VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering
J Component report
Programme : Int. Mtech CSE with BA
Course Title : REAL TIME ANALYTICS
Course code: CSE3069
Slot: E2+TE2
TITLE: CREDIT CARD FRAUD DETECTION USING PYTHON

Team members:
SWETHA B (20MIA1101)

SHREYA ALAJANGI (20MIA1172)

FACULTY: PROF.VALARMATHI S

TITLE: CREDIT CARD FRAUD DETECTION USING PYTHON

Abstract:

Credit card fraud detection is the process of identifying fraudulent transactions made using credit or debit cards. With the increasing usage of credit and debit cards for online and offline transactions, the risk of fraud has also increased. Fraudulent activities, such as stolen cards, identity theft, and unauthorized transactions, can cause significant financial losses to individuals and businesses. Credit card fraud detection involves the use of various statistical and machine learning techniques to analyze transactional data and detect any unusual or fraudulent activities. The analysis typically involves identifying patterns, trends, and anomalies in the data that could indicate potential fraud. Some common techniques used for credit card fraud detection include supervised and unsupervised machine learning algorithms, anomaly detection

Introduction:

In this project we will use various predictive models to see how accurate they are in detecting whether a transaction is a normal payment or a fraud and find the best one. Financial institutions and card issuers have adopted a comprehensive approach to security that tackles fraud on four fronts. The first step is to devalue sensitive information to make it less useful if it falls into the wrong hands. An example is tokenization, which "converts credit card numbers into randomly-generated values (tokens)." A token is a unique number relating to a specific transaction and has no use beyond that transaction. Consequently, a cybercriminal will find the token data he has obtained to be worthless. Second, there is increasing reliance on the analysis of data, with the object of detecting unusual patterns, for example where the location of the transaction differs from that of the cardholder's mobile phone. Third, pushing businesses and others in the payment system to observe industry protocols, such as the Payment Card Industry (PCI) standards, for the protection of data. For example, under PCI standards, neither point-of-sale (POS) terminals nor a business's own records, can store consumer's credit card numbers. Finally, alerting consumers to the dangers of card fraud and encouraging them to adopt best practices, such as monitoring their accounts regularly. Credit card fraud detection systems use a combination of statistical and machine learning techniques to analyze transactional data and identify patterns, trends, and anomalies that indicate potential fraud. Some common techniques include supervised and unsupervised machine learning algorithms, anomaly detection, and neural networks. These techniques help detect fraudulent activities in real-time and allow credit card issuers to take appropriate actions to prevent further losses. Effective credit card fraud detection systems can help reduce financial losses for financial institutions, merchants, and consumers. However, it is important to balance the need for fraud detection with the need for minimal disruption to legitimate transactions. False positives can be costly and can negatively impact customer experience. Therefore, credit card issuers must continuously refine their fraud detection systems to improve accuracy and minimize disruption to legitimate transactions.

Data Description:

Credit card fraud detection is the process of identifying and preventing fraudulent transactions made using credit cards. The data used for credit card fraud detection typically includes information about the transactions made using credit cards, as well as information about the cardholders and the merchants involved in the transactions.

The data includes the following features:

- Transaction amount: The amount of money involved in the transaction.
- Transaction date and time: The date and time at which the transaction was made.
- Cardholder information: Information about the cardholder, such as their name, addresses, and contact details.
- Card information: Information about the credit card, such as the card number, expiration date, and security code.
- Merchant information: Information about the merchant, such as their name, addresses, and contact details.
- Transaction type: The type of transaction, such as online purchase, in-store purchase, or cash withdrawal.
- Transaction status: Whether the transaction was approved or declined.
- Fraud status: Whether the transaction was fraudulent or not.

The goal of credit card fraud detection is to develop machine learning models that can accurately identify fraudulent transactions while minimizing false positives i.e., identifying legitimate transactions as fraudulent. This requires the use of advanced machine learning techniques such as ensemble methods, feature engineering, and oversampling of the minority class (fraudulent transactions).

