Fraud Detection with Machine Learning: A Comparative Study of Resampling and Modeling Strategies

Taher Alabbar

Abstract

This project evaluates multiple machine learning models combined with various resampling techniques to tackle class imbalance in credit card fraud detection. Models were assessed using F1-score, precision, recall, and PR-AUC on validation and test sets. The best model underwent targeted feature engineering to further improve performance, highlighting the effectiveness of resampling and feature enhancements in detecting fraudulent transactions.

Introduction

Credit card fraud detection is a critical application of machine learning due to the extreme imbalance between legitimate and fraudulent transactions.
This project follows a structured and realistic approach to tackle this problem by applying **multiple machine learning models** with various resampling strategies, aiming to identify the most effective solution for this classification challenge.

The project is broken down into the following key phases:

- **Model Evaluation:** Five models are trained—Logistic Regression, Random Forest, K-Nearest Neighbors (KNN), Neural Network, and a Voting Classifier combining RF, KNN, and NN.
- **Resampling Strategies:** Each model is assessed under six imbalance handling techniques—no resampling (baseline), Random Undersampling, NearMiss, custom KMeans Undersampling, Random Oversampling, and SMOTE.
- **Performance Metrics:** Evaluation is based on the **F1-score** on validation, along with precision, recall, and PR-AUC.
- **Model Selection:** The best-performing model and resampling technique combination is chosen.
- **Targeted Feature Engineering:** Further feature engineering is applied exclusively to the selected model.

- **Final Evaluation:** The improved model is retrained and evaluated once on unseen test data, simulating real-world deployment.

This design allows exploration of **how resampling methods and model architectures interact**, and how feature engineering enhances detection of rare events.

Workflow

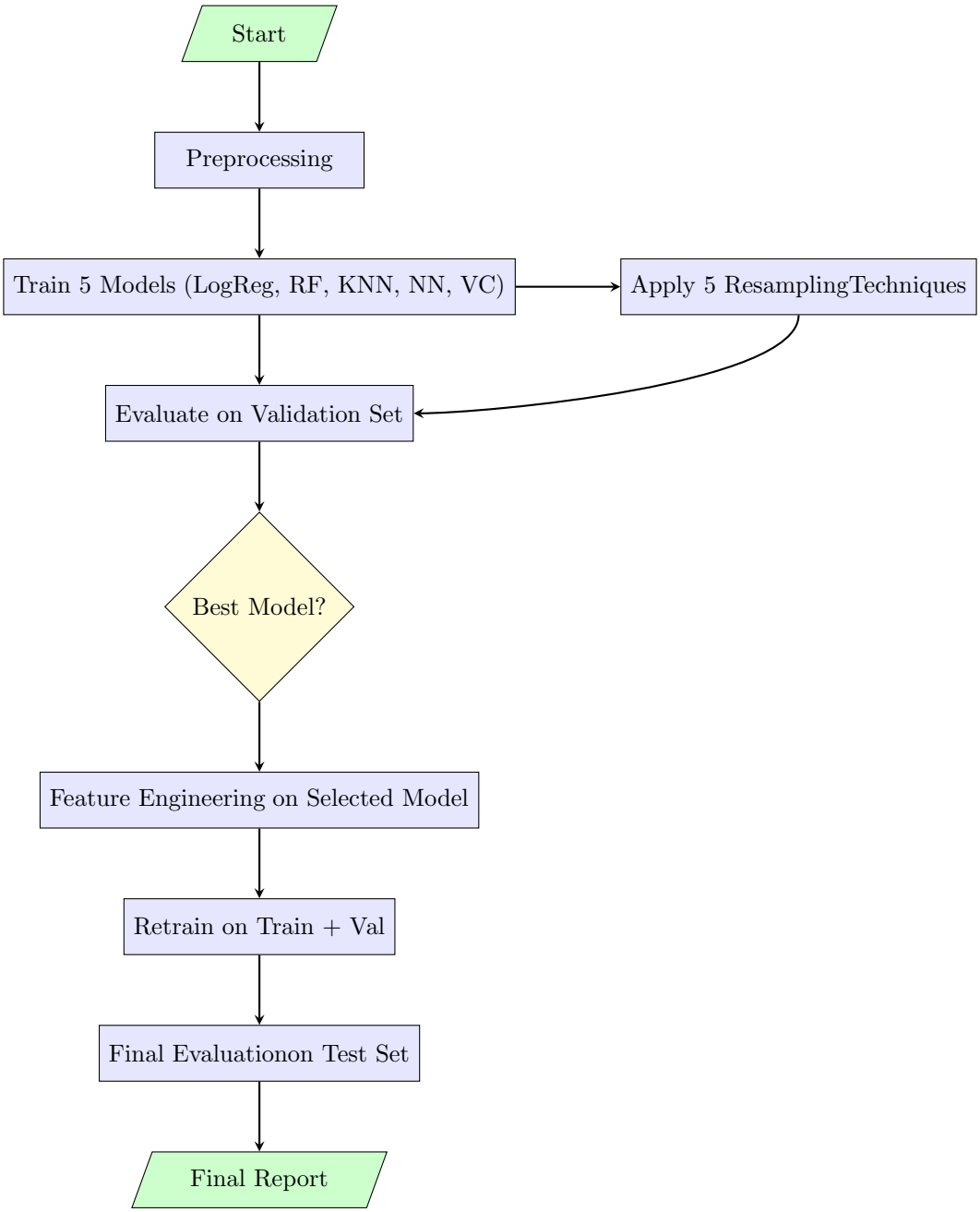

Figure 1: Project Workflow: From Preprocessing to Final Evaluation

Resampling Techniques for Imbalanced Data

The credit card fraud dataset used in this project is heavily imbalanced, with fraudulent transactions comprising less than 1 % of the total data. Training models directly on such data often leads to poor detection of the minority class. As a baseline, models were initially trained without any balancing strategy. For **Logistic Regression** and **Random Forest**, we also applied a `class_weight` parameter to implement a cost-sensitive approach—penalizing the misclassification of fraud cases to improve recall. To further address the imbalance, we explored and compared five resampling techniques:

- **Random Undersampling:** This technique reduces the size of the majority class by randomly eliminating examples. While it helps in balancing the classes, it risks discarding informative data, potentially weakening the model's understanding of non-fraudulent patterns.
- **NearMiss:** A more sophisticated form of undersampling. NearMiss selects majority class samples that are close to the minority class in feature space. This ensures the model is exposed to "difficult" non-fraud examples that are harder to distinguish from fraud, improving boundary learning. However, it may overly compress the majority class representation.
- **KMeans-Based Undersampling (Custom):** In this custom approach, KMeans clustering is applied to the majority class to identify groups of similar points. Representative samples from each cluster are then selected to preserve diversity and structure in the downsampled data. This method retains key patterns in the majority class while reducing volume, and avoids the randomness of conventional undersampling.
- **Random Oversampling:** Here, the minority class is upsampled by randomly duplicating existing samples. While simple, it may lead to overfitting, especially with models sensitive to repetition. Nonetheless, it ensures full use of available data without losing any majority class information.
- **SMOTE (Synthetic Minority Over-sampling Technique):** Rather than duplicating samples, SMOTE generates synthetic examples by interpolating between existing minority class instances. This introduces variation and helps improve model generalization. However, SMOTE can introduce noise if synthetic points fall in ambiguous or overlapping regions of feature space.

Each technique was applied independently, and its effects were evaluated using consistent model training and validation procedures. Their comparative influence on performance metrics—such as precision, recall, F1-score, and PR-AUC—guided the selection of models and further feature engineering steps.

Data Analysis

The dataset consists of the following features:

- **Time:** Number of seconds elapsed between this transaction and the first transaction in the dataset.
- **V1 – V28:** Anonymized features resulting from a PCA transformation applied to protect user identities and sensitive information.
- **Amount:** Transaction amount.
- **Class:** Binary label indicating transaction status; 1 for fraudulent transactions, 0 for legitimate.

Observations: Only `Time` and `Amount` were interpretable; the remaining features were anonymized via PCA. A `log1p` transformation was applied to `Amount` due to its heavy right skew, caused mainly by a few high-value legitimate transactions. However, empirical testing showed that `Time` contributed little to model performance and was therefore excluded from training. This decision was supported by the following:

- Fraudulent and legitimate transactions exhibited similar temporal distributions.
- The right skew in `Amount` was driven by outliers in legitimate transactions, while most values—including fraudulent ones—lay within the \$0–\$500 range.
- Correlation with the target variable was negligible: 0.01 for `Amount` and –0.01 for `Time`.
- Although `Time` may be noise or redundant for most resampling methods, it is crucial for `NearMiss`'s distance-based sampling. Removing `Time` caused failures and poorer performance with `NearMiss`, showing its reliance on geometric features. Thus, `Time` was retained whenever using `NearMiss`.
- Duplicate Rows: A total of 448 duplicated rows were found in the dataset (440 normal, 8 fraud). Removing them improved F1 performance for Logistic Regression and Neural Network, but reduced performance for K-Nearest Neighbors and Random Forest. After evaluation, we chose to keep the duplicates to maintain overall ensemble performance.

Although correlation is not a sufficient indicator on its own, empirical evaluation of model performance supported the decision to retain `Amount` and drop `Time`.

To support targeted feature engineering, we began with two key analyses. First, a **correlation analysis** was performed on the anonymized features `V1` through `V28` using an undersampled version of the dataset. This helped reveal subtle relationships with the target class that might be obscured by the overwhelming majority of legitimate transactions. Second, a **Random Forest feature importance analysis** was conducted across all available features to identify which ones contribute most to prediction performance. Together, these insights inform our strategy for creating new, informative features that can enhance model learning. (see Feature Engineering section)

