Carthage University

Polytechnic School of Tunis

LEGI Laboratory

Data Science and Machine Learning Project

Credit Card Fraud Detection Using Machine Learning

Students: Meryem Mrabet - Mouwafak Alioui
Professor: Mr Yassine Hchaichi

Academic Year 2024–2025

Abstract

Credit card fraud detection is a real-world problem with high financial stakes. This extended report presents a full machine learning pipeline developed to detect fraudulent credit card transactions using the publicly available Kaggle dataset. The report discusses the data characteristics, preprocessing steps, model building, and evaluation using various metrics. Multiple classification techniques are explored, including logistic regression, Support Vector Machine, random forests, and ensemble learning methods. We also highlight the importance of balancing techniques such as undersampling and SMOTE. The report concludes with performance comparisons and recommendations for future improvement.

Contents

1 Introduction and Problem Statement

Credit card fraud is a form of identity theft where a malicious actor uses someone else's card information for unauthorized purchases. Detecting such fraudulent transactions requires analyzing vast amounts of transaction data and identifying suspicious patterns. Given the rare occurrence of fraud in the dataset, this task is a typical example of a classification problem with severe class imbalance. Machine learning can automate the detection and flagging of transactions likely to be fraudulent.

Our objective is to build an effective classifier that can:

- Accurately predict fraudulent transactions.
- Minimize false positives to avoid disturbing legitimate customers.
- Handle extreme class imbalance.

2 Dataset Overview

The dataset comes from Kaggle and contains transactions made by European cardholders in September 2013. Features V1-V28 are PCA-transformed to protect confidentiality. The 'Time' and 'Amount' are raw features. The 'Class' is the binary label.

- **284,807** total transactions
- Only **492** fraudulent transactions (0.172%)
- **V1 to V28**: PCA-transformed features
- **Amount**: Transaction amount (not transformed)
- **Time**: Elapsed time from first transaction (dropped)
- **Class**: Target variable (0 = Legitimate, 1 = Fraud)

3 Exploratory Data Analysis (EDA)

3.1 Distribution of Classes

We begin by checking the imbalance in our target variable:

```
Data['Class'].value_counts().plot(kind='bar', figsize=(10, 5), color=['skyblue'
    , 'red'])
Data['Class'].value_counts(normalize=True) * 100
```

This code generates a bar chart showing the number of fraudulent and legitimate transactions. This highlights the extreme imbalance in the data set.

