Notebook

Credit Fraud Detector

Introduction

In this notebook we will use various predictive models to see how accurate they are in detecting whether a transacti
in the dataset, the features are scaled and the names of the features are not shown due to privacy reasons. Neverth
aspects of the dataset.

Our Goals:

- Understand the little distribution of the "little" data that was provided to us.
- Create a 50/50 sub-dataframe ratio of "Fraud" and "Non-Fraud" transactions. (NearMiss Algorith
- Determine the Classifiers we are going to use and decide which one has a higher accuracy.
- Create a Neural Network and compare the accuracy to our best classifier.
- Understand common mistaked made with imbalanced datasets.

Outline:

Correcting Previous Mistakes from Imbalanced Datasets:

- Never test on the oversampled or undersampled dataset.
- If we want to implement cross validation, remember to oversample or undersample your training
- Don't use **accuracy score** as a metric with imbalanced datasets (will be usually high and mislead **precision/recall score or confusion matrix**

References:

- Hands on Machine Learning with Scikit-Learn & TensorFlow by Aurélien Géron (O'Reilly). CopyRi
- Machine Learning - Over-& Undersampling - Python/ Scikit/ Scikit-Imblearn by Coding-Maniac
- auprc, 5-fold c-v, and resampling methods by Jeremy Lane (Kaggle Notebook)

Gather Sense of Our Data:

The first thing we must do is gather a **basic sense** of our data. Remember, except for the **transaction** columns are (due to privacy reasons). The only thing we know, is that those columns that are unknow

Summary:

- The transaction amount is relatively **small**. The mean of all the mounts made is approximately U
- There are no **"Null"** values, so we don't have to work on ways to replace values.
- Most of the transactions were **Non-Fraud** (99.83%) of the time, while **Fraud** transactions occurs

Feature Technicalities:

- **PCA Transformation:** The description of the data says that all the features went through a PCA technique) (Except for time and amount).
- **Scaling:** Keep in mind that in order to implement a PCA transformation features need to be prev features have been scaled or at least that is what we are assuming the people that develop the

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

# Imported Libraries

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA, TruncatedSVD
import matplotlib.patches as mpatches
import time
```

```
# Classifier Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import collections


# Other Libraries
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from imblearn.metrics import classification_report_imbalanced
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_
from collections import Counter
from sklearn.model_selection import KFold, StratifiedKFold
import warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv('/content/creditcard.csv')
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

```
df.describe()
```

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070ε
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563ε
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247ε
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433ε
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971ε
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583ε
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264ε
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167ε

```
# Good No Null Values!
df.isnull().sum().max()
```

```
0
```

```
df.columns
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

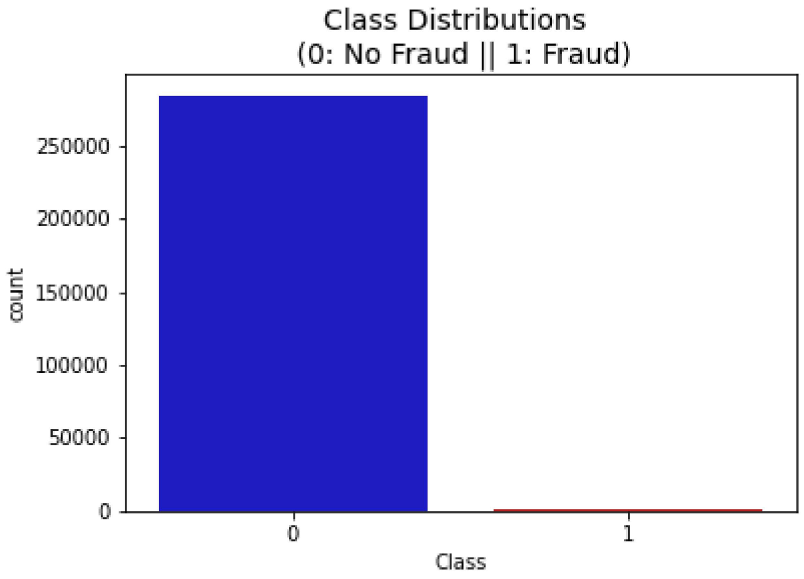
```
# The classes are heavily skewed we need to solve this issue later.
print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')
```

