1. Accomplishments to date

* Determined the data imbalance is 99.8% for normal transactions and 0.2% fraudulent transactions. Also created a bar plot to visually show just how large the imbalance is.
* To reduce training and testing time I took a sample of 10% from the original features and classes to train my models with.
* Then I performed an initial train, test, split with the testing size being 20%. Next, I repeated the train, test, split for validation.
* Created a KneighborsClassifier and used GridSearchCV with 5 fold cross validation. This resulted in an accuracy score of 100%.
* Created a DecisionTreeClassifier and used GridSearchCV with 5 fold cross validation. This resulted in an accuracy score of 100%.
* Created a GaussianNB and used GridSearchCV with 5 fold cross validation. This resulted in an accuracy score of 100%.
* Created a LinearRegression and used GridSearchCV with 5 fold cross validation. This resulted in an accuracy score of 100%.
* Created a LogisticRegression and used GridSearchCV with 5 fold cross validation. This resulted in an accuracy score of 100%.
* Created a LinearSVC and used GridSearchCV with 5 fold cross validation. This resulted in an accuracy score of 100%.
* Created a Keras neural network using all 30 features in the first layer 1 hidden layer with 50 nodes and an output layer with 1 node.

2. Remaining steps

* Need to check models for over fitting
* Find a better metric to use for imbalanced data
* Create better visuals to illustrate models and data
* Fix neural network model and tensorboard

**Code:**

'''

Alex Lux

The challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase.

Main challenges involved in credit card fraud detection are:

1. Enormous Data is processed every day and the model build must be fast enough to respond to the scam in time.

2. Imbalanced Data i.e most of the transactions (99.8%) are not fraudulent which makes it really hard for detecting the fraudulent ones

3. Data availability as the data is mostly private.

4. Misclassified Data can be another major issue, as not every fraudulent transaction is caught and reported.

5. Adaptive techniques used against the model by the scammers.

How to tackle these challenges?

1. The model used must be simple and fast enough to detect the anomaly and classify it as a fraudulent transaction as quickly as possible.

2. Imbalance can be dealt with by properly using some methods which we will talk about in the next paragraph

3. For protecting the privacy of the user the dimensionality of the data can be reduced.

4. A more trustworthy source must be taken which double-check the data, at least for training the model.

5. We can make the model simple and interpretable so that when the scammer adapts to it with just some tweaks we can have a new model up and running to deploy.

'''

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

LABELS = ["Normal", "Fraud"]

df = pd.read\_csv("creditcard.csv")

print("===============================================================================================")

print("======================================== DATA FRAME ===========================================")

print("===============================================================================================")

print(df.head())

print("===============================================================================================")

print("===================================== DATA DESCRIPTION ========================================")

print("===============================================================================================")

print(df.describe())

print()

print("===================================== NULL VALUES ========================================")

print(df.isnull().values.any())

count\_classes = pd.value\_counts(df['Class'], sort = True)

print(print("===================================== CLASS COUNT (99.8% Normal, 0.2% fraud) ========================================"))

print(count\_classes)

count\_classes.plot(kind = 'bar', rot=0)

plt.title("Transaction Class Distribution")

plt.xticks(range(2), LABELS)

plt.xlabel("Class")

plt.ylabel("Frequency")

plt.show()

X = df.drop(['Class'], axis = 1)

y = df['Class']

X\_data = X.values

y\_data = y.values

X\_data\_sample = X.sample(frac=0.1, random\_state=123)

y\_data\_sample = y.sample(frac=0.1 , random\_state=123)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_data\_sample, y\_data\_sample, test\_size=0.2, random\_state=123)

# print("X train:\n", X\_train)

# print("y train:\n", y\_train)

import warnings

warnings.filterwarnings('ignore', category=UserWarning)

import sys, os

if not sys.warnoptions:

    warnings.simplefilter("ignore")

    os.environ["PYTHONWARNINGS"] = "ignore" # Also affect subprocesses

X\_train\_new, X\_valid, y\_train\_new, y\_valid = train\_test\_split(X\_train, y\_train, test\_size=0.2, random\_state=123)