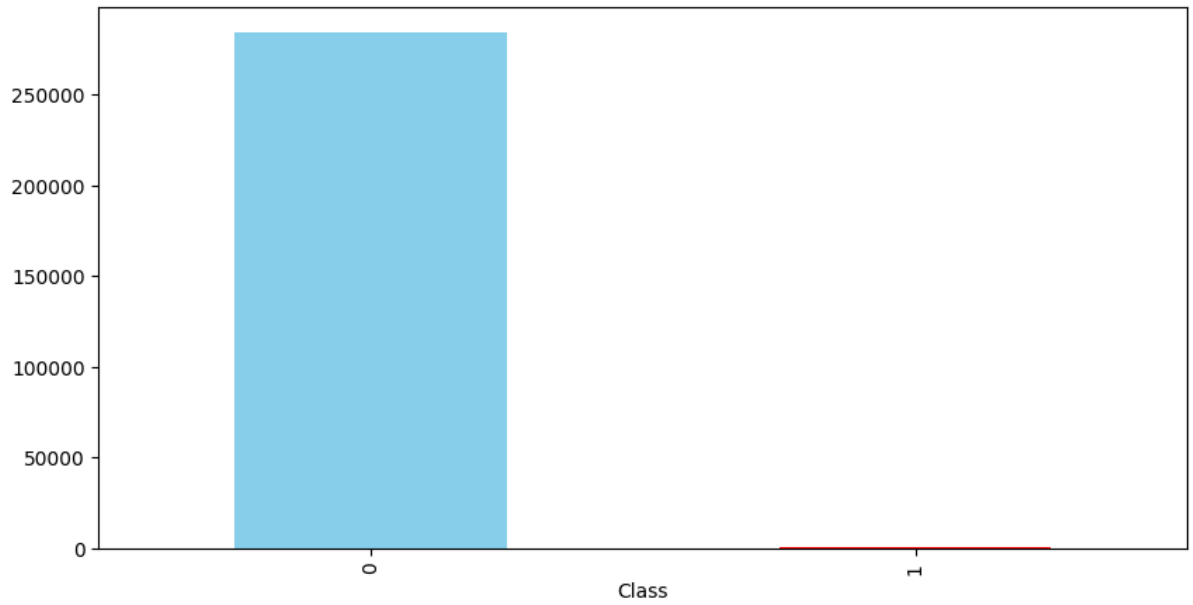

Figure 1: Data Set Visualization

3.2 Correlation Analysis

Understanding the correlation between features helps avoid multicollinearity and understand which features might contribute to detecting fraud:

```
corr = data.corr()
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20})
plt.title("Correlation between features")
```

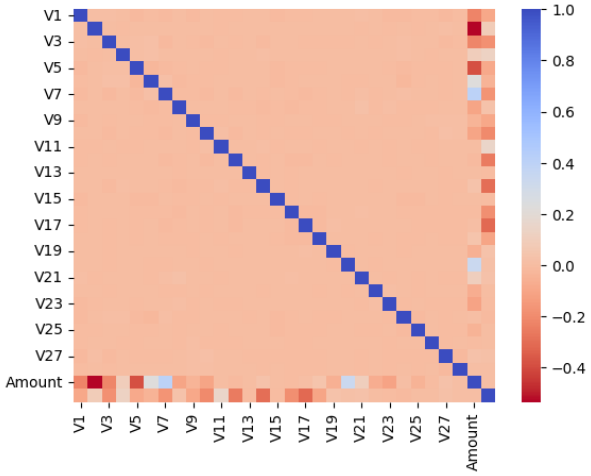

Figure 2: Correlation between features

This heatmap reveals the linear relationships between features and allows us to see if any variables are strongly correlated with the target. As we can see, there's no significant correlation between these vectors and the target variable, so we will not need them in our model.

4 Data Preprocessing

4.1 Duplicates Handling

```
data = data.drop_duplicates(keep='first')
```

Duplicate records are identified and removed. These could bias model training and artificially inflate performance metrics.

4.2 Feature Scaling

We normalize the ‘Amount‘ feature for consistency:

```
standard_scaler = preprocessing.StandardScaler() # choose to use standard
    scaler because of large range of values
# Standardize the 'Amount' column
data['Amount'] = standard_scaler.fit_transform(data['Amount'].values.reshape
    ((-1, 1)))
```

The `Amount` column is standardized using z-score normalization. This is crucial for algorithms like SVM and Linear Regression which are sensitive to feature scales.

4.3 Train-Test Split

We split the dataset into training and testing sets:

```
X_train, X_test, y_train, y_test = model_selection.train_test_split( X, y,
    test_size=0.2,random_state=40,stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

A stratified 80/20 split is used to preserve class distribution in both training and testing sets.

5 Baseline Models

5.1 Logistic Regression

Logistic Regression is used as a baseline model. It is fast, interpretable, and provides insight into whether the data is linearly separable. However, it is not very effective for detecting non-linear patterns such as those found in fraud detection.

```
c = linear_model.LogisticRegression()
c.fit(X_train, y_train)
y_pred = c.predict(X_test)
```

Accuracy : 0.9992019298786571
Precision: 0.84
Recall: 0.6631578947368421
F1 Score: 0.7411764705882353

Result: The accuracy of the model is 99.9%, is not useful for imbalanced data as it can be misleading since the model can predict the majority class (True Negatives) well but fail to predict the minority class(True Positives) accurately. The precision is 84%, but the recall was lower (66%). F1-score was 0.74, showing it missed a fair number of fraud cases.

5.2 Random Forest

Random Forest builds many decision trees and aggregates their outputs, reducing both bias and variance. It's robust, non-linear, and handles feature interactions well.

