Credit Card Fraud Detection

Abstract

This project aims to classify credit card transactions as fraudulent or non-fraudulent using historical data. By employing various machine learning algorithms, the project demonstrates a comparative analysis to identify the most accurate model for detecting fraudulent transactions, achieving high accuracy and balanced performance.

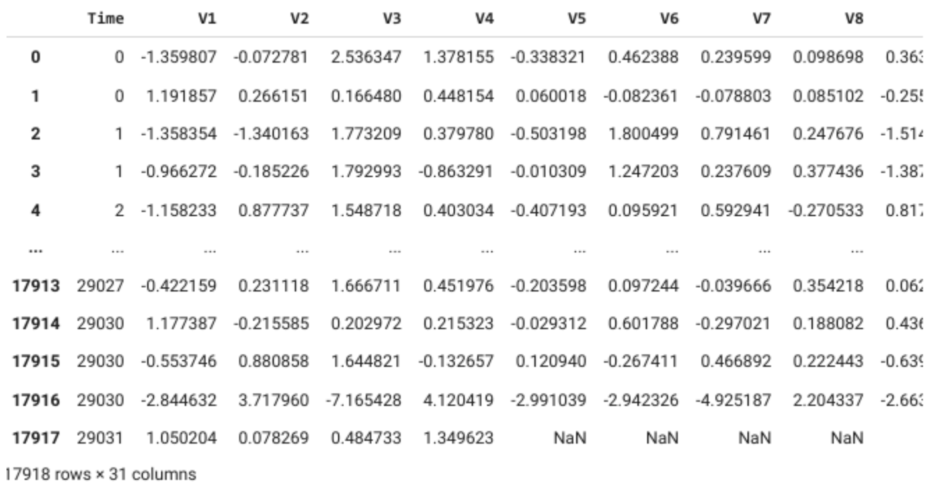
1. Introduction

Credit card fraud has become a significant concern with the rise of online transactions. Detecting fraudulent transactions is critical for financial institutions to prevent monetary losses and ensure customer trust. Machine learning provides powerful tools to identify patterns in large datasets, allowing for the prediction of potentially fraudulent behavior based on historical transaction data. This project investigates the use of supervised learning techniques to tackle this challenge.

2. Problem Definition

To classify transactions as fraudulent (Class 1) or non-fraudulent (Class 0) using the "Credit Card Fraud Detection" dataset available on Kaggle. The problem is characterized by extreme class imbalance, with fraudulent transactions constituting only a small fraction of the total dataset. This necessitates specialized preprocessing and evaluation techniques to ensure the models' effectiveness in real-world scenarios.

3. Dataset Description



	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81
...	...	...	...	...	...	...	...	...	...	
17913	29027	-0.422159	0.231118	1.666711	0.451976	-0.203598	0.097244	-0.039666	0.354218	0.06
17914	29030	1.177387	-0.215585	0.202972	0.215323	-0.029312	0.601788	-0.297021	0.188082	0.43
17915	29030	-0.553746	0.880858	1.644821	-0.132657	0.120940	-0.267411	0.466892	0.222443	-0.63
17916	29030	-2.844632	3.717960	-7.165428	4.120419	-2.991039	-2.942326	-4.925187	2.204337	-2.66
17917	29031	1.050204	0.078269	0.484733	1.349623	NaN	NaN	NaN	NaN	