Types of credit card frauds

*counterfeit fraud

*lost or stolen card fraud

*card does not present fraud

*identity fraud

* skimming fraud

To detect these counterfeit, stolen cards and card not present fraud classifiers like logistics regression, SVM, decision tree and random forest are very helpful

To Detect identity and skimming fraud anomaly detection and classification algorithms are very helpful

Our Goals:

- credit card fraud detection using machine learning algorithms is to identify fraudulent credit card transactions with high accuracy and speed
- Machine learning algorithms are used to analyze large volumes of transaction data in real-time to detect patterns and anomalies that may indicate fraudulent activity
- Scalability and Adaptability ML algorithms to handle large volumes of data and to adapt changing fraud patterns and techniques

Outline:

I. Understanding our data
II. Preprocessing
III. Machine learning algorithms and models
IV. Training and Testing
V. Result and Conclusion

Dataset:

The dataset (Creditcard.csv) presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2 … V28 are the principal components obtained with PCA; the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Fraudulent Credit Card Transactions Detection

```
# load libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
#from sklearn.ensemble import AdaBoostClassifier
```

Let's read the data into a Pandas data frame that we will call by the generic name "df":

```
[2] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] df = pd.read_csv("/content/drive/MyDrive/creditcard.csv")
```

```
[4] df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.

```
[5] df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
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+0
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+0
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+0
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-0
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-0
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-0

We note that the variables V1 to V28 have mean zero, with a standard deviation in the range of 0.3 to 2.

Preprocessing

```
# check for missing values
df.isnull().sum()
```

```
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
```

It appears that the only categorical variable is the response variable "Class". All the other variables are continuous numerical variables.

```
[8] df['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

```
[9] percentage_fraudulent = 100 * df[df.Class == 1].shape[0]/df[df.Class == 0].shape[0]
    print('The percentage of fraudulent transactions is : %.2f percent' % percentage_fraudulent )
```

```
The percentage of fraudulent transactions is : 0.17 percent
```

```
[10] features_list = list(df.columns)
     features_list.remove("Class")
     print(features_list)
```

```
corr_Class = df.corr()['Class']
print(corr_Class)
```

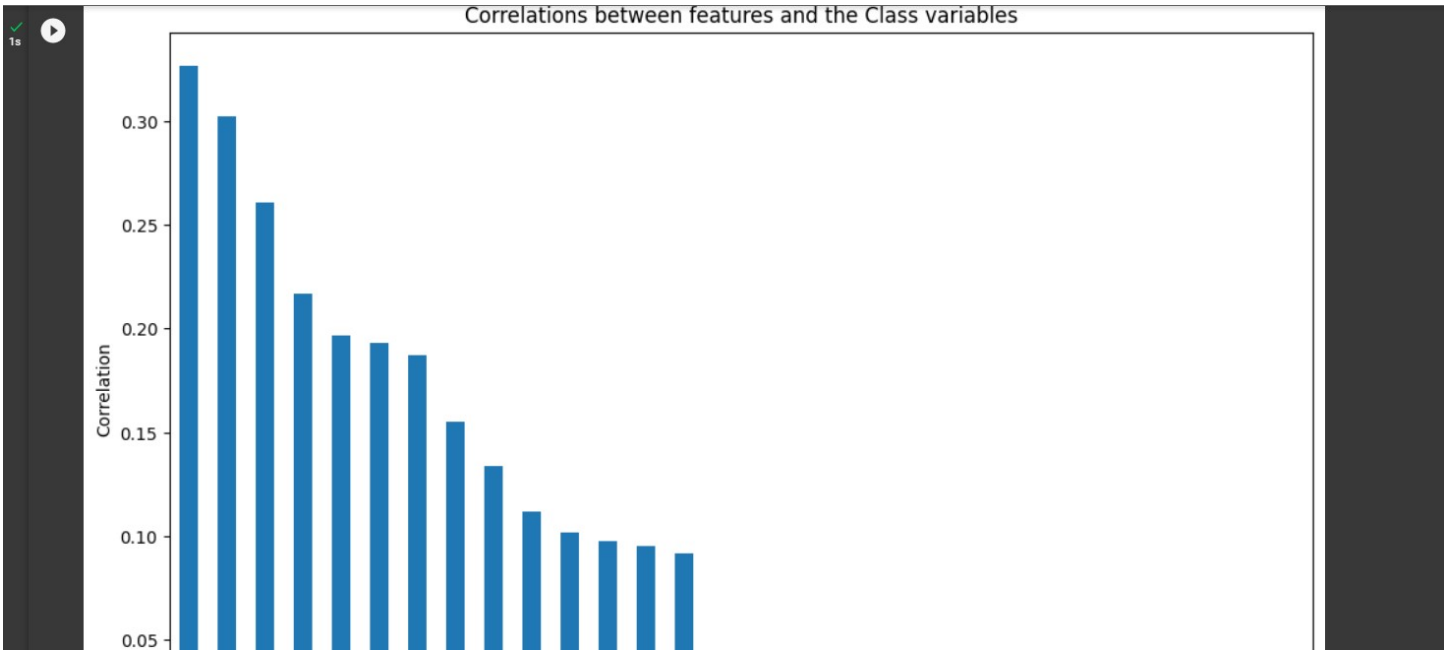
```
Time     -0.012323
V1       -0.101347
V2        0.091289
V3       -0.192961
V4        0.133447
V5       -0.094974
V6       -0.043643
V7       -0.187257
V8        0.019875
V9       -0.097733
V10      -0.216883
V11       0.154876
V12      -0.260593
V13      -0.004570
V14      -0.302544
V15      -0.004223
V16      -0.196539
V17      -0.326481
V18      -0.111485
V19       0.034783
V20       0.020090
V21       0.040413
V22       0.000805
V23      -0.002685
V24      -0.007221
```

Let's remove the last element ('Class') from the above list, and take the absolute value of the results and then sort the values:

```
[12] corr_Class = corr_Class[:-1].abs().sort_values(ascending = False)
     corr_Class
```

```
V17       0.326481
V14       0.302544
V12       0.260593
V10       0.216883
V16       0.196539
V3        0.192961
V7        0.187257
V11       0.154876
V4        0.133447
V18       0.111485
V1        0.101347
V9        0.097733
V5        0.094974
V2        0.091289
V6        0.043643
V21       0.040413
V19       0.034783
V20       0.020090
V8        0.019875
V27       0.017580
Time      0.012323
V28       0.009536
V24       0.007221
Amount    0.005632
```

```
plt.figure(figsize=(12,7.5))
corr_Class.plot(kind='bar')
plt.xlabel('Feature')
plt.ylabel('Correlation')
plt.title('Correlations between features and the Class variables')
```



From the above barplot, we can see that there is a significant jump in correlation values between the variables V2 and V6. It therefore makes sense to use the value of the correlation of the V2 variable as a cut-off for the features we want to keep:

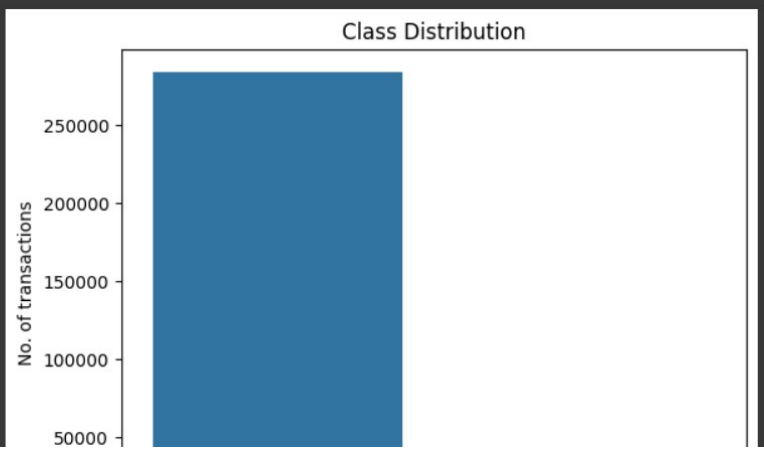
```
[13] relevant_features = list(corr_Class[np.abs(corr_Class) > 0.09].index)
     print(relevant_features)

['V17', 'V14', 'V12', 'V10', 'V16', 'V3', 'V7', 'V11', 'V4', 'V18', 'V1', 'V9', 'V5', 'V2']
```

We now want to compare the above features list to the tentative list of features