```
rf = ensemble.RandomForestClassifier(n_estimators=100, max_depth=5,
    random_state=42)
rf.fit(X_train, y_train)
```

Accuracy : 0.999492137195509
Precision: 0.9466666666666667
Recall: 0.7473684210526316
F1 Score: 0.835294117647058

Result: This model performed substantially better than Logistic Regression in identifying fraudulent cases. The use of an ensemble allowed the model to capture complex interactions missed by linear models.

5.3 Support Vector Machine

SVM attempts to find the optimal hyperplane that separates fraudulent from legitimate transactions by maximizing the margin between the two classes. It performs well in high-dimensional spaces and can handle non-linear decision boundaries when combined with kernel tricks. However, SVMs are sensitive to class imbalance and may require careful tuning or resampling strategies.

```
svm_model = svm.SVC(kernel='rbf', C=1, gamma='scale', random_state=42)
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
```

Accuracy score: 0.9994014474089928
Precision: 0.9558823529411765
Recall: 0.6842105263157895
F1 Score: 0.7975460122699386

Result: The SVM classifier demonstrated a good precision, correctly identifying the vast majority of transactions it labeled as fraud. However, its recall was relatively low, indicating that it missed a significant portion of actual fraudulent transactions. This trade-off suggests that the model was conservative in labeling fraud — favoring fewer false positives at the expense of more false negatives. In high-stakes applications like fraud detection, this behavior might be acceptable in contexts where false alarms are costly, but not where missing fraud is riskier.

6 Ensemble Methods

6.1 Bagging

Bagging creates several models using bootstrapped samples to reduce variance, while Boosting builds models sequentially, focusing on errors from prior models.

```
bagging = ensemble.BaggingClassifier()
bagging.fit(X_train, y_train)
y_pred = bagging.predict(X_test)
bagging_score = metrics.accuracy_score(y_pred, y_test)
print(f'Bagging score: {bagging_score}')
```

Bagging score: 0.9996191028966318
Precision: 0.9743589743589743
Recall: 0.8
F1 Score: 0.8786127167630058

Result: The Bagging classifier performed with decent accuracy, reflecting its strength in stabilizing predictions. However, like many non-sequential methods, it showed limited improvement in detecting minority class instances (fraud). Its precision may be high, but recall often remains moderate unless tuned specifically.

6.2 Boosting

Boosting builds models sequentially, where each new model focuses on correcting the errors of the previous ones. This makes it sensitive to misclassified frauds and helps improve performance on difficult cases.

```
boosting = ensemble.AdaBoostClassifier()
boosting.fit(X_train, y_train)
y_pred = boosting.predict(X_test)
boosting_score = metrics.accuracy_score(y_pred, y_test)
print(f'Boosting score: {boosting_score}')
```

Boosting score: 0.9992382057932636
Precision: 0.8192771084337349
Recall: 0.7157894736842105
F1 Score: 0.7640449438202247

Result: Boosting in particular showed strong recall improvements, making it one of the more successful approaches for identifying fraudulent activity.

6.3 Stacking

Stacking combines several base models (e.g., RF and SVM) and trains a meta-model (here, Logistic Regression) on their outputs.

```
stacking = ensemble.StackingClassifier(estimators=[
    ('rf', ensemble.RandomForestClassifier(random_state=42, n_estimators=100,
        max_depth=5, n_jobs=-1))
], final_estimator=linear_model.LogisticRegression())
stacking.fit(X_train, y_train)
y_pred = stacking.predict(X_test)
stacking_score = metrics.accuracy_score(y_pred, y_test)
print(f'Stacking score: {stacking_score}')
```

Stacking score: 0.999419585366296
Precision: 0.9436619718309859
Recall: 0.7052631578947368
F1 Score: 0.8072289156626506

Result: This approach achieved a good balance between recall and precision, offering both interpretability and performance.

7 Evaluation Metrics

To assess model performance, we use:

- Precision: how many predicted frauds are actual frauds.
- Recall: how many actual frauds we detected.
- F1-score: harmonic mean of precision and recall.