17918 rows × 31 columns

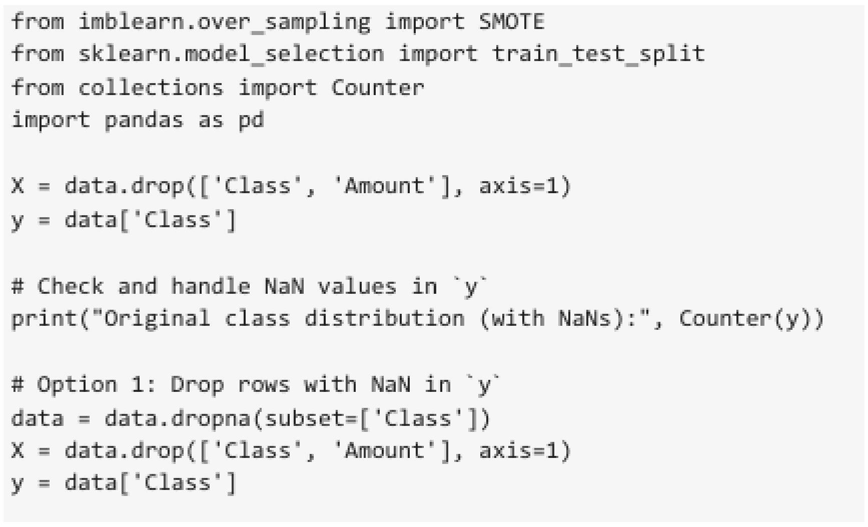
The dataset contains 31 columns and 17918 entries:

- **Features V1-V28**: Principal components derived from PCA, representing anonymized transaction details.
- **Time**: Time elapsed since the first transaction.
- **Amount**: Transaction amount, representing the financial magnitude of the transaction.
- **Class**: Target variable (0 = Non-Fraud, 1 = Fraud).

This dataset is widely used in research on fraud detection due to its realistic distribution of fraud cases and anonymized features, preserving privacy.

4. Data Preprocessing

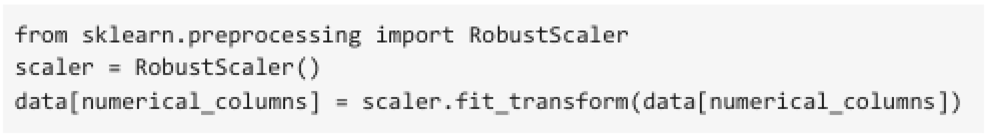
4.1 Data Cleaning



```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from collections import Counter
import pandas as pd

X = data.drop(['Class', 'Amount'], axis=1)
y = data['Class']

# Check and handle NaN values in `y`
print("Original class distribution (with NaNs):", Counter(y))

# Option 1: Drop rows with NaN in `y`
data = data.dropna(subset=['Class'])
X = data.drop(['Class', 'Amount'], axis=1)
y = data['Class']
```

- Removed rows with null values to ensure data integrity.
- Checked for duplicate entries and inconsistencies to prevent redundant processing.

4.2 Scaling



```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
```

- Used RobustScaler to scale numerical features. This scaler is resilient to outliers, making it suitable for datasets with extreme values.

4.3 Addressing Class Imbalance

```
# Apply SMOTE to handle class imbalance
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print("Resampled class distribution:", Counter(y_train_resampled))
```

- Applied SMOTE (Synthetic Minority Oversampling Technique) to balance the dataset. This technique generates synthetic examples for the minority class, enhancing the model's ability to learn patterns specific to fraudulent transactions.

5. Exploratory Data Analysis (EDA)

5.1 Class Distribution

```
class_counts = data['Class'].value_counts()

print("Count of Non-Fraud Transactions (Class 0):", class_counts[0])
print("Count of Fraudulent Transactions (Class 1):", class_counts[1])
```