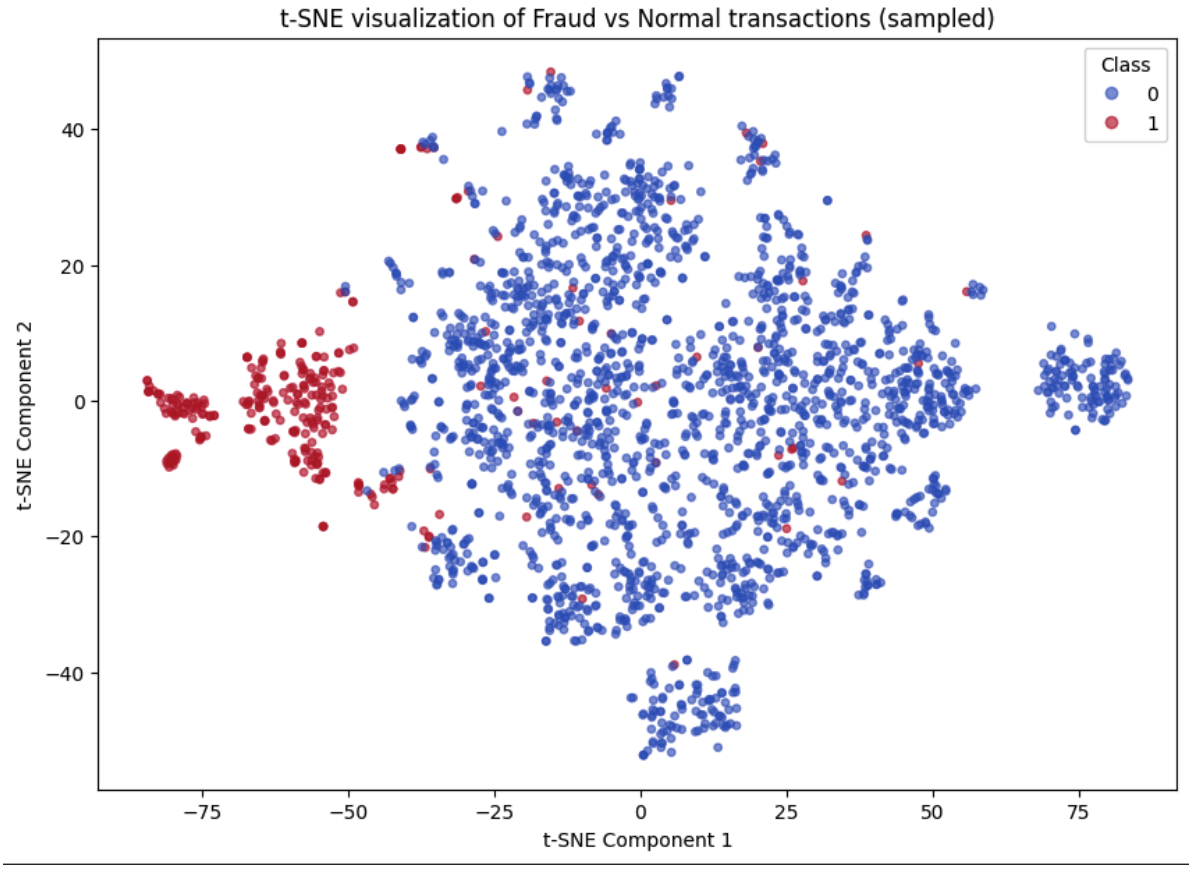

Figure 1: t-SNE visualization of Fraud vs. Normal transactions (with random undersampling of the majority class).

To better understand class separation, we applied **t-SNE** (t-distributed Stochastic Neighbor Embedding) on a subset of the data created by randomly undersampling the majority (normal) class. Directly applying t-SNE on the full dataset would obscure class structure due to extreme imbalance.

The visualization reveals two distinct clusters, with fraudulent transactions partially overlapping the larger cluster of legitimate ones. This indicates that some fraud cases exhibit patterns similar to normal behavior, making them harder to detect. As a result, oversampling methods like **SMOTE** may inadvertently generate synthetic fraud samples in these overlapping regions, introducing noise and reducing model reliability.

Important Notes Before Modeling

- **Avoid Pre-Resampling Due to Data Leakage:** Applying resampling methods *before* cross-validation may seem convenient, but it introduces a serious risk of data leakage. For example, methods like KMeans undersampling select representative majority samples based on clustering. If resampling is done before splitting, information from the minority class may influence the selection process, leaking into both training and validation sets. This contamination can artificially boost performance metrics and lead to overly optimistic, unreliable model evaluation.

- **Note on Cross-Validation:** It is essential to use `StratifiedKFold` with `GridSearchCV` for imbalanced data. Random splitting may distort class distributions in folds, leading to misleading results. While `RepeatedStratifiedKFold` reduces variance via multiple rounds, I found `StratifiedKFold` sufficient and computationally more efficient.

- **Order of Preprocessing (Resample → Scale → Train):** This sequence is essential for building robust and realistic models. Resampling is performed *before* scaling to ensure that synthetic or selected samples (e.g., from SMOTE or KMeans) are created based on the original geometry of the data. These methods rely on raw distances and density relationships, which become distorted if scaling is applied beforehand. Once class balance is achieved, scaling is applied to standardize feature magnitudes—a crucial step for distance-based algorithms (e.g., KNN, SVM), and gradient-based models (e.g., logistic regression, neural networks). Reversing this order can misguide the resampling process and produce unrealistic synthetic points or clusters, leading to degraded model generalization.

- **Preprocessing Pipeline (Resample → Scale → PCA → Train):** Maintaining this order is critical. PCA relies on feature variances and correlations, so it must be applied *after* scaling to ensure all features contribute fairly. Applying PCA *before* resampling is also problematic: resampling methods like KMeans or NearMiss depend on distances in the original feature space. If PCA is applied first, it alters that space, distorting the resampling process and harming model performance. This order ensures proper class balancing and realistic dimensionality reduction.

- **Limitations of Grid Search Cross-Validation Performance:** Automated hyperparameter tuning using `GridSearchCV` or `RandomizedSearchCV` selects parameters based on cross-validation performance. However, these parameters may not generalize well to truly unseen data, especially when the dataset is small, imbalanced, or noisy. Cross-validation splits can introduce variance, leading to overfitting to specific folds. Always evaluate the final model on a separate hold-out validation set. *In some cases, manual tuning using a dedicated validation set allows for more interpretable, stable, and realistic adjustments that reflect actual deployment conditions.*

- **Interaction Between Resampling, Scaling, and Model Type:** The effectiveness of resampling methods often depends on the chosen scaling strategy. For example, distance-based undersampling techniques like `KMeans` or `NearMiss` tend to perform better with `MinMaxScaler`, as it preserves relative distances in a normalized

range. In contrast, some models (e.g., `Logistic Regression`) may benefit more from `StandardScaler` due to assumptions about feature distributions. There is no universally optimal combination—experimenting with different resampling and scaling pairs is essential, as their interactions can significantly impact model performance.

- **Reproducibility Through Fixed Random Seed:** To ensure reproducibility and consistent results across experiments, I used a fixed random seed (`random_state=42`) throughout all resampling techniques, model training procedures, and cross-validation steps. This makes results comparable and minimizes randomness in evaluations. make it smaller

- **Threshold Optimization:** Instead of using the default 0.5 cutoff, I optimized the classification threshold by searching for the value that maximizes F1 score on validation data. This improves the balance between precision and recall in imbalanced classification.