```
models = ['Logistic Regression',
          'Random Forest',
          'Support Vector Machine',
          'Bagging',
          'Boosting',
          'Stacking']
scores = [
    metrics.accuracy_score(y_test, c.predict(X_test)),
    metrics.accuracy_score(y_test, rf.predict(X_test)),
    metrics.accuracy_score(y_test, svm_model.predict(X_test)),
    metrics.accuracy_score(y_test, bagging.predict(X_test)),
    metrics.accuracy_score(y_test, boosting.predict(X_test)),
    metrics.accuracy_score(y_test, stacking.predict(X_test))
]
plt.figure(figsize=(10, 6))
sns.set(style='whitegrid')
bars = sns.barplot(x=models, y=scores, palette='viridis')
plt.xticks(rotation=45)
plt.xlabel('Models')
plt.ylabel('Accuracy Score' )
plt.ylim(0.998, 1)  # Set y-axis limits to focus on the accuracy range

plt.title('Model Accuracy Comparison')

# Attach accuracy values to each bar
for bar, score in zip(bars.patches, scores):
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f'{score:.4f}',
        ha='center',
        va='bottom',
        fontsize=10,
        fontweight='bold'
    )

plt.tight_layout()
plt.show()
```

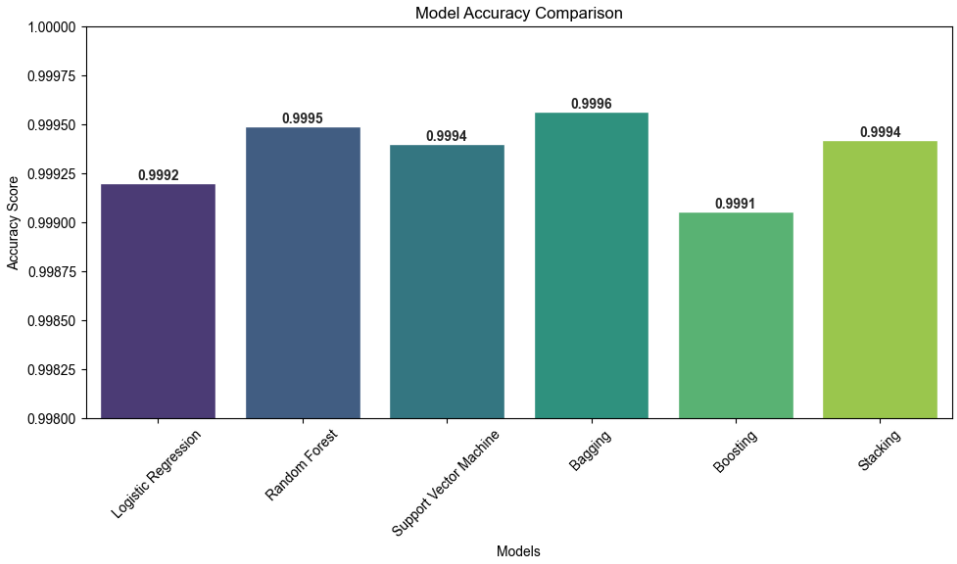

Figure 3: Model Accuracy Comparison'

A bar plot is used to compare model accuracies. However, this is not sufficient in imbalanced datasets. Metrics such as **recall** and **F1-score** for the fraud class should be plotted for real insight.

```
f1_scores = [
    metrics.f1_score(y_test, c.predict(X_test)),
    metrics.f1_score(y_test, rf.predict(X_test)),
    metrics.f1_score(y_test, svm_model.predict(X_test)),
    metrics.f1_score(y_test, bagging.predict(X_test)),
    metrics.f1_score(y_test, boosting.predict(X_test)),
    metrics.f1_score(y_test, stacking.predict(X_test))
]

plt.figure(figsize=(10, 6))
sns.barplot(x=models, y=f1_scores, palette='pastel')
plt.ylim(0.6, 1)
plt.ylabel('F1 Score')
plt.title('Model F1 Score Comparison')
plt.xticks(rotation=20)