```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
```

Note: Notice how imbalanced is our original dataset! Most of the transactions are non-fraud. If we us predictive models and analysis we might get a lot of errors and our algorithms will probably overfit si are not fraud. But we don't want our model to assume, we want our model to detect patterns that give

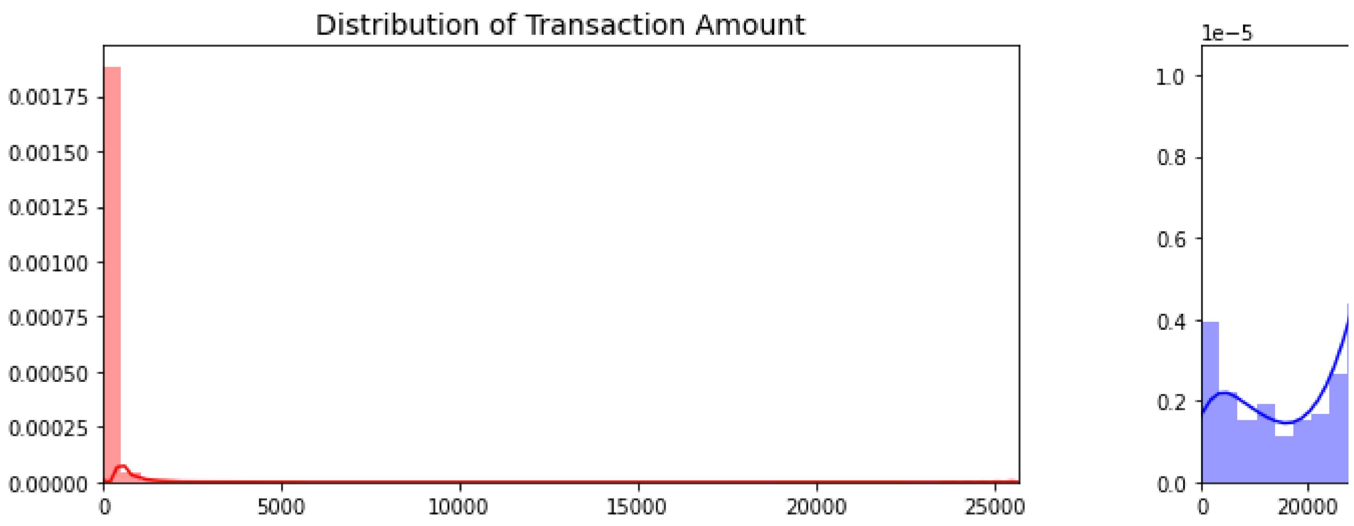
```
colors = ["#0101DF", "#DF0101"]

sns.countplot('Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
```

```
Text(0.5, 1.0, 'Class Distributions \n (0: No Fraud || 1: Fraud)')
```



Distributions: By seeing the distributions we can have an idea how skewed are these features, we car
features. There are techniques that can help the distributions be less skewed which will be implemen

```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount_val = df['Amount'].values
time_val = df['Time'].values

sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])