Logistic Regression

RobustScaler, which is more resilient to outliers than StandardScaler, improved precision in Logistic Regression—key for reducing false positives in fraud detection. MinMaxScaler was applied with distance-based undersamplers like KMeans and NearMiss to preserve feature ranges critical for these methods. Parameters for resampling and scaling were optimized via GridSearchCV. Thresholds reported correspond to optimal values per model.

1. Without Resampling (Cost-Sensitive Learning)

Parameters:

$$C = 10, \quad \text{class\_weight} = \{0 : 1, 1 : 4\}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	81.14%	88.72%	74.75%	78.69%
Validation	82.14%	88.46%	76.67%	74.32%

Table 1: Performance metrics at threshold 0.737

The model exhibits high precision, reflecting a conservative approach that minimizes false alarms.

2. Random Undersampling

Parameters:

$$\text{sampling\_strategy} = 0.5, \quad C = 0.1, \quad \text{class\_weight} = \text{None}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	70.18%	78.78%	63.28%	69.61%
Validation	70.52%	73.49%	67.78%	65.84%

Table 2: Performance metrics at threshold 0.980

Performance declined markedly compared to cost-sensitive learning, indicating that random undersampling discards important majority class information. Class weights were unnecessary due to balanced classes post-sampling.

3. NearMiss Undersampling

Parameters:

$$n\_neighbors = 3, \quad version = 1, \quad C = 0.1, \quad \text{class\_weight} = \text{'balanced'}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	66.46%	62.61%	70.82%	70.59%
Validation	69.74%	64.76%	75.56%	66.99%

Table 3: Performance metrics at threshold 0.899

NearMiss targets borderline majority samples near the minority class, improving recall at the cost of precision, increasing sensitivity to fraud but also false alarms.

4. Undersampling with KMeans (Custom Undersampler)

Parameters:

$$n\_clusters = 300, \quad C = 0.1, \quad \text{class\_weight} = \text{None}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	78.70%	78.06%	79.34%	76.08%
Validation	82.02%	82.95%	81.11%	72.98%

Table 4: Performance metrics at threshold 0.495

KMeans undersampling with MinMax scaling closely matches baseline F1. It favors recall with slightly reduced precision, making it suitable when detecting more fraud cases outweighs false alarms. This method selects representative majority samples, improving learning compared to random undersampling and reducing overfitting risks seen in NearMiss.

5. Random Oversampling

Parameters:

$$C = 0.1, \quad \text{class\_weight} = \text{None}, \quad \text{sampling\_strategy} = 0.5$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	75.32%	73.67%	77.05%	74.26%
Validation	78.69%	77.42%	80.00%	72.72%

Table 5: Performance metrics at threshold 0.949

Random oversampling improves recall by preserving majority data but introduces duplicate minority samples, potentially adding noise. It achieves a recall-biased F1, beneficial for fraud detection.

6. SMOTE Oversampling

Parameters:

$$k\_neighbors = 3, \quad \text{sampling\_strategy} = 0.5, \quad C = 0.1, \quad \text{class\_weight} = \text{None}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	75.53%	75.66%	75.41%	73.30%
Validation	78.86%	81.18%	76.67%	71.76%

Table 6: Performance metrics at threshold 0.960

SMOTE offers better precision than random oversampling but at a cost of reduced recall, possibly due to less diverse synthetic minority samples.

Summary:

- Baseline logistic regression with class weights yielded the best overall performance, with balanced recall and strong precision.
- KMeans undersampling closely followed, providing a good trade-off between recall and precision through more representative majority sampling.
- Random undersampling led to performance degradation by losing key majority data.
- NearMiss increased recall by focusing on borderline samples but at the expense of precision.
- Random oversampling improved recall but added noise via duplicated minority samples.
- SMOTE improved precision compared to random oversampling but reduced recall due to synthetic sample characteristics.

Neural Network

Three scalers were tested with the neural network: StandardScaler, RobustScaler, and MinMaxScaler. StandardScaler performed best, followed by RobustScaler, while MinMaxScaler gave the worst results due to its sensitivity to outliers. Despite lacking outlier resistance, StandardScaler preserved useful feature distributions that supported stable training. Although both neural networks and logistic regression use gradient-based optimization, their sensitivity to scaling differs. Neural networks benefit from StandardScaler's zero-mean, unit-variance scaling for smoother convergence, while logistic regression performs better with RobustScaler due to its handling of extreme values.

Note: This entire section is manually tuned because the grid search results were unrepresentative and performed poorly on the validation set. Therefore, grid tuning alone was insufficient.

1. Without Resampling

Parameters:

$$\text{hidden\_layers} = (128,), \quad \text{activation} = \text{ReLU}, \quad \text{learning\_rate} = 0.001, \quad \alpha = 0.001$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	93.22%	99.26%	87.87%	97.54%
Validation	86.42%	97.22%	77.78%	83.51%

Table 7: Performance metrics at threshold 0.606

The neural network demonstrated strong generalization and near-perfect training accuracy. It outperformed logistic regression by achieving both higher precision and recall at a lower threshold, reflecting its ability to model complex patterns while maintaining conservative predictions.

2. Random Undersampling

Parameters: `sampling_strategy = 0.5, activation = ReLU, alpha = 0.001, hidden_layers = (128,), learning_rate = 0.001`

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	47.08%	32.43%	85.90%	60.91%
Validation	44.98%	30.96%	82.22%	57.76%

Table 8: Performance metrics at threshold 0.990

Using random undersampling significantly degraded the neural network's performance, highlighting that NNs benefit from larger, richer datasets to capture complex patterns effectively.