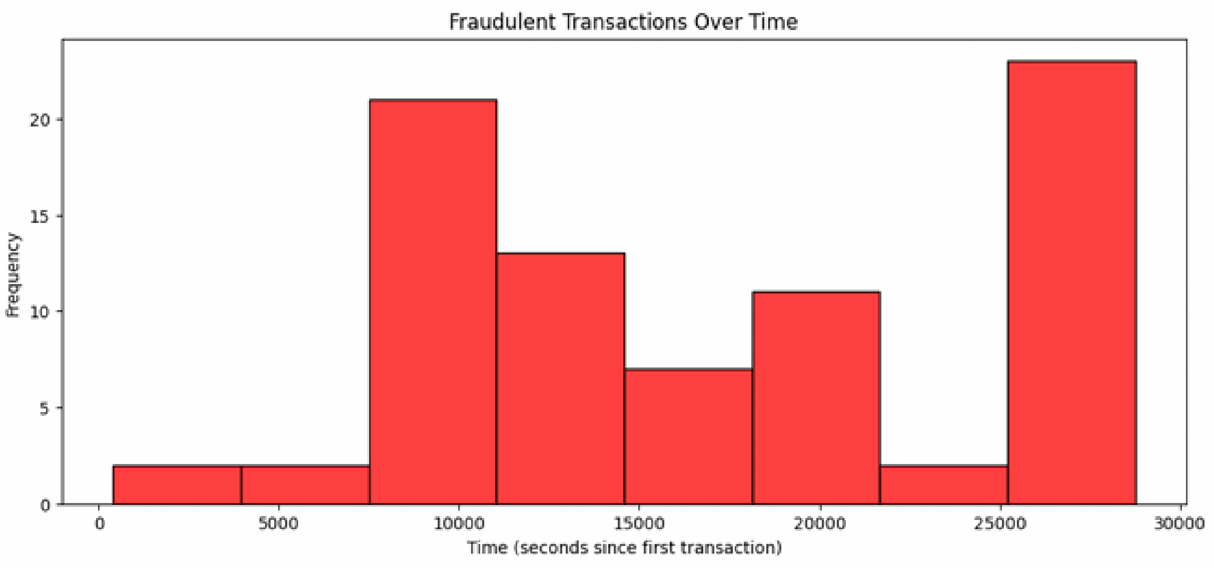
```
Count of Non-Fraud Transactions (Class 0): 17836
Count of Fraudulent Transactions (Class 1): 81
```

The dataset exhibits a highly imbalanced class distribution:

- Non-Fraud: 17836
- Fraud: 81

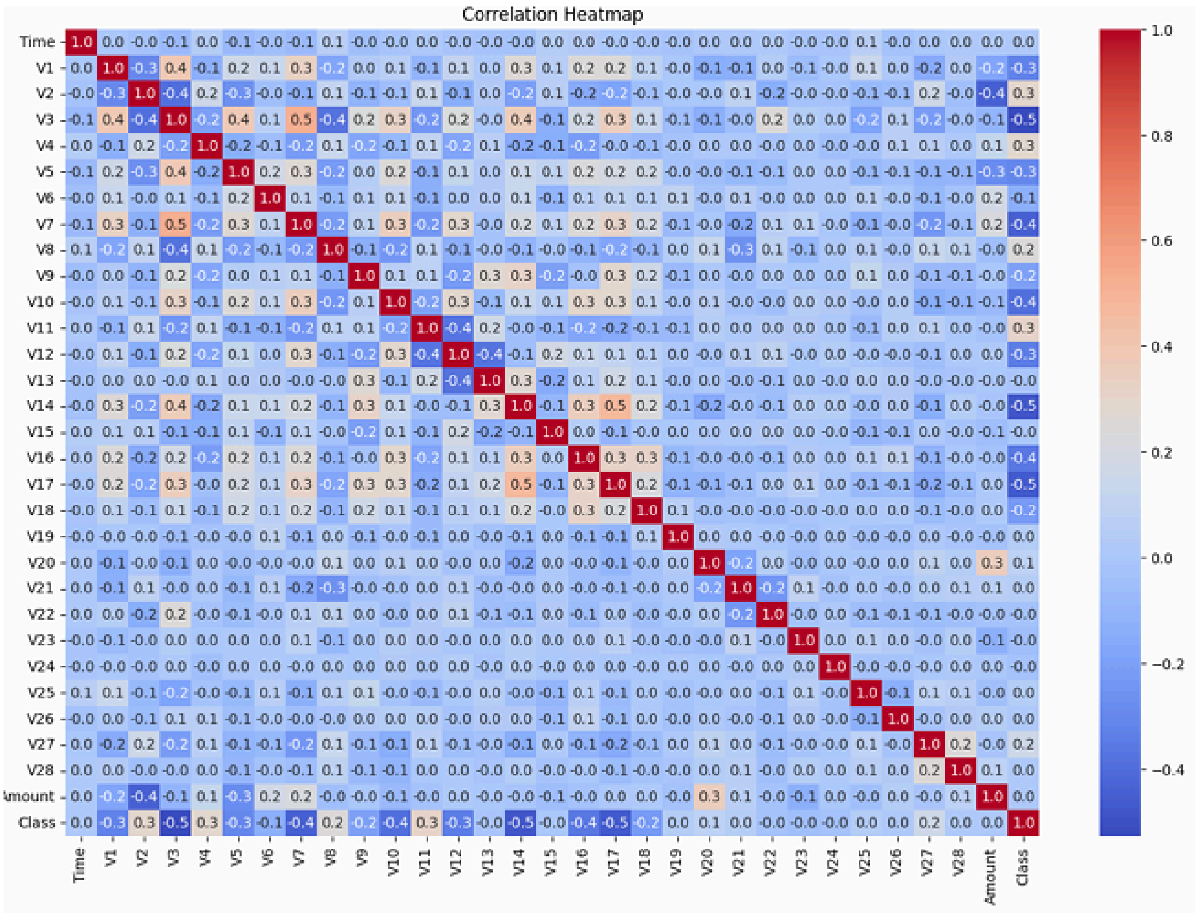
5.2 Visualizations

1. **Fraudulent Transactions Over Time**



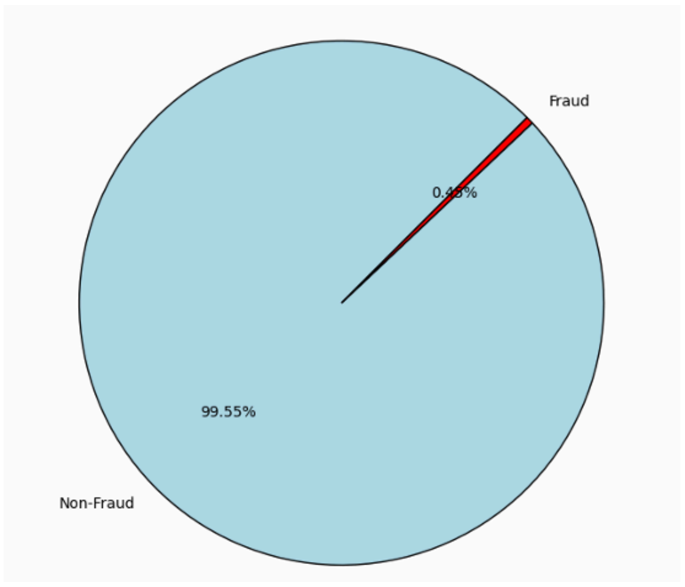
    - Highlighted clusters of fraud activity, providing insights into temporal patterns.

2. **Correlation Heatmap**



- ○ Identified features with strong correlations, aiding feature selection and model interpretability.

3. **Pie Chart for Class Distribution**



- ○ Visualized the imbalance between classes, emphasizing the need for resampling techniques.