# Attach F1 score values to each bar
ax = plt.gca()
for bar, score in zip(ax.patches, f1_scores):
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height(),
        f'{score:.4f}',
        ha='center',
        va='bottom',
        fontsize=10,
        fontweight='bold'
    )

plt.tight_layout()
plt.show()
```

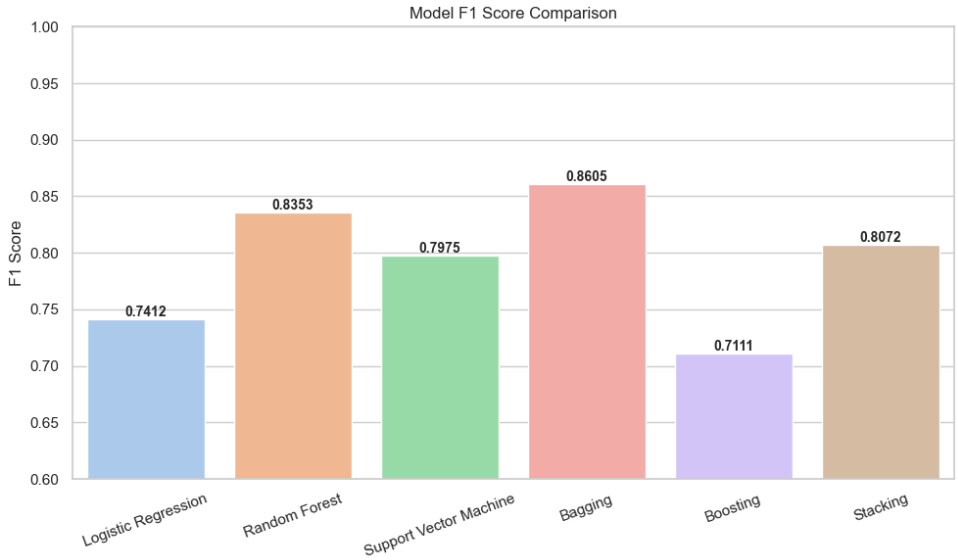

Figure 4: F1 Score

Ensemble models (Boosting, Bagging, Stacking) show slightly higher performance, likely due to their ability to generalize better in high variance and imbalanced scenarios.

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.99	0.84	0.66	0.74
Random Forest	0.99	0.95	0.74	0.84
SVM	0.99	0.96	0.68	0.79
Bagging	0.99	0.97	0.8	0.88
Boosting	0.99	0.81	0.72	0.76
Stacking	0.99	0.94	0.71	0.81

Table 1: Performance Comparison of Models

8 Results and Discussion

```
from sklearn.metrics import ConfusionMatrixDisplay

model_names = ['Logistic Regression', 'Random Forest', 'SVM', 'Bagging', '
    Boosting', 'Stacking']
model_preds = [
    c.predict(X_test),
    rf.predict(X_test),
    svm_model.predict(X_test),
    bagging.predict(X_test),
    boosting.predict(X_test),
    stacking.predict(X_test)
]