The model became overly sensitive, labeling many cases as fraud—resulting in high recall but very low precision, meaning many false alarms. This indicates the model's decision quality was compromised. Employing more sophisticated undersampling techniques could help balance the dataset better and improve overall performance.

3. NearMiss Undersampling

Parameters: `n_neighbors = 3, version = 1, activation = ReLU, alpha = 0.001, hidden_layers = (256, 128), learning_rate = 0.01`

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	67.11%	67.67%	66.56%	66.95%
Validation	71.19%	72.41%	70.00%	65.08%

Table 9: Performance metrics at threshold 0.990

NearMiss undersampling improved over random undersampling by selecting borderline majority samples, leading to a more balanced precision and recall. However, it still performs significantly below the baseline neural network, reflecting the model's sensitivity to reduced data size. Compared to KMeans, NearMiss focuses on edges of the decision boundary, which can lead to overfitting and less representative training data.

4. KMeans Undersampling

Parameters:

$n\_clusters = 500, \quad \text{activation} = \text{ReLU}, \quad \alpha = 0.001, \quad \text{hidden\_layers} = (128, 64), \quad \text{learning\_rate} = 0.01$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	76.95%	81.62%	72.79%	73.31%
Validation	78.31%	85.53%	72.22%	69.89%

Table 10: Performance metrics at threshold 0.949

KMeans undersampling performed better than random undersampling since it aims to provide a representative selection of the majority class, unlike random undersampling which discards samples arbitrarily. Overall, all undersampling methods lagged behind the baseline neural network, reflecting the model's strength in capturing complex patterns and its need for richer data. Oversampling methods are expected to yield better results.

5. Random Oversampling

Parameters:

$$\text{hidden\_layers} = (128,), \quad \text{activation} = \text{ReLU}, \quad \text{learning\_rate} = 0.001, \quad \alpha = 0.001$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	99.03%	98.07%	100.00%	99.78%
Validation	86.23%	93.51%	80.00%	83.65%

Table 11: Performance metrics at threshold 0.980

RobustScaler is preferred because it's resistant to outliers. When using oversampling methods like random oversampling or SMOTE, outliers can distort synthetic samples if standard scaling is applied. By using the median and IQR, RobustScaler creates more stable features, helping the neural network learn true patterns instead of noise.

6. SMOTE Oversampling

Parameters:

$$\text{hidden\_layers} = (128,), \quad \text{activation} = \text{ReLU}, \quad \text{learning\_rate} = 0.001, \quad \alpha = 0.001$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	99.02%	98.38%	99.67%	99.99%
Validation	85.71%	92.31%	80.00%	82.93%

Table 12: Performance metrics at threshold 0.970

Surprisingly, random oversampling outperformed SMOTE, consistent with our Logistic Regression results. This suggests that simple duplication better preserves the data distribution than synthetic samples. While recall improved—capturing more fraud cases—precision dropped, leading to more false positives.

Summary:

- The best-performing setup was the neural network trained without any resampling technique, demonstrating strong generalization and high performance across all metrics.
- Among the resampling methods, random oversampling and SMOTE achieved nearly equivalent results, both significantly outperforming undersampling techniques. Surprisingly, simple duplication (random oversampling) slightly outperformed SMOTE, likely due to better preservation of the original data distribution.
- KMeans undersampling delivered relatively strong results compared to other undersampling strategies, as it maintained representative majority class samples without discarding information randomly.
- NearMiss and random undersampling produced the weakest results due to loss of information, which limited the neural network's capacity to learn complex patterns.

- Unlike in logistic regression—where undersampling worked better—neural networks benefited more from oversampling. This contrast underscores that resampling strategies must be chosen model-specifically rather than applied uniformly.
- The neural network required extensive manual tuning; although grid search performed well on cross-validation splits, it did not select the best model for the validation set because the optimal parameter combination was present but not chosen due to split-based tuning.
- Lastly, the superior performance without resampling highlights the neural network's inherent capacity to learn from imbalanced data and warns against blindly applying resampling across all models.

Random Forest

Random Forests are ensemble learning models that build a collection of decision trees, each trained on a random subset of data using bootstrapping (bagging). Predictions are aggregated by majority voting in classification tasks, yielding a robust and stable model. Each tree also considers a random subset of features, promoting diversity among trees and reducing overfitting. This allows Random Forests to handle noisy, high-dimensional data and capture non-linear patterns effectively without extensive parameter tuning. Furthermore, Random Forests are scale-invariant, so feature scaling is unnecessary.

Baseline (No Resampling)

Parameters:

$$n\_estimators = 100, \quad \text{max\_depth} = \text{None}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	98.92%	98.52%	99.33%	99.98%
Validation	88.24%	93.75%	83.33%	85.06%

Table 13: Performance metrics at optimal threshold 0.364

The baseline **Random Forest** outperforms the Neural Network, balancing high precision and recall effectively. It captures complex feature interactions and remains robust without scaling or heavy tuning. This model preserves the original data distribution, yielding the best overall fraud detection performance. **Applying** `class_weight='balanced'` was tested but led to worse performance, likely due to disrupting the model's natural handling of class imbalance and the original feature distribution.

Random Undersampling

Parameters:

$$\text{sampling\_strategy} = 0.5, \quad n\_estimators = 100, \quad \text{max\_depth} = \text{None}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	85.86%	90.55%	81.64%	85.27%
Validation	82.14%	88.46%	76.67%	75.68%

Table 14: Performance metrics at threshold 0.939

Performance declines compared to baseline, mainly due to loss of majority class information affecting recall. Precision remains high, indicating the model limits false positives effectively despite reduced overall sensitivity.

