UNIFIED MENTOR
YOUR SKILL, SUCCESS & JOURNEY

Project Title	**Credit Card Fraud Detection**
language	Machine learning, python, SQL, Excel
Tools	VS code, Jupyter notebook
Domain	Data Analyst
Project Difficulties level	Advance

Dataset : Dataset is available in the given link. You can download it at your convenience.

[Click here to download data set]

About Dataset

Problem Statement:

Company ABC, a major credit card company, faces challenges with their existing fraud detection system. The current system exhibits slow responsiveness in recognizing new patterns of fraud, leading to significant financial losses. To address this issue, they have contracted us to design and implement an algorithm that can efficiently identify and flag potentially fraudulent transactions for further investigation. The data provided consists of two tables: "cc_info," containing general credit card and cardholder information, and "transactions," containing details of credit card transactions that occurred between August 1st and October 30th.

Objective:

The primary goal of this project is to build an advanced fraud detection system using neural networks to identify transactions that appear unusual and potentially fraudulent. By applying object-oriented programming (OOPs) concepts, we aim to develop a scalable and modular solution that can handle large volumes of data and provide valuable insights to Company ABC.

Data Dictionary

We have two files in our dataset *cc_info.csv* and *transactions.csv*

Here is the column description for cc_info.csv

COLUMN NAME	DESCRIPTION
credit_card	Unique identifier for each transaction.
city	The city where the transaction occurred
state	The state or region where the transaction occurred
zipcode	The postal code of the transaction location
credit_card_limit	The credit limit associated with the credit card used in the transaction

Here is the column description for transactions.csv

COLUMN NAME	DESCRIPTION
credit_card	Unique identifier for each transactions
date	The date of the transaction (between August 1st and October 30th)
transaction_dollar_amount	The dollar amount of the transaction

Long	The longitude coordinate of the transaction location
Lat	The latitude coordinate of the transaction location
Lat	The credit limit associated with the credit card used in the transaction

Here's a comprehensive step-by-step guide and implementation for a credit card fraud detection machine learning project. The project involves data preprocessing, model training using a Random Forest Classifier, and anomaly detection using Isolation Forest.

Project Steps

1. **Data Preprocessing**
2. **Handling Imbalanced Data**
3. **Feature Scaling**
4. **Model Training and Evaluation**

Implementation Code

```
# Importing necessary libraries
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

# Load the dataset
data = pd.read_csv('creditcard.csv')

# Display basic info about the dataset
print(data.info())

# Feature Scaling
scaler = StandardScaler()
data['Amount'] = scaler.fit_transform(data['Amount'].values.reshape(-1, 1))
```

```
data['Time'] = scaler.fit_transform(data['Time'].values.reshape(-1, 1))

# Splitting the dataset
X = data.drop('Class', axis=1)
y = data['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Handling Imbalanced Data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Random Forest Classifier
rfc = RandomForestClassifier(random_state=42)
rfc.fit(X_train_res, y_train_res)
y_pred_rfc = rfc.predict(X_test)

# Evaluation of Random Forest Classifier
print("Random Forest Classifier:")
print(classification_report(y_test, y_pred_rfc))
print("Accuracy:", accuracy_score(y_test, y_pred_rfc))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred_rfc))

# Anomaly Detection using Isolation Forest
iso_forest = IsolationForest(contamination=0.001)
iso_forest.fit(X_train)
y_pred_if = iso_forest.predict(X_test)
y_pred_if = [1 if x == -1 else 0 for x in y_pred_if]

# Evaluation of Isolation Forest
print("\nIsolation Forest:")
print(classification_report(y_test, y_pred_if))
print("Accuracy:", accuracy_score(y_test, y_pred_if))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred_if))
```

Explanation of Code

1. **Data Preprocessing:**
   - Load the dataset using `pandas`.
   - Display basic information about the dataset to understand its structure.
2. **Handling Imbalanced Data:**
   - Use `SMOTE` to oversample the minority class (fraud cases) in the training set.
3. **Feature Scaling:**
   - Standardize the `Amount` and `Time` features using `StandardScaler`.

4. **Model Training:**
    - Split the dataset into training and testing sets using `train_test_split`.
    - Train a `RandomForestClassifier` on the oversampled training data.
    - Predict on the test set and evaluate using accuracy, ROC AUC score, and a classification report.
    - Train an `IsolationForest` for anomaly detection on the training data.
    - Predict anomalies on the test set and evaluate.