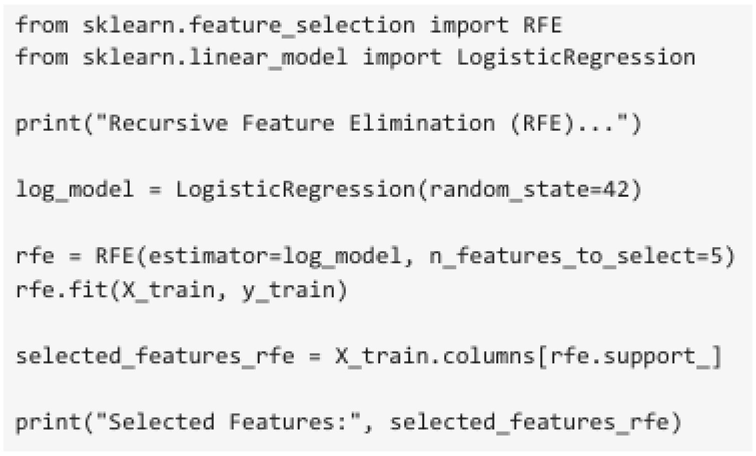
6. Feature Selection

6.1 Importance of Feature Selection

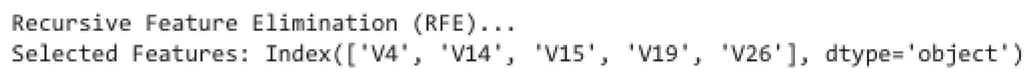
Feature selection is a critical step in machine learning as it helps reduce dimensionality, improve model interpretability, and prevent overfitting. By identifying the most relevant features, the model can focus on the most impactful variables, enhancing predictive performance.

6.2 Methodology

Using Recursive Feature Elimination (RFE):



```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

print("Recursive Feature Elimination (RFE)...")

log_model = LogisticRegression(random_state=42)

rfe = RFE(estimator=log_model, n_features_to_select=5)
rfe.fit(X_train, y_train)

selected_features_rfe = X_train.columns[rfe.support_]

print("Selected Features:", selected_features_rfe)
```



```
Recursive Feature Elimination (RFE)...
Selected Features: Index(['V4', 'V14', 'V15', 'V19', 'V26'], dtype='object')
```

- Selected features: V4, V14, V15, V19, V26.
- These features were identified as having the highest predictive power based on their correlation with the target variable.

7. Model Selection and Evaluation

7.1 Machine Learning Techniques

A variety of machine learning algorithms were tested, each with unique strengths:

- Logistic Regression
- Random Forest
- Decision Trees
- XGBoost
- LightGBM
- CatBoost
- Support Vector Machine

1. **Logistic Regression**: A linear model well-suited for binary classification problems.

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

log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train_resampled, y_train_resampled)

y_pred = log_reg.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred) * 100)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 98.77232142857143

Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      0.99      0.99      3568
         1.0       0.27      1.00      0.42        16

    accuracy                           0.99      3584
   macro avg       0.63      0.99      0.71      3584
weighted avg       1.00      0.99      0.99      3584
```

2. **Random Forest**: An ensemble method combining multiple decision trees to enhance robustness and accuracy.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_resampled, y_train_resampled)
y_pred_rf = rf.predict(X_test)

print("\nRandom Forest")
print("Accuracy:", accuracy_score(y_test, y_pred_rf)*100)
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest
Accuracy: 99.94419642857143

Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      3568
         1.0       0.89      1.00      0.94        16

    accuracy                           1.00      3584
   macro avg       0.94      1.00      0.97      3584
weighted avg       1.00      1.00      1.00      3584
```

3. Decision Trees: A simple yet powerful method for classification, providing interpretable decision rules.

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_resampled, y_train_resampled)
y_pred_dt = dt.predict(X_test)