NearMiss Undersampling

Parameters:

$$n\_neighbors = 3, \quad \text{version} = 1, \quad n\_estimators = 100, \quad \text{max\_depth} = \text{None}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	69.93%	63.78%	77.38%	52.42%
Validation	69.11%	65.35%	73.33%	50.17%

Table 15: Performance metrics at threshold 1.000

NearMiss aggressively removes majority samples near the minority class, significantly reducing precision and PR-AUC. This indicates over-aggressive undersampling harms model generalization.

KMeans Undersampling

Parameters:

$$n\_clusters = 300, \quad n\_estimators = 100, \quad \text{max\_depth} = \text{None}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	86.40%	90.94%	82.30%	89.81%
Validation	79.25%	91.30%	70.00%	77.95%

Table 16: Performance metrics at threshold 0.939

KMeans undersampling performs better than NearMiss by preserving representative majority samples, achieving higher precision and F1. However, recall decreases compared to random undersampling, suggesting the model favors reducing false alarms over detecting all frauds.

Random Oversampling

Parameters:

$$\text{sampling\_strategy} = 0.5, \quad n\_estimators = 100, \quad \text{max\_depth} = \text{None}$$

Random oversampling outperforms random undersampling by preserving all majority samples and increasing minority class instances, improving recall without sacrificing precision excessively. The Random Forest's robustness to noise helps it handle duplicated minority samples well.

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	98.87%	97.76%	100.00%	100.00%
Validation	86.55%	91.36%	82.22%	85.20%

Table 17: Performance metrics at threshold 0.283

SMOTE Oversampling

Parameters:

$$k\_neighbors = 3, \quad \text{sampling\_strategy} = 0.5, \quad n\_estimators = 100, \quad \text{max\_depth} = \text{None}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	100.00%	100.00%	100.00%	100.00%
Validation	84.34%	92.11%	77.78%	83.91%

Table 18: Performance metrics at threshold 0.545

SMOTE slightly underperforms random oversampling, with lower recall and F1 despite higher precision. Synthetic samples may introduce artificial patterns that impact generalization negatively.

Summary:

- The baseline Random Forest trained on original imbalanced data achieved the best overall performance, preserving natural class distributions.
- Random oversampling closely followed, enhancing minority representation without losing majority information, benefiting recall.
- Random undersampling decreased performance by discarding valuable majority class data, reducing recall.
- NearMiss undersampling was too aggressive, causing notable performance degradation.
- KMeans undersampling balanced precision and recall better than NearMiss but still lagged behind random undersampling.
- SMOTE introduced synthetic samples but performed worse than random oversampling, likely due to less natural minority class representation.
- The performance drop with sophisticated undersampling methods indicates that Random Forest performs best when trained on data distributions close to the original. Aggressive or complex sampling techniques may distort useful patterns and harm generalization.

- Overall, oversampling methods performed better than undersampling, as they avoid losing information and maintain data distribution better aligned with Random Forest's strengths.

K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm that classifies a point based on the majority class among its K nearest neighbors. As a lazy learner, KNN stores all training data and defers computation until prediction. Its reliance on distance metrics like Euclidean distance makes feature scaling (e.g., standardization or normalization) essential for effective performance. KNN can model complex class boundaries but is highly sensitive to noise, irrelevant features, and class imbalance. The parameter K determines how many neighbors are considered: small values can lead to overfitting, while large values risk underfitting. Optimal performance requires tuning both K and the distance metric.

Baseline (No Resampling)

Parameters:

$$k = 7, \quad \text{weights} = \text{"distance"}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Validation	86.59%	95.95%	78.89%	84.38%

Table 19: Validation performance metrics at threshold 0.414

This configuration yielded high F1 and precision, indicating a conservative classifier that prioritizes minimizing false positives. Standard scaling was essential. The perfect training performance hints at overfitting, but strong generalization is retained.

Random Undersampling

Parameters:

$$\text{sampling\_strategy} = 0.5, \quad k = 9, \quad \text{weights} = \text{"uniform"}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	76.45%	85.43%	69.18%	66.44%
Validation	76.54%	86.11%	68.89%	66.59%

Table 20: Performance metrics at threshold 0.889

Undersampling significantly reduced performance due to information loss. Applying PCA improved generalization:

PCA smoothed the feature space, helping mitigate information loss from undersampling.

With PCA	F1 Score	Precision	Recall	PR-AUC
Training	80.41%	83.69%	77.38%	72.80%
Validation	78.57%	84.62%	73.33%	69.20%

Table 21: Performance metrics with PCA at threshold 0.788

NearMiss Undersampling

Parameters:

$$n\_neighbors = 9, \quad \text{weights} = \text{"uniform"}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	74.92%	77.54%	72.46%	63.00%
Validation	74.71%	77.38%	72.22%	61.71%

Table 22: Performance metrics at threshold 0.889

Performance was moderate but lower than random undersampling. Including the `Time` feature and using MinMax scaling were crucial. PCA did not help.

KMeans Undersampling

Before dropping the `Time` feature (very poor results):

Dataset	F1 Score	Precision	Recall	PR-AUC
Validation	25.31%	15.03%	80.00%	12.42%

Table 23: Performance metrics at threshold 0.889

After dropping the `Time` feature (significant improvement):
Parameters:

$$n\_clusters = 500, \quad n\_neighbors = 9, \quad \text{weights} = \text{"uniform"}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	79.35%	78.10%	80.66%	76.95%
Validation	81.14%	83.53%	78.89%	75.42%

Table 24: Performance metrics at threshold 0.788

This was the best KNN performance among all undersampling strategies. Dropping the `Time` feature eliminated noise, highlighting the importance of proper feature selection when using distance-based models and clustering methods.