plt.show()
```



Scaling and Distributing

In this phase of our kernel, we will first scale the columns comprise of **Time** and **Amount** . Time and amount should
hand, we need to also create a sub sample of the dataframe in order to have an equal amount of Fraud and Non-Fra
understand patterns that determines whether a transaction is a fraud or not.

What is a sub-Sample?

In this scenario, our subsample will be a dataframe with a 50/50 ratio of fraud and non-fraud transactions. Meaning
fraud and non fraud transactions.

Why do we create a sub-Sample?

In the beginning of this notebook we saw that the original dataframe was heavily imbalanced! Using the original dat

- **Overfitting:** Our classification models will assume that in most cases there are no frauds! What when a fraud occurs.
- **Wrong Correlations:** Although we don't know what the "V" features stand for, it will be useful to u influence the result (Fraud or No Fraud) by having an imbalance dataframe we are not able to se and features.

Summary:

- **Scaled amount** and **scaled time** are the columns with scaled values.
- There are **492 cases** of fraud in our dataset so we can randomly get 492 cases of non-fraud to c
- We concat the 492 cases of fraud and non fraud, **creating a new sub-sample.**

```
# Since most of our data has already been scaled we should scale the columns that are left to
from sklearn.preprocessing import StandardScaler, RobustScaler
```

```
# RobustScaler is less prone to outliers.

std_scaler = StandardScaler()
rob_scaler = RobustScaler()

df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))

df.drop(['Time','Amount'], axis=1, inplace=True)


scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']

df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)

# Amount and Time are Scaled!

df.head()
```

Splitting the Data (Original DataFrame)

Before proceeding with the **Random UnderSampling technique** we have to separate the orginal datafr
**remember although we are splitting the data when implementing Random UnderSampling or OverSa
models on the original testing set not on the testing set created by either of these techniques.** The r
dataframes that were undersample and oversample (in order for our models to detect the patterns), a

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit

print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')

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

```
sss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

for train_index, test_index in sss.split(X, y):
    print("Train:", train_index, "Test:", test_index)
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]

# We already have X_train and y_train for undersample data thats why I am using original to d
# original_Xtrain, original_Xtest, original_ytrain, original_ytest = train_test_split(X, y, t

# Check the Distribution of the labels


# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(original_ytrain, return_counts=True)
test_unique_label, test_counts_label = np.unique(original_ytest, return_counts=True)
print('-' * 100)

