set(style='white', font_scale=1.75, palette=sns.xkcd_palette(col_list))
df = pd.read_csv("creditcard.csv")
```

```
df.head(n=5)
df.shape
df.isnull().values.any()
pd.value_counts(df['Class'], sort=True)
# **Balance of Data Visualization**
# plotting a bar graph of the dataset
count_classes = pd.value_counts(df['Class'], sort=True)
count_classes.plot(kind='bar', rot=0)
plt.title("Transaction class ditribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency")
# 2 types of transactions
# naming normal class as 0
normal_df = df[df.Class == 0]
# naming fraud class as 1
fraud_df = df[df.Class == 1]
normal_df.shape
fraud_df.shape
# Looking at the amount in different transaction classes
normal_df.Amount.describe()
fraud_df.Amount.describe()
# **graphical representation**
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(fraud_df.Amount, bins=bins)
ax1.set_title('Fraud')
ax2.hist(normal_df.Amount, bins=bins)
ax2.set_title('Normal')
plt.xlabel('Amount($)')
plt.ylabel('Number of transactions')
plt.xlim((0, 10000))
plt.yscale('log')
plt.show()
# **Fraudulent transactions relation with time**
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(normal_df.Time, normal_df.Amount)
ax1.set_title('Normal')
ax2.scatter(fraud_df.Time, fraud_df.Amount)
ax2.set_title('Fraud')
plt.xlabel('Time (in seconds)')
plt.ylabel('Amount')
plt.show()
# **Autoencoder**
# **Normalize and Scale Data**
# data = df.drop(['Time'], axis=1) #if you think the var is unimportant
df_norm = df
df_norm['Time'] = StandardScaler().fit_transform(
```

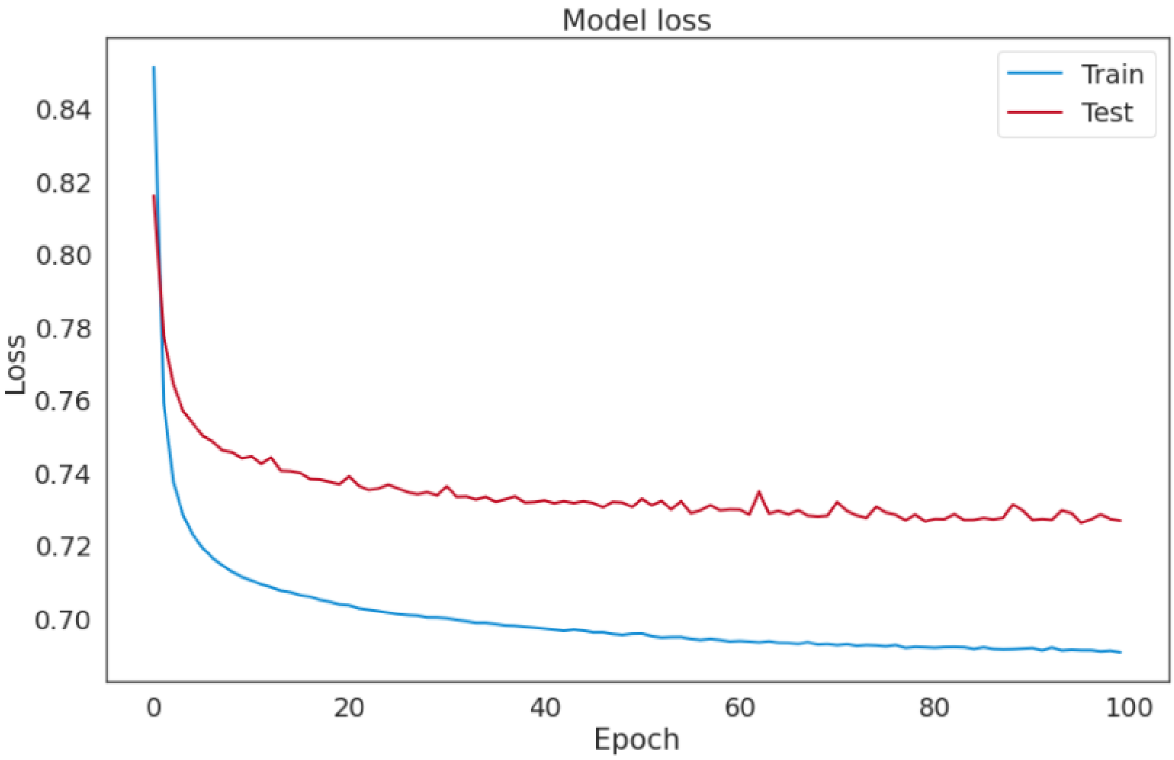
```
df_norm['Time'].values.reshape(-1, 1))
df_norm['Amount'] = StandardScaler().fit_transform(
df_norm['Amount'].values.reshape(-1, 1))
# Dividing Training and Test Set
train_x, test_x = train_test_split(
df_norm, test_size=TEST_PCT, random_state=RANDOM_SEED)
train_x = train_x[train_x.Class == 0] # where normal transactions
train_x = train_x.drop(['Class'], axis=1) # drop the class column