Random Oversampling

Parameters:

$$\text{PCA (12 components)}, \quad \text{StandardScaler}, \quad k = 5, \quad \text{weights} = \text{"distance"}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Validation	82.28%	95.59%	72.22%	74.67%

Table 25: Performance metrics at threshold 0.889

Oversampling preserved all original data and outperformed all undersampling methods. PCA slightly improved test F1 by smoothing distances and reducing dimensions. Despite the perfect training metrics (possible overfitting), test performance remained robust.

SMOTE Oversampling

Parameters:

$$\text{sampling\_strategy} = 0.7, \quad \text{balance\_k\_neighbors} = 3, \quad k = 3, \quad \text{weights} = \text{"distance"}$$

Dataset	F1 Score	Precision	Recall	PR-AUC
Validation	72.43%	70.53%	74.44%	54.62%

Table 26: Performance metrics at threshold 0.667

SMOTE underperformed compared to other resampling strategies. Synthetic samples likely distorted the distance space, negatively impacting KNN, which relies heavily on distance relationships. PCA worsened results in this case.

Summary:

- KNN achieved its best performance with standard scaling and without resampling, reaching an F1 of 86.2%.
- Random oversampling with PCA yielded a competitive F1 of 82.28%, proving surprisingly effective despite the risk of overfitting.

- Among all undersampling methods, KMeans undersampling (after removing the `Time` feature) yielded the best result: F1 = 81.14%, Recall = 78.89%, outperforming NearMiss in all metrics.
- NearMiss and Random undersampling degraded performance by removing valuable data, although PCA helped slightly in the latter.
- SMOTE performed worse than random oversampling due to the introduction of synthetic samples that may have distorted distance metrics.
- PCA was helpful with random oversampling but hurt performance when used with SMOTE or aggressive undersampling.
- The `Time` feature negatively affected clustering in KMeans and distance metrics in KNN—dropping it significantly improved results, emphasizing the importance of feature selection.
- Overall, KNN models are highly sensitive to data distribution, feature scaling, and dimensionality. Oversampling—especially random—proved most beneficial for preserving structure and improving generalization.

Comparison of Resampling Methods Across Models

Logistic Regression

Resampling Method	Summary
Baseline	Balanced precision and recall; effective class weighting handles imbalance well.
Random Undersampling	Significant drop in recall due to loss of majority data; precision remains high.
NearMiss Undersampling	Improved recall but precision drops; more sensitive to fraud but more false alarms.
KMeans Undersampling	Better balance between recall and precision than NearMiss; favors fewer false alarms.
Random Oversampling	Improves recall by increasing minority class samples; slight precision drop.
SMOTE Oversampling	Better precision than random oversampling but recall drops; synthetic samples may reduce generalization.

Table 27: Logistic Regression performance summary across resampling techniques

Neural Network

Resampling Method	Summary
Baseline	Strong generalization, good precision and recall.
Random Undersampling	Severe degradation; very low precision with unstable predictions.
NearMiss Undersampling	Moderate improvements over random undersampling but underperforms baseline.
KMeans Undersampling	Improves over other undersampling but still behind baseline.
Random Oversampling	High recall and precision; benefits from richer minority class.
SMOTE Oversampling	Strong precision and recall close to random oversampling; good minority class coverage.

Table 28: Neural Network performance summary across resampling techniques

Random Forest

Resampling Method	Summary
Baseline	Best overall performance, robust with original data distribution.
Random Undersampling	Reduced recall and F1; precision high, limits false positives.
NearMiss Undersampling	Poor generalization; aggressive removal leads to low precision and PR-AUC.
KMeans Undersampling	Balanced precision and recall; better than NearMiss; still below baseline and random undersampling.
Random Oversampling	Strong recall and precision; robust to duplicated minority samples.
SMOTE Oversampling	Slightly worse than random oversampling; synthetic patterns may impact generalization.

Table 29: Random Forest performance summary across resampling techniques

K-Nearest Neighbors

Resampling Method	Summary
Baseline	High precision, strong F1; scaling essential; hints of overfitting but good generalization.
Random Undersampling	Reduced performance due to information loss; PCA helps recovery slightly.
NearMiss Undersampling	Moderate performance but lower than random undersampling; feature selection important.
KMeans Undersampling	Best undersampling performance; dropping noisy features (e.g., Time) improves results.
Random Oversampling	Outperforms all undersampling methods; PCA improves performance further.
SMOTE Oversampling	Lower F1 and PR-AUC than random oversampling; sensitive to synthetic sample quality.

Table 30: K-Nearest Neighbors performance summary across resampling techniques

Key Takeaways:

- Logistic Regression and Random Forest perform best with minimal resampling or oversampling that preserves data distribution.
- Neural Networks benefit strongly from oversampling methods but suffer from undersampling due to the need for large, diverse datasets.
- K-Nearest Neighbors depend heavily on good feature scaling and selection; oversampling outperforms undersampling, and PCA can improve results.
- Undersampling generally reduces performance due to loss of majority class information, with NearMiss often being too aggressive.
- Oversampling improves minority class detection at some cost of noise and potential overfitting but provides better recall, which is crucial in fraud detection.
- Choosing a resampling technique depends on the model, dataset, and desired balance between precision and recall for the application.
- **Empirically, the best results across all models were achieved using the baseline (original) data without any resampling. This highlights that the first step when handling imbalanced datasets should be to evaluate model performance on the raw data. Resampling techniques alter the original distribution and may not always be as effective as assumed.**

Soft Voting Ensemble and Comparative Evaluation

In this section, we construct a soft Voting Classifier that ensembles the three best-performing base models: Neural Network (NN), K-Nearest Neighbors (KNN), and Random Forest (RF). Each of these models achieved their strongest performance under the baseline setup (i.e., without resampling). **As resampling methods such as SMOTE, NearMiss and KMeans had already reduced the individual models' performance, they were not applied to the Voting Classifier.** Instead, to maintain consistency and preserve optimal results, the Voting Classifier is evaluated solely on the original imbalanced dataset.