print('Label Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
```

Random Under-Sampling:

In this phase of the project we will implement *"Random Under Sampling"* which basically consists o **balanced dataset** and thus avoiding our models to overfitting.

Steps:

- The first thing we have to do is determine how **imbalanced** is our class (use "value_counts()" on amount for each label)
- Once we determine how many instances are considered **fraud transactions** (Fraud = "1") , we sh the same amount as fraud transactions (assuming we want a 50/50 ratio), this will be equivalen non-fraud transactions.
- After implementing this technique, we have a sub-sample of our dataframe with a 50/50 ratio w step we will implement is to **shuffle the data** to see if our models can maintain a certain accura

Note: The main issue with "Random Under-Sampling" is that we run the risk that our classification mo would like to since there is a great deal of **information loss** (bringing 492 non-fraud transaction from

↳ *1 cell hidden*

Equally Distributing and Correlating:

Now that we have our dataframe correctly balanced, we can go further with our **analysis** and **data pre**

↳ *5 cells hidden*

Anomaly Detection:

Our main aim in this section is to remove "extreme outliers" from features that have a high correlation impact on the accuracy of our models.

Interquartile Range Method:

- **Interquartile Range (IQR):** We calculate this by the difference between the 75th percentile and 2 threshold beyond the 75th and 25th percentile that in case some instance pass this threshold th
- **Boxplots:** Besides easily seeing the 25th and 75th percentiles (both end of the squares) it is als beyond the lower and higher extreme).

Outlier Removal Tradeoff:

We have to be careful as to how far do we want the threshold for removing outliers. We determine the 1.5) by the (Interquartile Range). The higher this threshold is, the less outliers will detect (multiplying this threshold is the more outliers it will detect.

**The Tradeoff*: * The lower the threshold the more outliers it will remove however, we want to focus m outliers. Why? because we might run the risk of information loss which will cause our models to have threshold and see how it affects the accuracy of our classification models.

Summary:

- **Visualize Distributions:** We first start by visualizing the distribution of the feature we are going t V14 is the only feature that has a Gaussian distribution compared to features V12 and V10.
- **Determining the threshold:** After we decide which number we will use to multiply with the iqr (th proceed in determining the upper and lower thresholds by substrating q25 - threshold (lower ext threshold (upper extreme threshold).
- **Conditional Dropping:** Lastly, we create a conditional dropping stating that if the "threshold" is e will be removed.
- **Boxplot Representation:** Visualize through the boxplot that the number of "extreme outliers" hav

Note: After implementing outlier reduction our accuracy has been improved by over 3%! Some outliers but remember, we have to avoid an extreme amount of information loss or else our model runs the ris

Reference: More information on Interquartile Range Method: How to Use Statistics to Identify Outliers Learning Mastery blog)

↳ 6 cells hidden

Classifiers (UnderSampling):

In this section we will train four types of classifiers and decide which classifier will be more effective in detecting **fra** data into training and testing sets and separate the features from the labels.

Summary:

- **Logistic Regression** classifier is more accurate than the other three classifiers in most cases. (\ Regression)
- **GridSearchCV** is used to determine the paremeters that gives the best predictive score for the c
- Logistic Regression has the best Receiving Operating Characteristic score (ROC), meaning that separates **fraud** and **non-fraud** transactions.

Learning Curves:

- The **wider the gap** between the training score and the cross validation score, the more likely you
- If the score is low in both training and cross-validation sets this is an indication that our model i
- **Logistic Regression Classifier** shows the best score in both training and cross-validating sets.

↳ 13 cells hidden

A Deeper Look into LogisticRegression:

In this section we will ive a deeper look into the **logistic regression classifier**.

Terms:

- **True Positives:** Correctly Classified Fraud Transactions
- **False Positives:** Incorrectly Classified Fraud Transactions
- **True Negative:** Correctly Classified Non-Fraud Transactions
- **False Negative:** Incorrectly Classified Non-Fraud Transactions
- **Precision:** True Positives/(True Positives + False Positives)
- **Recall:** True Positives/(True Positives + False Negatives)
- Precision as the name says, says how precise (how sure) is our model in detecting fraud transa
cases our model is able to detect.
- **Precision/Recall Tradeoff:** The more precise (selective) our model is, the less cases it will detec
a precision of 95%, Let's say there are only 5 fraud cases in which the model is 95% precise or m
say there are 5 more cases that our model considers 90% to be a fraud case, if we lower the pre
model will be able to detect.

Summary:

- **Precision starts to descend** between 0.90 and 0.92 nevertheless, our precision score is still pret
score.

↳ 14 cells hidden

Test Data with Logistic Regression:

Confusion Matrix:

Positive/Negative: Type of Class (label) ["No", "Yes"] **True/False:** Correctly or Incorrectly classified by

True Negatives (Top-Left Square): This is the number of **correctly** classifications of the "No" (No Frau

False Negatives (Top-Right Square): This is the number of **incorrectly** classifications of the "No"(No F

False Positives (Bottom-Left Square): This is the number of **incorrectly** classifications of the "Yes" (F

True Positives (Bottom-Right Square): This is the number of **correctly** classifications of the "Yes" (Fra

Summary:

- **Random UnderSampling:** We will evaluate the final performance of the classification models in **in mind that this is not the data from the original dataframe.**

```
from sklearn.metrics import confusion_matrix

# Logistic Regression fitted using SMOTE technique
y_pred_log_reg = log_reg_sm.predict(X_test)

# Other models fitted with UnderSampling
y_pred_knear = knears_neighbors.predict(X_test)
y_pred_svc = svc.predict(X_test)
y_pred_tree = tree_clf.predict(X_test)


log_reg_cf = confusion_matrix(y_test, y_pred_log_reg)
kneighbors_cf = confusion_matrix(y_test, y_pred_knear)
svc_cf = confusion_matrix(y_test, y_pred_svc)
tree_cf = confusion_matrix(y_test, y_pred_tree)