Additional Resources

- Credit Card Fraud Detection Dataset on Kaggle
- SMOTE Documentation
- Random Forest Classifier Documentation
- Isolation Forest Documentation

This implementation provides a solid foundation for building and evaluating a credit card fraud detection model using both classification and anomaly detection techniques. Feel free to customize the code to explore other machine learning models or techniques.

SAMPLE CODE

Handling Imbalanced Dataset

Table of Contents

In [1]:

```python
# Importing libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [2]:

```python
# Reading dataset
df=pd.read_csv('/kaggle/input/creditcard/creditcard.csv')
df.head(10)
```

Out[2]:

7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80	0
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	93.20	0
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	3.68	0

10 rows × 31 columns

In [3]:

```python
# shape of dataset
df.shape
```

Out[3]:

```
(284807, 31)
```

In [4]:

```
# null values
df.isnull().sum()
```

Out[4]:

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0

dtype: int64
```

No null Values

In [5]:

```python
# Check if it is balanced or not
df['Class'].value_counts()
```

Out[5]:

```
Class
0    284315
1       492

Name: count, dtype: int64
```

Huge difference between no of 1s and 0s

In [6]:

```python
# create X and Y
X=df.drop('Class', axis=1)
y=df['Class']
X,y
```

Out[6]:

```
(            Time         V1         V2        V3        V4        V5  \
 0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
 1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
 2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
 3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
 4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
 ...          ...        ...        ...       ...       ...       ...
 284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
 284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
 284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
 284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
 284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

               V6        V7        V8        V9  ...       V20       V21  \
 0       0.462388  0.239599  0.098698  0.363787  ...  0.251412 -0.018307
 1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.069083 -0.225775
 2       1.800499  0.791461  0.247676 -1.514654  ...  0.524980  0.247998
 3       1.247203  0.237609  0.377436 -1.387024  ... -0.208038 -0.108300
 4       0.095921  0.592941 -0.270533  0.817739  ...  0.408542 -0.009431
 ...          ...       ...       ...       ...  ...       ...       ...
 284802 -2.606837 -4.918215  7.305334  1.914428  ...  1.475829  0.213454
 284803  1.058415  0.024330  0.294869  0.584800  ...  0.059616  0.214205
 284804  3.031260 -0.296827  0.708417  0.432454  ...  0.001396  0.232045
```

```
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.127434  0.265245
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.382948  0.261057

             V22       V23       V24       V25       V26       V27       V28  \
0       0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1      -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2       0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3       0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4       0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
...          ...       ...       ...       ...       ...       ...       ...
284802  0.111864  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
284803  0.924384  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
284804  0.578229 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
284805  0.800049 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
284806  0.643078  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649

        Amount
0       149.62
1         2.69
2       378.66
3       123.50
4        69.99
...        ...
284802    0.77
284803   24.79
284804   67.88
284805   10.00
284806  217.00

[284807 rows x 30 columns],
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0

Name: Class, Length: 284807, dtype: int64)
```

In [7]:

```python
# import libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV, train_test_split
```

1. Cross Validation like KFold

In [8]:

```
# KFold
log_class=LogisticRegression()
grid={'C':10.0**np.arange(-2,3), 'penalty': ['l1', 'l2']}
cv=KFold(n_splits=5, random_state=None, shuffle=False)
```

In [9]:

```
# train test split
X_train, X_test, y_train, y_test=train_test_split(X,y, train_size=0.7)
```

In [10]:

```
# grid search cv
clf=GridSearchCV(log_class, grid, cv=cv, n_jobs=-1, scoring='f1_macro')
clf.fit(X_train, y_train)
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[10]:

GridSearchCV

estimator: LogisticRegression

LogisticRegression

In [11]:

```
# Prediction and scores
y_pred=clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[85245    42]
 [   52   104]]
0.9988998513628969
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85287
           1       0.71      0.67      0.69       156

    accuracy                           1.00     85443
   macro avg       0.86      0.83      0.84     85443
weighted avg       1.00      1.00      1.00     85443
```

Aim is to decrease False Positive and False Negative

---

2. RandomForest Classifier

In [12]:

```
# train test split
X_train, X_test, y_train, y_test=train_test_split(X,y, train_size=0.7)
```

In [13]:

```
# RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier()
classifier.fit(X_train, y_train)
```

Out[13]:

```
RandomForestClassifier
```

```
RandomForestClassifier()
```

In [14]:

```
# Prediction and scores
y_pred=classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[85288     5]
 [   38   112]]
0.9994967405170698
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85293
           1       0.96      0.75      0.84       150

    accuracy                           1.00     85443
   macro avg       0.98      0.87      0.92     85443
weighted avg       1.00      1.00      1.00     85443
```

In KFold FP, FN were 43, 41. Now it has decreased to 7, 46. So Decision trees are **not much impacted** by imbalanced dataset. After hyperparameter tuning we will get better result.

---

3. RandomForest Classifier with classweight

In [15]:

```
# train test split
X_train, X_test, y_train, y_test=train_test_split(X,y, train_size=0.7)
```

In [16]:

```
y_train.value_counts()
```

Out[16]:

```
Class
0    199020
1       344

Name: count, dtype: int64
```

In [17]:

```
# class weight, give more weight for 1 as no of 1s are less
class_weight=dict({0:1, 1: 100})
class_weight
```

Out[17]:

```
{0: 1, 1: 100}
```

In [18]:

```
# RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier(class_weight=class_weight)
classifier.fit(X_train, y_train)
```

Out[18]:

RandomForestClassifier

```
RandomForestClassifier(class_weight={0: 1, 1: 100})
```

In [19]:

```
# Prediction and scores
y_pred=classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[85287     8]
 [   34   114]]
0.9995084442259752
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.93      0.77      0.84       148

    accuracy                           1.00     85443
   macro avg       0.97      0.89      0.92     85443
weighted avg       1.00      1.00      1.00     85443
```

Previously FP, FN was 7, 46, now it is 6, 46 .It has improved using class-weight.

---

4. Under Sampling

When to use:

Under-sampling is useful when you have a **large dataset**, and the **majority class significantly dominates** the minority class.

Advantages:

Helps **reduce the size of the majority class**, making the dataset more balanced. Can lead to **faster training times**, especially when dealing with large datasets.

Disadvantages:

Loss of potentially valuable information from the majority class. May **increase the risk of overfitting** on the reduced dataset.

In [20]:

```
# Installing library
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in
/opt/conda/lib/python3.10/site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in
/opt/conda/lib/python3.10/site-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in
/opt/conda/lib/python3.10/site-packages (from imbalanced-learn) (1.11.2)
Requirement already satisfied: scikit-learn>=1.0.2 in
/opt/conda/lib/python3.10/site-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in
/opt/conda/lib/python3.10/site-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.10/site-packages (from imbalanced-learn) (3.1.0)
```

In [21]:

```
# train test split
X_train, X_test, y_train, y_test=train_test_split(X,y, train_size=0.7)
```

In [22]:

```
# import library
from collections import Counter
from imblearn.under_sampling import NearMiss
```

In [23]:

```
# performing fit
ns=NearMiss()
X_res,Y_res=ns.fit_resample(X_train, y_train)
print('NO of class before fit ',Counter(y_train))
print('NO of class after fit ',Counter(Y_res))
```

```
NO of class before fit  Counter({0: 199023, 1: 341})
NO of class after fit  Counter({0: 341, 1: 341})
```

No of 0s are decreased to no of 1s.

In [24]:

```
# RandomForestClassifier
```

```python
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier()
classifier.fit(X_res, Y_res)
```

Out[24]:

```
RandomForestClassifier
```

```
RandomForestClassifier()
```

In [25]:

```python
# Prediction and scores
y_pred=classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[57841 27451]
 [   13   138]]
0.6785693386234097
              precision    recall  f1-score   support

           0       1.00      0.68      0.81     85292
           1       0.01      0.91      0.01       151

    accuracy                           0.68     85443
   macro avg       0.50      0.80      0.41     85443
weighted avg       1.00      0.68      0.81     85443
```

Very bad accuracy, so not preferred.

---

5. Over Sampling

When to use:

Over-sampling is beneficial when you have a **small dataset** and the minority class is underrepresented.

Advantages:

Helps **increase the size of the minority class**, balancing the dataset. **Mitigates the risk of losing important information** from the majority class.

Disadvantages:

Can **lead to overfitting**, as it **duplicates the minority class** examples. May result in **increased training time** due to the larger dataset.

In [26]:

```
# train test split
X_train, X_test, y_train, y_test=train_test_split(X,y, train_size=0.7)
```

In [27]:

```
# import library
from imblearn.over_sampling import RandomOverSampler
```

In [28]:

```
# performing fit
ns=RandomOverSampler()
X_res,Y_res=ns.fit_resample(X_train, y_train)
print('NO of class before fit ',Counter(y_train))
print('NO of class after fit ',Counter(Y_res))
```

```
NO of class before fit  Counter({0: 199021, 1: 343})
NO of class after fit  Counter({0: 199021, 1: 199021})
```

Here no of 1s are increased.

In [29]:

```python
# RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier()
classifier.fit(X_res, Y_res)
```

Out[29]:

RandomForestClassifier

```
RandomForestClassifier()
```

In [30]:

```python
# Prediction and scores
y_pred=classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[85289     5]
 [   30   119]]
0.9995903701883126
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85294
           1       0.96      0.80      0.87       149

    accuracy                           1.00     85443
   macro avg       0.98      0.90      0.94     85443
weighted avg       1.00      1.00      1.00     85443
```

Confusion matrix score has improved now.

---

6. SMOTETomek

When to use:

SMOTETomek is suitable when you want to **address class imbalance** while simultaneously **cleaning noisy or borderline examples**. It's especially useful when you **suspect that there are noisy samples or overlapping classes** in your dataset. You might use SMOTETomek when you have a **relatively low-dimensional feature space**.

Advantages:

It **combines the strengths** of both over-sampling (**SMOTE**) and under-sampling (**Tomek links**) techniques. **SMOTE generates synthetic examples** for the minority class, making it larger. Tomek links are used to **remove noisy and borderline examples** that could potentially confuse the classifier. It helps in **improving the balance of the dataset** while **reducing noise** and potentially enhancing the classifier's performance.

Disadvantages:

Like SMOTE, the effectiveness of SMOTETomek **can vary based on the dataset's characteristics**. It may require parameter tuning to balance the trade-off between over-sampling and under-sampling.

In [31]:

```
# train test split
X_train, X_test, y_train, y_test=train_test_split(X,y, train_size=0.7)
```

In [32]:

```
# import library
from imblearn.combine import SMOTETomek
```

In [33]:

```
# performing fit
ns=SMOTETomek()
X_res,Y_res=ns.fit_resample(X_train, y_train)
```

In [34]:

```
# print No of class before and after
print('NO of class before fit ',Counter(y_train))
```

```
print('NO of class after fit ',Counter(Y_res))
```

```
NO of class before fit  Counter({0: 199026, 1: 338})
NO of class after fit  Counter({0: 198339, 1: 198339})
```

It takes more time.

In [35]:

```
# RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier()
classifier.fit(X_res, Y_res)
```

Out[35]:

RandomForestClassifier

```
RandomForestClassifier()
```

In [36]:

```
# Prediction and scores
y_pred=classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[85261    28]
 [   20   134]]
0.9994382219725431
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85289
           1       0.83      0.87      0.85       154

    accuracy                           1.00     85443
   macro avg       0.91      0.93      0.92     85443
weighted avg       1.00      1.00      1.00     85443
```

7. Easy Ensemble

When to use:

Easy Ensemble is a good choice when you have a **highly imbalanced dataset** and you are **willing to invest computational resources** to create balanced subsets.

Advantages:

Creates **multiple balanced subsets** of the dataset. **Reduces the risk of overfitting** and **provides robustness**.

Disadvantages:

Requires **multiple iterations and models**, which can be **computationally expensive**. **May not be suitable for very large datasets**.

In [37]:

```
# train test split
X_train, X_test, y_train, y_test=train_test_split(X,y, train_size=0.7)
```

In [38]:

```
# import library
from imblearn.ensemble import EasyEnsembleClassifier
```

In [39]:

```
eec = EasyEnsembleClassifier(random_state=42)
eec.fit(X_train, y_train)
```

Out[39]:

EasyEnsembleClassifier

```
EasyEnsembleClassifier(random_state=42)
```

In [40]:

```python
# Prediction and scores
y_pred=eec.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[82742  2550]
 [   16   135]]
0.9699682829488665
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     85292
           1       0.05      0.89      0.10       151

    accuracy                           0.97     85443
   macro avg       0.53      0.93      0.54     85443
weighted avg       1.00      0.97      0.98     85443
```

[Reference link](#)