test_y = test_x['Class'] # save the class column for the test set
test_x = test_x.drop(['Class'], axis=1) # drop the class column
train_x = train_x.values # transform to ndarray
test_x = test_x.values
train_x.shape
# **Creating the Model using Autoencoders**
nb_epoch = 100
batch_size = 128
input_dim = train_x.shape[1] # num of columns, 30
encoding_dim = 14
hidden_dim = int(encoding_dim / 2) # i.e. 7
learning_rate = 1e-7
input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="tanh",
activity_regularizer=regularizers.l1(learning_rate))(input_layer)
encoder = Dense(hidden_dim, activation="relu")(encoder)
decoder = Dense(hidden_dim, activation='tanh')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(metrics=['accuracy'],
loss='mean_squared_error',
optimizer='adam')
cp = ModelCheckpoint(filepath="autoencoder_fraud.h5",
save_best_only=True,
verbose=0)
tb = TensorBoard(log_dir='./logs',
histogram_freq=0,
write_graph=True,
write_images=True)
history = autoencoder.fit(train_x, train_x,
epochs=nb_epoch,
batch_size=batch_size,
shuffle=True,
validation_data=(test_x, test_x),
verbose=1,
callbacks=[cp, tb]).history
# **Model Evaluation**
# Model Loss
plt.plot(history['loss'], linewidth=2, label='Train')
plt.plot(history['val_loss'], linewidth=2, label='Test')
plt.legend(loc='upper right')
plt.title('Model loss')
```

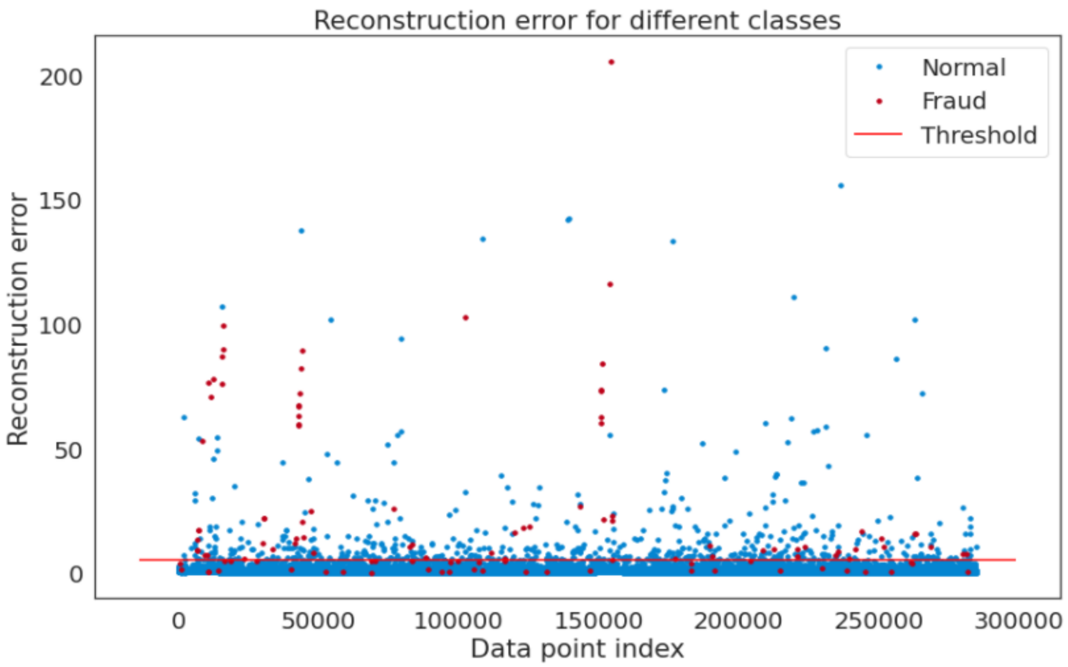
```
plt.ylabel('Loss')
plt.xlabel('Epoch')
# plt.ylim(ymin=0.70,ymax=1)
plt.show()
# Reconstruction Error Check
test_x_predictions = autoencoder.predict(test_x)
mse = np.mean(np.power(test_x - test_x_predictions, 2), axis=1)
error_df = pd.DataFrame({'Reconstruction_error': mse,
True_class': test_y})
error_df.describe()
# ROC Curve Check
false_pos_rate, true_pos_rate, thresholds = roc_curve(
error_df.True_class, error_df.Reconstruction_error)
roc_auc = auc(false_pos_rate, true_pos_rate,)
plt.plot(false_pos_rate, true_pos_rate,
linewidth=5, label='AUC = %0.3f' % roc_auc)
plt.plot([0, 1], [0, 1], linewidth=5)
plt.xlim([-0.01, 1])
plt.ylim([0, 1.01])