```
tentative_features_list = ['V2', 'V3', 'V4', 'V7','V9', 'V10', 'V11', 'V12', 'V14', 'V16', 'V17', 'V18']
set(relevant_features) - set(tentative_features_list)
```

```
[14] X = df.loc[:, relevant_features]
     y = df['Class']
     print('Shape of X: ' , X.shape)
     print('Shape of y: ' , y.shape)

     Shape of X:  (284807, 14)
     Shape of y:  (284807,)

[15] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
     print('Shape of X_train: ' , X_train.shape)
     print('Shape of y_train: ' , y_train.shape)
     print('\nShape of X_test: ' , X_test.shape)
     print('Shape of y_test: ' , y_test.shape)

     Shape of X_train:  (213605, 14)
     Shape of y_train:  (213605,)

     Shape of X_test:  (71202, 14)
     Shape of y_test:  (71202,)

[16] from sklearn.preprocessing import StandardScaler
```

```
[18] sns.countplot(x = 'Class', data = df)
     # df["Class"].value_counts().plot(kind = "bar")
     plt.title("Class Distribution")
     plt.xlabel("Class")
     plt.ylabel("No. of transactions")
     plt.show()
```



Out of 284807 transactions, only 492 are fraudulent.

```
[20] # define predictors and target
     target = "Class"
     # all columns except class
     predictors = df.columns.tolist()[:-1]
```

```
[21] # split the data
     train_df, test_df = train_test_split(df, test_size = 0.2, random_state = 42, shuffle = True)
     train_df, valid_df = train_test_split(train_df, test_size = 0.2, random_state = 42, shuffle = True)
```

Random forest classifier

Random forests are an ensemble learning method that combines multiple decision trees to improve the accuracy of the model. Random forests are known for their high accuracy and robustness and can be used to detect counterfeit fraud by identifying patterns in the data that suggest fraudulent activity, such as multiple transactions at the same merchant or transactions with unusual amounts.

Random Forest classifier

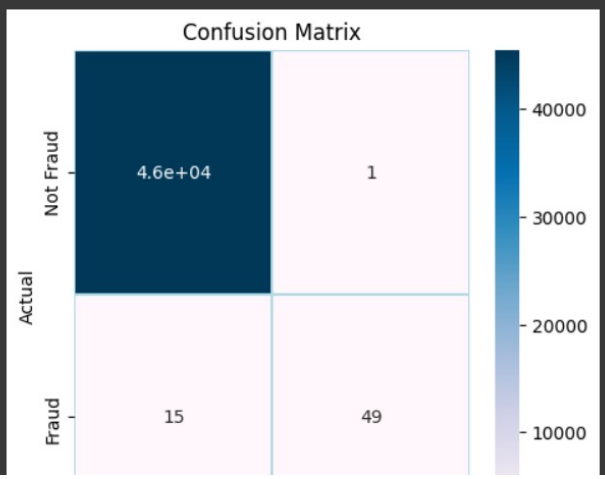
+ Code + Text

```
[23] clf = RandomForestClassifier(n_jobs = 4,
                                  random_state = 42,
                                  criterion = "gini",
                                  n_estimators = 100,
                                  verbose = False)
```

```
[24] clf.fit(train_df[predictors], train_df[target].values)
```

```
RandomForestClassifier
RandomForestClassifier(n_jobs=4, random_state=42, verbose=False)
```

```
[27] # confusion matrix
     cm = pd.crosstab(valid_df[target].values, preds, rownames = ['Actual'], colnames = ['Predicted'])
     fig, ax1 = plt.subplots(ncols = 1, figsize = (5, 5))
     sns.heatmap(cm, xticklabels = ["Not Fraud", "Fraud"], yticklabels = ["Not Fraud", "Fraud"], annot = True, ax = ax1, linewidth = .2, lineco
     plt.title("Confusion Matrix")
     plt.show()
```