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.neighbors import KNeighborsClassifier

parameters\_knn = {'n\_neighbors':[1, 3, 5, 7, 9, 11, 13, 15], 'metric': ["manhattan", "chebyshev", "hamming"], 'weights': ["uniform", "distance"]}

knn = KNeighborsClassifier()

knn\_clf = GridSearchCV(estimator=knn, param\_grid=parameters\_knn, cv=5, n\_jobs=-1, verbose=1)

knn\_clf.fit(X\_train\_new, y\_train\_new)

print(f"BEST PARAMETERS FOR KNN CLASSIFIER: {knn\_clf.best\_params\_}")

knn\_best = knn\_clf.best\_estimator\_

knn\_predictions = knn\_best.predict(X\_valid)

print(classification\_report(y\_valid, knn\_predictions))

from sklearn.tree import DecisionTreeClassifier

parameters\_dt = {"criterion": ["gini", "entropy"], "splitter": ["best", "random"]}

dt = DecisionTreeClassifier()

dt\_clf = GridSearchCV(estimator=dt, param\_grid=parameters\_dt, cv=5, n\_jobs=-1, verbose=1)

dt\_clf.fit(X\_train\_new, y\_train\_new)

print(f"BEST PARAMETERS FOR DECISION TREE CLASSIFIER: {dt\_clf.best\_params\_}")

dt\_best = dt\_clf.best\_estimator\_

dt\_predictions = dt\_best.predict(X\_valid)

print(classification\_report(y\_valid, dt\_predictions))

from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

params\_NB = {'var\_smoothing': np.logspace(0,-9, num=100)}

gs\_NB\_clf = GridSearchCV(estimator=nb, param\_grid=params\_NB, cv=5, n\_jobs=-1, verbose=1)

gs\_NB\_clf.fit(X\_train\_new, y\_train\_new)

print(f"BEST PARAMETERS FOR GAUSSIAN NAIVE BAYES CLASSIFIER: {gs\_NB\_clf.best\_params\_}")

nb\_best = gs\_NB\_clf.best\_estimator\_

nb\_predictions = nb\_best.predict(X\_valid)

print(classification\_report(y\_valid, nb\_predictions))

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

parameters\_lr = {'fit\_intercept':[True,False],'copy\_X':[True, False]}

lr\_clf = GridSearchCV(estimator=lr, param\_grid=parameters\_lr, cv=5, n\_jobs=-1, verbose=1)

lr\_clf.fit(X\_train\_new, y\_train\_new)

print(f"BEST PARAMETERS FOR LINEAR REGRESSION CLASSIFIER: {lr\_clf.best\_params\_}")

lr\_best = lr\_clf.best\_estimator\_

lr\_predictions = lr\_best.predict(X\_valid)

print(classification\_report(y\_valid, lr\_predictions.round()))

from sklearn.linear\_model import LogisticRegression

logistic\_r = LogisticRegression()

parameters\_logistic\_r = {'penalty':['none', 'l2', 'l1', 'elasticnet'], 'C':[0.01, 0.1, 0.5, 1], 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}

logistic\_r\_clf = GridSearchCV(estimator=logistic\_r, param\_grid=parameters\_logistic\_r, cv=5, n\_jobs=-1, verbose=1)

logistic\_r\_clf.fit(X\_train\_new, y\_train\_new)

print(f"BEST PARAMETERS FOR LINEAR REGRESSION CLASSIFIER: {logistic\_r\_clf.best\_params\_}")

logistic\_r\_best = logistic\_r\_clf.best\_estimator\_

logistic\_r\_predictions = logistic\_r\_best.predict(X\_valid)

print(classification\_report(y\_valid, logistic\_r\_predictions))

from sklearn.svm import LinearSVC

svc = LinearSVC()

parameters\_svc = {'penalty': ['l1', 'l2', 'elasticnet', 'none'], 'loss': ['hinge', 'squared\_hinge'], 'C': [1 , 10 , 0.1]}