plt.legend(loc='lower right')
plt.title('Receiver operating characteristic curve (ROC)')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
# **Recall vs. Precision Thresholding**
precision_rt, recall_rt, threshold_rt = precision_recall_curve(
error_df.True_class, error_df.Reconstruction_error)
plt.plot(recall_rt, precision_rt, linewidth=5, label='Precision-Recall curve')
plt.title('Recall vs Precision')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
plt.plot(threshold_rt, precision_rt[1:], label="Precision", linewidth=5)
plt.plot(threshold_rt, recall_rt[1:], label="Recall", linewidth=5)
plt.title('Precision and recall for different threshold values')
plt.xlabel('Threshold')
plt.ylabel('Precision/Recall')
plt.legend()
plt.show()
# **Reconstruction Error vs Threshold Check**
threshold_fixed = 5
groups = error_df.groupby('True_class')
fig, ax = plt.subplots()
for name, group in groups:
ax.plot(group.index, group.Reconstruction_error, marker='o', ms=3.5, linestyle='',
label="Fraud" if name == 1 else "Normal")
ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[
1], colors="r", zorder=100, label='Threshold')
ax.legend()
```

```
plt.title("Reconstruction error for different classes")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
# **Confusion Matrix**
pred_y = [
1 if e > threshold_fixed else 0 for e in error_df.Reconstruction_error.values]
conf_matrix = confusion_matrix(error_df.True_class, pred_y)
plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS,
yticklabels=LABELS, annot=True, fmt="d")
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

SCREENSHOTS OF OUTPUT

MODEL LOSS:




Reconstruction error for different classes
Normal
Fraud
Threshold
Reconstruction error
0
50
100
150
200
0
50000
100000
150000
200000
250000
300000
Data point index


Card fraud losses 2018 split by type (as a percentage of total losses)
2009
9%
11%
1%
18%
61%
2018
7%
14%
1%
2%
76%
Lost & Stolen
Card not received
Counterfeit card
Remote Purchase (CNP)
Card ID Theft