print("Decision Tree")
print("Accuracy:", accuracy_score(y_test, y_pred_dt)*100)
print("\nClassification Report:\n", classification_report(y_test, y_pred_dt))
```

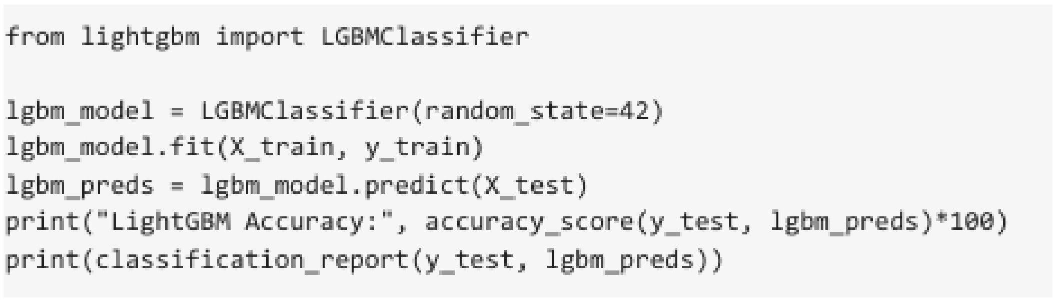
```
Decision Tree
Accuracy: 99.83258928571429

Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      3568
         1.0       0.78      0.88      0.82        16

    accuracy                           1.00      3584
   macro avg       0.89      0.94      0.91      3584
weighted avg       1.00      1.00      1.00      3584
```

4. XGBoost: A gradient boosting algorithm known for its efficiency and superior performance on tabular data.

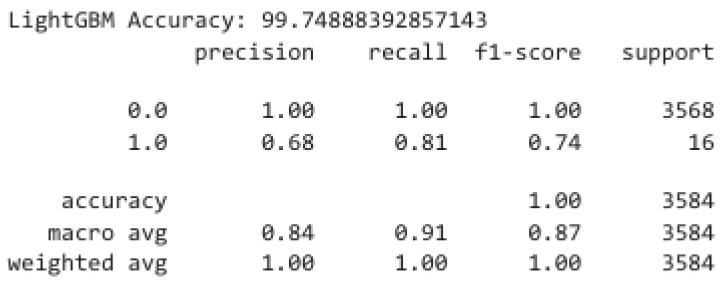
```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(X_train, y_train)
xgb_preds = xgb_model.predict(X_test)
print("XGBoost Accuracy:", accuracy_score(y_test, xgb_preds)*100)
print(classification_report(y_test, xgb_preds))
```

```
XGBoost Accuracy: 99.88839285714286
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      3568
         1.0       0.83      0.94      0.88        16

    accuracy                           1.00      3584
   macro avg       0.92      0.97      0.94      3584
weighted avg       1.00      1.00      1.00      3584
```

5. LightGBM: A gradient boosting framework optimized for speed and scalability

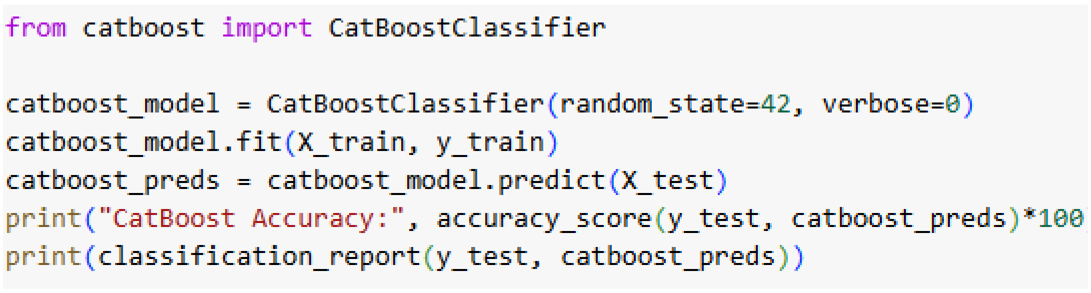