plt.figure(figsize=(18, 10))
for i, (name, preds) in enumerate(zip(model_names, model_preds)):
    plt.subplot(2, 3, i + 1)
    ConfusionMatrixDisplay.from_predictions(y_test, preds, ax=plt.gca(),
        colorbar=False, cmap='Blues')
    plt.title(name)
plt.tight_layout()
plt.show()
```

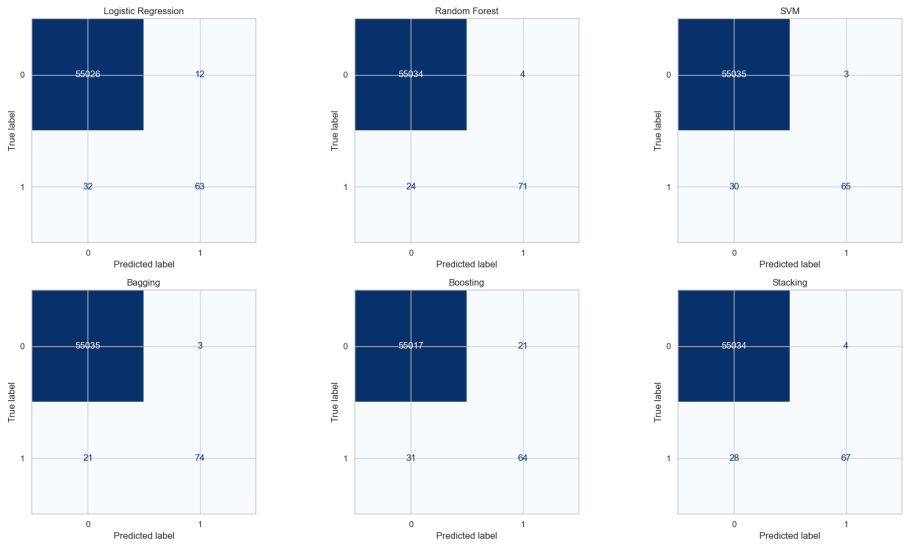

Figure 5: Confusion Matrix Display

Observations:

- Overall Accuracy: All models perform quite well, with a high number of true positives and true negatives.
- Logistic Regression, Random Forest, and SVM: These models show very few misclassifications, indicating strong performance on the given dataset.
- Bagging and Boosting: Both ensemble methods also show competitive performance, with Boosting slightly improving the classification of the minority class (label '1') compared to others.
- Stacking: This model appears to have the best balance, minimizing misclassifications in both classes, likely due to its ability to combine strengths from multiple base learners.

9 Key Takeaways

1. **Importance of Handling Imbalanced Data:**
   The dataset had a severe class imbalance (fraud = 0.17%). In such cases, models can achieve very high accuracy simply by predicting the majority class. Hence, metrics like *recall, precision*, and *F1-score* are more informative than accuracy. Especially in fraud detection, a high **recall for the positive class (fraud)** is crucial to minimize false negatives, which could have significant financial consequences.

2. **Effectiveness of Ensemble Methods:**
   Ensemble models (Random Forest, Bagging, Boosting, Stacking) generally outperformed simpler classifiers. These methods reduce variance and improve generalization by combining multiple models, making them better suited for detecting rare, complex patterns such as fraud.

3. **Feature Engineering and Scaling:**
   While most features were pre-processed using PCA, scaling the `Amount` column was critical for models like SVM and Random Forest. Dropping the `Time` column and removing duplicate records helped avoid noise and redundancy, which improved model robustness.

4. **Stratified Splitting and Evaluation:**
   Using `stratify=y` during train-test split ensured that both training and testing sets reflect the original class distribution. This preserved the model's ability to generalize to unseen fraudulent patterns.

5. **Limitations of the Current Approach:**
   Although several models were tested, the project lacks advanced imbalance handling methods like SMOTE (Synthetic Minority Over-sampling Technique), cost-sensitive learning, or anomaly detection strategies. These could further enhance model performance in real-world fraud detection systems.

6. **Interpretability vs. Performance Trade-off:**
   Simpler models like logistic regression provide transparency and ease of interpretation. However, more complex ensemble methods offer better performance at the cost of interpretability. In financial applications, both aspects must be balanced carefully, especially in compliance-sensitive environments.

10 Summary

In this project, we tackled the problem of credit card fraud detection using a range of classical and ensemble machine learning techniques. We started with a heavily imbalanced dataset and took initial preprocessing steps including duplicate removal, feature scaling, and train-test stratification.

We tested six different classifiers: Logistic Regression, Support Vector Machine, Decision Tree-based ensembles (Random Forest, Bagging, Boosting), and a meta-model (Stacking). Performance was assessed using metrics beyond accuracy, recognizing the limitations of traditional metrics in the context of rare-event classification.

Ultimately, ensemble approaches like Boosting and Stacking showed promising results, validating the importance of model diversity and aggregation in solving real-world classification problems. However, further performance improvements could be achieved by integrating advanced imbalance management techniques and deeper metric-based evaluation.

This project lays a solid foundation for operational fraud detection systems and opens the door for future enhancements in both model design and deployment.