svc\_clf = GridSearchCV(estimator=svc, param\_grid=parameters\_svc, cv=3, n\_jobs=-1, verbose=1)

svc\_clf.fit(X\_train\_new, y\_train\_new)

print(f"BEST PARAMETERS FOR LinearSVC CLASSIFIER: {svc\_clf.best\_params\_}")

svc\_best = svc\_clf.best\_estimator\_

svc\_predictions = svc\_best.predict(X\_valid)

print(classification\_report(y\_valid, svc\_predictions))

from tensorflow import keras

import tensorflow as tf

from keras.layers import Dense

from keras.models import Sequential

# Number of features (first layer inputs)

n\_inputs = 30

nn\_model = Sequential()

# define first hidden layer and visible layer

nn\_model.add(Dense(50, input\_dim=n\_inputs, activation='relu', kernel\_initializer='he\_uniform'))

# define output layer

nn\_model.add(Dense(1, activation='sigmoid'))

# define loss and optimizer

nn\_model.compile(loss='binary\_crossentropy', optimizer='adam')

import datetime

log\_dir = "logs/" + datetime.datetime.now().strftime("%Y-%m-%d-%H\_%M\_%S")

filepath = 'nn\_model.hdf5'

from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint(filepath=filepath, monitor='val\_accuracy', verbose=3, save\_best\_only=True, mode='max')

tensorboard\_callbacks = tf.keras.callbacks.TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

nn\_model.fit(X\_train\_new, y\_train\_new, epochs=10, callbacks=[checkpoint, tensorboard\_callbacks])

eval = nn\_model.evaluate(X\_valid, y\_valid)

print(f"EVALUATION: {eval}")

nn\_predictions = nn\_model.predict(X\_valid)

from sklearn.metrics import roc\_auc\_score

print(roc\_auc\_score(y\_valid,nn\_predictions))

nn\_predictions\_flat = nn\_predictions.flatten()

y\_pred = np.where(nn\_predictions\_flat > 0.5, 1, 0)

print(accuracy\_score(y\_valid, y\_pred))

print(classification\_report(y\_valid, y\_pred))

from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error

r2Score = r2\_score(y\_valid, y\_pred)

maeScore = mean\_absolute\_error(y\_valid, y\_pred)

mseScore = mean\_squared\_error(y\_valid, y\_pred)

print(f"R2: {r2Score}, MAE: {maeScore}, MSE: {mseScore}")

**Terminal Output:**

PS C:\Users\Alex Lux\Desktop\DATA MINING\Project> python3 .\kaggleCCF.py

===============================================================================================

======================================== DATA FRAME ===========================================

===============================================================================================

Time V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 ... V20 V21 V22 V23 V24 V25 V26 V27 V28 Amount Class

0 0.0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.239599 0.098698 0.363787 0.090794 ... 0.251412 -0.018307 0.277838 -0.110474 0.066928 0.128539 -0.189115 0.133558 -0.021053 149.62

0

1 0.0 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803 0.085102 -0.255425 -0.166974 ... -0.069083 -0.225775 -0.638672 0.101288 -0.339846 0.167170 0.125895 -0.008983 0.014724 2.69

0

2 1.0 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461 0.247676 -1.514654 0.207643 ... 0.524980 0.247998 0.771679 0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752 378.66

0

3 1.0 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 0.237609 0.377436 -1.387024 -0.054952 ... -0.208038 -0.108300 0.005274 -0.190321 -1.175575 0.647376 -0.221929 0.062723 0.061458 123.50

0

4 2.0 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 0.592941 -0.270533 0.817739 0.753074 ... 0.408542 -0.009431 0.798278 -0.137458 0.141267 -0.206010 0.502292 0.219422 0.215153 69.99

0

[5 rows x 31 columns]

===============================================================================================

===================================== DATA DESCRIPTION ========================================

===============================================================================================