```
from lightgbm import LGBMClassifier

lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_train, y_train)
lgbm_preds = lgbm_model.predict(X_test)
print("LightGBM Accuracy:", accuracy_score(y_test, lgbm_preds)*100)
print(classification_report(y_test, lgbm_preds))
```

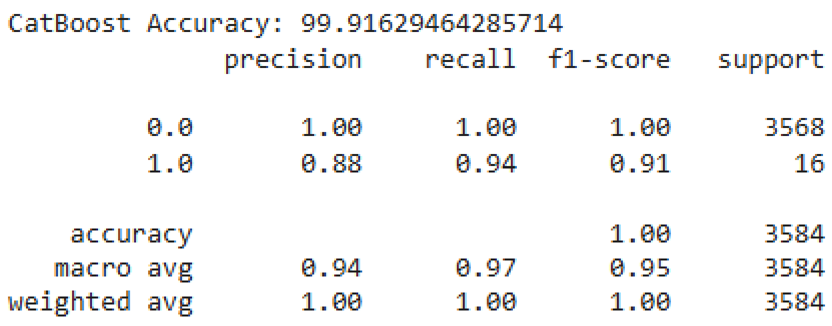


```
LightGBM Accuracy: 99.74888392857143
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      3568
         1.0       0.68      0.81      0.74        16

    accuracy                           1.00      3584
   macro avg       0.84      0.91      0.87      3584
weighted avg       1.00      1.00      1.00      3584
```

6. CatBoost: A boosting algorithm specifically designed to handle categorical features effectively.



```
from catboost import CatBoostClassifier

catboost_model = CatBoostClassifier(random_state=42, verbose=0)
catboost_model.fit(X_train, y_train)
catboost_preds = catboost_model.predict(X_test)
print("CatBoost Accuracy:", accuracy_score(y_test, catboost_preds)*100
print(classification_report(y_test, catboost_preds))
```



```
CatBoost Accuracy: 99.91629464285714
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      3568
         1.0       0.88      0.94      0.91        16

    accuracy                           1.00      3584
   macro avg       0.94      0.97      0.95      3584
weighted avg       1.00      1.00      1.00      3584
```

7. SVM: A supervised machine learning algorithm known as Support Vector Machine, designed to classify data by finding the hyperplane that best separates different classes in a high-dimensional space.

```
from sklearn.svm import SVC

svm_model = SVC(random_state=42,probability=True)
svm_model.fit(X_train, y_train)
svm_preds = svm_model.predict(X_test)
print("SVM Accuracy:", accuracy_score(y_test, svm_preds)*100)
print(classification_report(y_test, svm_preds))
```

```
SVM Accuracy: 99.91629464285714
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      3568
         1.0       0.93      0.88      0.90        16

    accuracy                           1.00      3584
   macro avg       0.97      0.94      0.95      3584
weighted avg       1.00      1.00      1.00      3584
```

7.3 Results Summary

The table below summarizes the performance metrics for each model:

Model	**Accuracy (%)**	**Precision (Fraud)**	**Recall (Fraud)**	**F1-Score (Fraud)**
Logistic Regression	98.77	27%	100%	42%
Random Forest	99.94	89%	100%	94%
Decision Trees	99.83	78%	88%	82%
XGBoost	99.89	83%	94%	88%
LightGBM	99.75	68%	81%	74%
CatBoost	99.91	88%	94%	91%
SVM	99.91	93%	88%	90%

8. Results and Discussion

The **Random Forest** model demonstrated the best overall performance, achieving near-perfect accuracy and recall for fraudulent transactions. While Logistic Regression performed well, its lower precision highlights its limitations in classifying fraudulent transactions. Ensemble methods like Random Forest and XGBoost consistently outperformed simpler models due to their ability to capture complex relationships in the data.

9. Conclusion

Machine learning models, particularly ensemble methods like Random Forest, effectively detect credit card fraud. The project highlights the importance of addressing class imbalance and selecting relevant features for optimal performance. Future work could involve exploring deep learning architectures, incorporating real-time detection capabilities, and testing on larger, more diverse datasets.

10. References

- Kaggle Dataset: Credit Card Fraud Detection.
- Python Libraries: Pandas, Seaborn, Matplotlib, Scikit-learn, SMOTE.
- Machine Learning Algorithms: Logistic Regression, Random Forest, XGBoost, LightGBM, CatBoost.

BY

Prem Raj. P