```
[ ] # area under curve
    from sklearn.metrics import roc_auc_score
    roc_auc_score(valid_df[target].values, preds)
```

The precision, being the percentage of predicted positives that were correctly classified, is 0.88: out of the predicted positives (84 + 9 = 93), 84 were correctly classified, which represents 90% of all predicted positives. In other words, 1 out of 10 predicted positives is a false positive.

The recall, being the percentage of actual positives that were correctly classified, is 0.74: out of the actual positives (84 + 29 = 113), 84 were correctly classified, which represents 74% of all actual positives. In other words, 3 out of every 4 fraudulent transactions are caught, with the remaining 1 out of every 4 transactions remaining undetected.

Logistic Regression:

Logistic regression is a statistical method that predicts a binary outcome based on the input features. In the case of credit card fraud detection, the binary outcome is either fraud or non-fraud. Logistic regression can be useful in identifying counterfeit fraud because it can identify patterns in the data that suggest fraudulent activity, such as unusual spending patterns or transactions outside of the user's usual geographic location.

Logistic regression

```
[29] from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
[30] X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-1], df.iloc[:,-1], test_size=0.3, random_state=42)
```

```
[31] # Train a logistic regression model
     model = LogisticRegression()
     model.fit(X_train, y_train)

     # Evaluate the model on the testing set
     y_pred = model.predict(X_test)
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
     print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9988998513628969
Confusion Matrix:
 [[85259    48]
 [   46    90]]
Classification Report:
              precision    recall  f1-score   support
```

```
roc_auc_score(y_test, y_pred)
```

```
0.8764023313030652
```

The performance of the logistic regression is not as good as the random forest classifier: the precision is lower, as is the recall.

Decision tree Classifier

Decision trees are another popular algorithm used for classification problems, including credit card fraud detection. They work by dividing the data into smaller and smaller subsets based on the input features until the subsets are homogenous. Decision trees are easy to interpret and explain, making them a popular choice for credit card fraud detection. The decision tree model can be used to identify the most important features in predicting whether a transaction is fraudulent or not. By understanding which features are most important, analysts can develop strategies to reduce the risk of fraud by monitoring and verifying these features more closely.

```
[ ] from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
    from sklearn.model_selection import train_test_split
```

```
[33] # Split the dataset into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-1], df.iloc[:,-1], test_size=0.3, random_state=42)
```

```
[34]
     # Train a decision tree model
     model = DecisionTreeClassifier()
     model.fit(X_train, y_train)

     # Evaluate the model on the testing set
     y_pred = model.predict(X_test)
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
     print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9990988144142879
Confusion Matrix:
 [[85259    48]
 [   29   107]]
Classification Report:
              precision    recall  f1-score   support
```

decision tree classifiers have been shown to be effective in detecting credit card fraud, achieving high accuracy and reducing the number of false positives.

Anomaly Detection

Anomaly detection algorithms can be used to detect unusual patterns or outliers in credit card transaction data that may be indicative of skimming fraud. These algorithms can be trained on a dataset of known legitimate transactions to identify patterns that are consistent with normal behavior. Any transactions that fall outside of these patterns can be flagged as potentially fraudulent and investigated further.

```
Anomaly detection
```

```
[40] # Fit an Elliptic Envelope model on the training set
     from sklearn.covariance import EllipticEnvelope
     model = EllipticEnvelope(contamination=0.01)
     model.fit(X_train)

     # Predict the labels for the testing set
     y_pred = model.predict(X_test)

     # Compute the accuracy score
```

```
[40]   warnings.warn(
     /usr/local/lib/python3.9/dist-packages/sklearn/covariance/_robust_covariance.py:184: RuntimeWarning: Det
       warnings.warn(
     /usr/local/lib/python3.9/dist-packages/sklearn/covariance/_robust_covariance.py:184: RuntimeWarning: Det
       warnings.warn(
     /usr/local/lib/python3.9/dist-packages/sklearn/covariance/_robust_covariance.py:184: RuntimeWarning: Det
       warnings.warn(
     /usr/local/lib/python3.9/dist-packages/sklearn/covariance/_robust_covariance.py:184: RuntimeWarning: Det
```