fig, ax = plt.subplots(2, 2,figsize=(22,12))


sns.heatmap(log_reg_cf, ax=ax[0][0], annot=True, cmap=plt.cm.copper)
ax[0, 0].set_title("Logistic Regression \n Confusion Matrix", fontsize=14)
ax[0, 0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0, 0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(kneighbors_cf, ax=ax[0][1], annot=True, cmap=plt.cm.copper)
ax[0][1].set_title("KNearsNeighbors \n Confusion Matrix", fontsize=14)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(svc_cf, ax=ax[1][0], annot=True, cmap=plt.cm.copper)
ax[1][0].set_title("Suppor Vector Classifier \n Confusion Matrix", fontsize=14)
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(tree_cf, ax=ax[1][1], annot=True, cmap=plt.cm.copper)
ax[1][1].set_title("DecisionTree Classifier \n Confusion Matrix", fontsize=14)
ax[1][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][1].set_yticklabels(['', ''], fontsize=14, rotation=360)


plt.show()
```

```
from sklearn.metrics import classification_report


print('Logistic Regression:')
print(classification_report(y_test, y_pred_log_reg))
print('KNears Neighbors:')
print(classification_report(y_test, y_pred_knear))

print('Support Vector Classifier:')
print(classification_report(y_test, y_pred_svc))

print('Support Vector Classifier:')
print(classification_report(y_test, y_pred_tree))
```

```
# Final Score in the test set of logistic regression
from sklearn.metrics import accuracy_score

# Logistic Regression with Under-Sampling
y_pred = log_reg.predict(X_test)
undersample_score = accuracy_score(y_test, y_pred)



# Logistic Regression with SMOTE Technique (Better accuracy with SMOTE t)
y_pred_sm = best_est.predict(original_Xtest)
oversample_score = accuracy_score(original_ytest, y_pred_sm)


d = {'Technique': ['Random UnderSampling', 'Oversampling (SMOTE)'], 'Score': [undersample_sco
final_df = pd.DataFrame(data=d)

# Move column
score = final_df['Score']
final_df.drop('Score', axis=1, inplace=True)
final_df.insert(1, 'Score', score)

# Note how high is accuracy score it can be misleading!
final_df
```

Neural Networks Testing Random UnderSampling Data vs OverSamplir

In this section we will implement a simple Neural Network (with one hidden layer) in order to see whic
we implemented in the (undersample or oversample(SMOTE)) has a better accuracy for detecting frau

Our Main Goal:

Our main goal is to explore how our simple neural network behaves in both the random undersample
whether they can predict accuractely both non-fraud and fraud cases. Why not only focus on fraud? Ir
you purchased an item your card gets blocked because the bank's algorithm thought your purchase w
emphasize only in detecting fraud cases but we should also emphasize correctly categorizing non-fra

The Confusion Matrix:

Here is again, how the confusion matrix works:

- **Upper Left Square:** The amount of **correctly** classified by our model of no fraud transactions.
- **Upper Right Square:** The amount of **incorrectly** classified transactions as fraud cases, but the a
- **Lower Left Square:** The amount of **incorrectly** classified transactions as no fraud cases, but the
- **Lower Right Square:** The amount of **correctly** classified by our model of fraud transactions.

Summary (Keras || Random UnderSampling):

- **Dataset:** In this final phase of testing we will fit this model in both the **random undersampled su**
  in order to predict the final result using the **original dataframe testing data.**
- **Neural Network Structure:** As stated previously, this will be a simple model composed of one in
  equals the number of features) plus bias node, one hidden layer with 32 nodes and one output n
  or 1 (No fraud or fraud).
- **Other characteristics:** The learning rate will be 0.001, the optimizer we will use is the AdamOptir
  in this scenario is "Relu" and for the final outputs we will use sparse categorical cross entropy, w
  instance case is no fraud or fraud (The prediction will pick the highest probability between the tv

↳ 1 cell hidden