Time V1 V2 V3 V4 V5 V6 ... V24 V25 V26 V27 V28 Amount Class

count 284807.000000 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 ... 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 284807.000000 284807.000000

mean 94813.859575 1.168375e-15 3.416908e-16 -1.379537e-15 2.074095e-15 9.604066e-16 1.487313e-15 ... 4.473266e-15 5.340915e-16 1.683437e-15 -3.660091e-16 -1.227390e-16 88.349619 0.001727

std 47488.145955 1.958696e+00 1.651309e+00 1.516255e+00 1.415869e+00 1.380247e+00 1.332271e+00 ... 6.056471e-01 5.212781e-01 4.822270e-01 4.036325e-01 3.300833e-01 250.120109 0.041527

min 0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00 -1.137433e+02 -2.616051e+01 ... -2.836627e+00 -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01 0.000000 0.000000

25% 54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01 -6.915971e-01 -7.682956e-01 ... -3.545861e-01 -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02 5.600000 0.000000

50% 84692.000000 1.810880e-02 6.548556e-02 1.798463e-01 -1.984653e-02 -5.433583e-02 -2.741871e-01 ... 4.097606e-02 1.659350e-02 -5.213911e-02 1.342146e-03 1.124383e-02 22.000000 0.000000

75% 139320.500000 1.315642e+00 8.037239e-01 1.027196e+00 7.433413e-01 6.119264e-01 3.985649e-01 ... 4.395266e-01 3.507156e-01 2.409522e-01 9.104512e-02 7.827995e-02 77.165000 0.000000

max 172792.000000 2.454930e+00 2.205773e+01 9.382558e+00 1.687534e+01 3.480167e+01 7.330163e+01 ... 4.584549e+00 7.519589e+00 3.517346e+00 3.161220e+01 3.384781e+01 25691.160000 1.000000

[8 rows x 31 columns]

===================================== NULL VALUES ========================================

False

===================================== CLASS COUNT (99.8% Normal, 0.2% fraud) ========================================

None

0 284315

1 492

Name: Class, dtype: int64

Fitting 5 folds for each of 48 candidates, totalling 240 fits

BEST PARAMETERS FOR KNN CLASSIFIER: {'metric': 'manhattan', 'n\_neighbors': 3, 'weights': 'uniform'}

precision recall f1-score support

0 1.00 1.00 1.00 4549

1 1.00 0.12 0.22 8

accuracy 1.00 4557

macro avg 1.00 0.56 0.61 4557

weighted avg 1.00 1.00 1.00 4557

Fitting 5 folds for each of 4 candidates, totalling 20 fits

BEST PARAMETERS FOR DECISION TREE CLASSIFIER: {'criterion': 'gini', 'splitter': 'random'}

precision recall f1-score support

0 1.00 1.00 1.00 4549

1 0.71 0.62 0.67 8

accuracy 1.00 4557

macro avg 0.86 0.81 0.83 4557

weighted avg 1.00 1.00 1.00 4557

Fitting 5 folds for each of 100 candidates, totalling 500 fits

BEST PARAMETERS FOR GAUSSIAN NAIVE BAYES CLASSIFIER: {'var\_smoothing': 1.0}

precision recall f1-score support

0 1.00 1.00 1.00 4549

1 0.00 0.00 0.00 8

accuracy 1.00 4557

macro avg 0.50 0.50 0.50 4557

weighted avg 1.00 1.00 1.00 4557

Fitting 5 folds for each of 4 candidates, totalling 20 fits

BEST PARAMETERS FOR LINEAR REGRESSION CLASSIFIER: {'copy\_X': True, 'fit\_intercept': False}

precision recall f1-score support

0 1.00 1.00 1.00 4549

1 1.00 0.62 0.77 8

accuracy 1.00 4557

macro avg 1.00 0.81 0.88 4557

weighted avg 1.00 1.00 1.00 4557

Fitting 5 folds for each of 80 candidates, totalling 400 fits

BEST PARAMETERS FOR LINEAR REGRESSION CLASSIFIER: {'C': 0.01, 'penalty': 'none', 'solver': 'newton-cg'}