```
[83] roc_auc_score(y_test, y_pred)
```

We observe that the accuracy for anomaly detection is 87%

AdaBoost Classifier

AdaBoost (Adaptive Boosting) is a popular ensemble learning algorithm that can be used for classification problems, including credit card fraud detection. The main idea behind AdaBoost is to combine multiple "weak" classifiers into a "strong" classifier. In the context of credit card fraud detection, a weak classifier could be a simple decision tree or logistic regression model that is not very accurate on its own but can still provide some predictive power.

AdaBoost

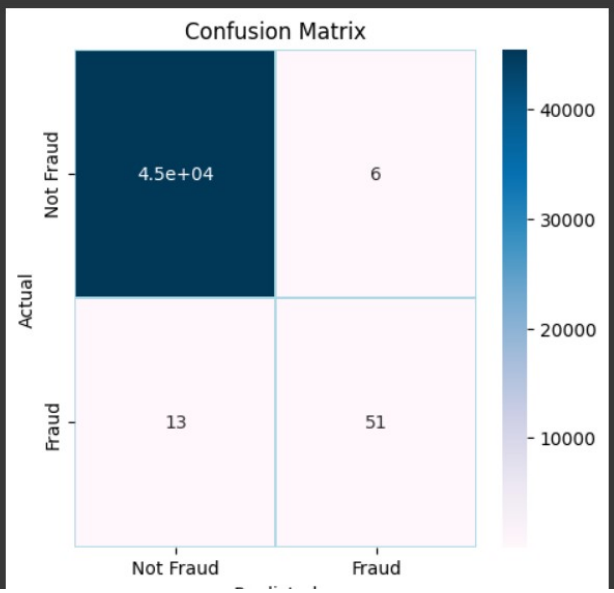
```
[45] from sklearn.ensemble import AdaBoostClassifier
```

```
[46] X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-1], df.iloc[:,-1], test_size=0.3, random_state=42)
```

```
[48] clf = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1), n_estimators=100)
     clf.fit(X_train, y_train)
     y_pred = clf.predict(X_test)
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
     print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in versi
  warnings.warn(
Accuracy: 0.9994850368081645
Confusion Matrix:
 [[85291    16]
 [   28   108]]
Classification Report:
              precision    recall  f1-score   support
```

```
[50] sns.heatmap(cm, xticklabels = ["Not Fraud", "Fraud"], yticklabels = ["Not Fraud", "Fraud"], annot = True, ax = ax1, linewidth = .2, linec
     plt.title("Confusion Matrix")
     plt.show()
```



CatBoost Classifier

CatBoost is a popular gradient boosting framework used in machine learning that can be applied to various applications, including credit card fraud detection. CatBoost, with its ability to handle categorical features, can be particularly useful for fraud detection tasks that involve transaction data.

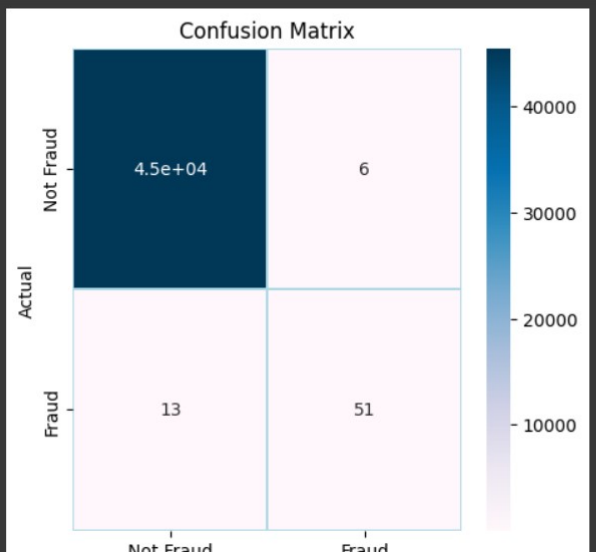
```
[59] from catboost import CatBoostClassifier
     clf = CatBoostClassifier(iterations = 500, learning_rate = 0.02, depth = 12,
                              eval_metric = "AUC", random_seed = 42, bagging_temperature = 0.2,
                              od_type = "Iter", metric_period = 50, od_wait = 100)
```

```
[61] X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-1], df.iloc[:,-1], test_size=0.3, random_state=42)
```

```
[62] model = CatBoostClassifier(iterations=1000, learning_rate=0.1, depth=6, loss_function='Logloss', eval_metric='AUC', random_seed=
     model.fit(X_train, y_train, verbose=False)
     y_pred = model.predict(X_test)
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
     print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9996254813150287
Confusion Matrix:
 [[85298     9]
 [   23   113]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
```