precision recall f1-score support

0 1.00 1.00 1.00 4549

1 0.71 0.62 0.67 8

accuracy 1.00 4557

macro avg 0.86 0.81 0.83 4557

weighted avg 1.00 1.00 1.00 4557

Fitting 3 folds for each of 24 candidates, totalling 72 fits

BEST PARAMETERS FOR LINEAR REGRESSION CLASSIFIER: {'C': 10, 'loss': 'hinge', 'penalty': 'l2'}

precision recall f1-score support

0 1.00 1.00 1.00 4549

1 0.00 0.00 0.00 8

accuracy 1.00 4557

macro avg 0.50 0.50 0.50 4557

weighted avg 1.00 1.00 1.00 4557

2022-05-06 17:21:33.656212: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'cudart64\_110.dll'; dlerror: cudart64\_110.dll not found

2022-05-06 17:21:33.656356: I tensorflow/stream\_executor/cuda/cudart\_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.

2022-05-06 17:21:36.184215: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'cudart64\_110.dll'; dlerror: cudart64\_110.dll not found

2022-05-06 17:21:36.184505: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'cublas64\_11.dll'; dlerror: cublas64\_11.dll not found

2022-05-06 17:21:36.184840: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'cublasLt64\_11.dll'; dlerror: cublasLt64\_11.dll not found

2022-05-06 17:21:36.185085: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'cufft64\_10.dll'; dlerror: cufft64\_10.dll not found

2022-05-06 17:21:36.185425: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'curand64\_10.dll'; dlerror: curand64\_10.dll not found

2022-05-06 17:21:36.185747: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'cusolver64\_11.dll'; dlerror: cusolver64\_11.dll not found

2022-05-06 17:21:36.186021: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'cusparse64\_11.dll'; dlerror: cusparse64\_11.dll not found

2022-05-06 17:21:36.186369: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'cudnn64\_8.dll'; dlerror: cudnn64\_8.dll not found

2022-05-06 17:21:36.186460: W tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1850] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.

Skipping registering GPU devices...

2022-05-06 17:21:36.186940: I tensorflow/core/platform/cpu\_feature\_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

Epoch 1/10

560/570 [============================>.] - ETA: 0s - loss: 287.8470WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 1ms/step - loss: 283.3059

Epoch 2/10

527/570 [==========================>...] - ETA: 0s - loss: 4.4827WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 1ms/step - loss: 5.2308

Epoch 3/10

565/570 [============================>.] - ETA: 0s - loss: 8.9520WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 1ms/step - loss: 8.8798

Epoch 4/10

568/570 [============================>.] - ETA: 0s - loss: 10.2503WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 931us/step - loss: 10.2216

Epoch 5/10

525/570 [==========================>...] - ETA: 0s - loss: 10.9842WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 916us/step - loss: 10.1243

Epoch 6/10

518/570 [==========================>...] - ETA: 0s - loss: 12.1647WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 918us/step - loss: 12.5059

Epoch 7/10

550/570 [===========================>..] - ETA: 0s - loss: 14.5881WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 963us/step - loss: 14.7423

Epoch 8/10

566/570 [============================>.] - ETA: 0s - loss: 15.2827WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 933us/step - loss: 15.1862

Epoch 9/10

549/570 [===========================>..] - ETA: 0s - loss: 9.3385WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 974us/step - loss: 9.0848

Epoch 10/10

549/570 [===========================>..] - ETA: 0s - loss: 12.3400WARNING:tensorflow:Can save best model only with val\_accuracy available, skipping.

570/570 [==============================] - 1s 1ms/step - loss: 11.8957

143/143 [==============================] - 0s 743us/step - loss: 0.5875

EVALUATION: 0.5874761343002319

0.8024153660145087

0.9982444590739522

precision recall f1-score support

0 1.00 1.00 1.00 4549

1 0.50 0.38 0.43 8

accuracy 1.00 4557

macro avg 0.75 0.69 0.71 4557

weighted avg 1.00 1.00 1.00 4557

R2: -0.001758628269949325, MAE: 0.0017555409260478386, MSE: 0.00175554092604783860..