```
# confusion matrix
cm = pd.crosstab(y_valid, preds, rownames = ['Actual'], colnames = ['Predicted'])
fig, ax1 = plt.subplots(ncols = 1, figsize = (5, 5))
sns.heatmap(cm, xticklabels = ["Not Fraud", "Fraud"], yticklabels = ["Not Fraud", "Fraud"], annot = True, ax = ax1, linewidth = .2, linecolor
plt.title("Confusion Matrix")
plt.show()
```



XGBoost Classifier

XGBoost is a widely-used gradient boosting framework in machine learning that can also be applied to credit card fraud detection. Credit card fraud detection typically involves identifying fraudulent transactions by analyzing transaction data. XGBoost can be useful in this application as it is capable of handling complex, high-dimensional data and can be effective in identifying patterns and anomalies in transaction data.

XGBoost

```
[70] from xgboost import XGBClassifier
     X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-1], df.iloc[:,-1], test_size=0.3, random_state=42)
```

```
[72] xgb_clf = xgb.XGBClassifier()
     xgb_clf.fit(X_train, y_train)
     y_pred = xgb_clf.predict(X_test)
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
     print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9996722961506501
Confusion Matrix:
 [[85301     6]
 [   22   114]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.95      0.84      0.89       136

    accuracy                           1.00     85443
   macro avg       0.97      0.92      0.95     85443
weighted avg       1.00      1.00      1.00     85443
```

XGBoost seems to have the best balance between precision and recall, with a precision score of ~0.9 (1 out of 10 predicted positives is a false positive), and a recall score of ~ 0.90 (22% of fraudulent transactions go undetected).

LightGBM

LightGBM is another popular gradient boosting framework that can be applied to credit card fraud detection. Like XGBoost and CatBoost, it is capable of handling large datasets and can effectively handle high-dimensional and sparse data. In credit card fraud detection, LightGBM can be useful for identifying patterns and anomalies in transaction data. Similar to using XGBoost or CatBoost, we can prepare the data and select relevant features.

LightGBM

```
[75] import lightgbm as lgb
     X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-1], df.iloc[:,-1], test_size=0.3, random_state=42)
```

```
[76] params = {
         'objective': 'binary',
         'boosting_type': 'gbdt',
         'metric': 'auc',
         'num_leaves': 31,
         'learning_rate': 0.05,
         'feature_fraction': 0.9
     }
```

```
[79] train_data = lgb.Dataset(X_train, label=y_train)
     model = lgb.train(params, train_data, num_boost_round=100)
     y_pred = model.predict(X_test)
     y_pred = np.where(y_pred > 0.5, 1, 0)
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
     print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
[LightGBM] [Info] Number of positive: 356, number of negative: 199008
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.066885 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 199364, number of used features: 30
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.21      0.76      0.33       136

    accuracy                           1.00     85443
   macro avg       0.60      0.88      0.66     85443
weighted avg       1.00      1.00      1.00     85443
```

```
[ ] # area under curve
```

This has shown a promising solution for credit card fraud detection and has demonstrated high accuracy and efficiency in various studies and competitions.

Results:

Discussion and Conclusion:

In order to get a feel of how good our classification models were, let us compare the performance of these models

By comparison, our best models were the following:

- Random forest with balanced class weights: 27% precision and 89% recall.
- A logistic regression with balanced class weights: 5% precision and 91% recall. This is the best recall we obtained in all the models we have studied. The drawback is the precision, which is quite small (note that this model is at least as good as the neural network with oversampling referenced above).
- Decision tree classifier with balanced class weights: 69% precision and 79% recall. This is also the best recall we obtained in all the models we have studied. And also the precision we got is also good.
- Anomaly detection , we observe that the accuracy for anomaly detection is 87%
- Adaboost classifier with balanced class weights: 87% precision and 79% recall. This is also the best recall we obtained in all the models we have studied. And also the precision we got is also good.
- CatBoost classifier with balanced class weights: 93% precision and 83% recall. This is also the best recall we obtained in all the models we have studied. And also the precision we got is also good.
- XGBoost classifier with balanced class weights: 95% precision and 84% recall. This is also the best recall we obtained in all the models we have studied. And also the precision we got is also good.