Best weight configuration:

$$\text{Weights} = (5, 3, 4) \quad \text{for} \quad \text{RF, NN, and KNN respectively}$$

This configuration was selected based on the individual performance rankings of the base classifiers, with RF consistently performing best, followed by KNN, then NN.

Performance Evaluation

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	99.67%	99.35%	100.00%	100.00%
Validation	88.37%	92.68%	84.44%	86.26%

Table 31: Voting Classifier performance at threshold 0.293

To assess the effectiveness of the Voting Classifier (VC), we compared it to the best standalone model—Random Forest—whose performance is summarized below:

Dataset	F1 Score	Precision	Recall	PR-AUC
Training	99.84%	99.67%	100.00%	100.00%
Validation	88.24%	93.75%	83.33%	85.06%

Table 32: Random Forest performance at threshold 0.364

Summary

Compared to the Random Forest model, the Voting Classifier delivers a slightly higher F1 score on the test set—our primary evaluation metric. It also achieves a higher PR-AUC, suggesting more reliable performance across different decision thresholds. Additionally, the ensemble improves recall, indicating better detection of fraud cases, while maintaining a reasonably high precision. These results show that the ensemble model strikes a better balance between minimizing false negatives and false positives, making it the most robust and effective model in our pipeline.

Feature Engineering

In this section, we shift our focus to improving model performance by enhancing the feature set. The goal is to extract more informative patterns from the data through carefully crafted features, rather than further tuning.
To guide the feature engineering process, we begin by analyzing the feature importances derived from the **Random Forest** model. This allows us to identify which features are contributing most significantly to the model's predictions.

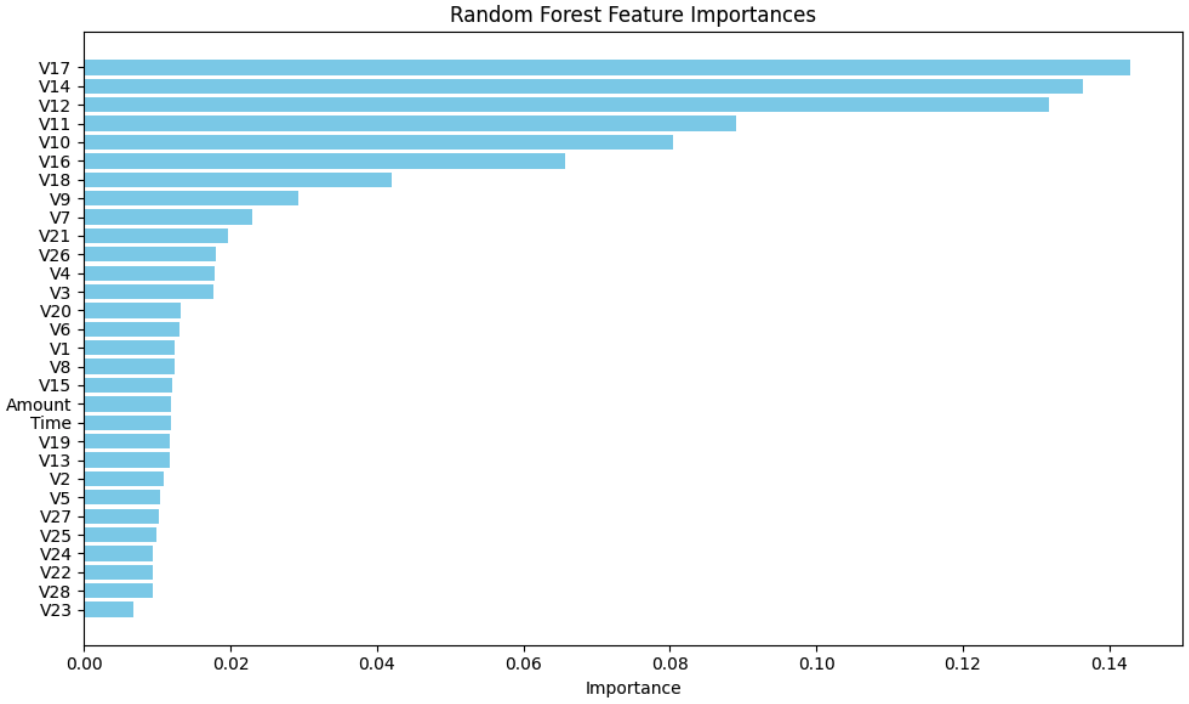

Figure 2: Random Forest Feature Importances

Next, we examine the linear correlation between the anonymized features `V1` through `V28` and the target `Class`. To better reveal these relationships, we first applied undersampling to the majority class—an essential step, as the extreme class imbalance can mask subtle but important correlations. This analysis provides statistical insight into which features are most associated with fraudulent transactions and helps identify promising candidates for transformation or interaction terms.

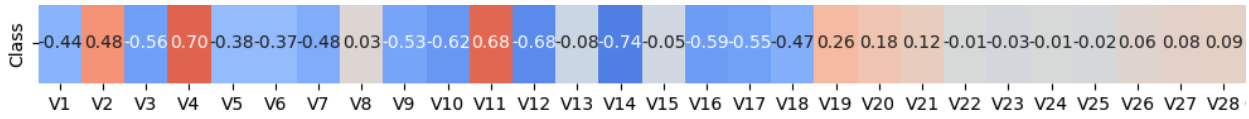

Figure 3: Correlation between V1–V28 Features and Class (see EDA for full table)

After extensive experimentation with various transformations and feature engineering techniques on `V17` and `V14`, including logarithmic, polynomial, and interaction terms, the following observations were made:

- Applying logarithmic transformations such as `log1p(V17^2)` and `log1p(V14^2)` decreased performance significantly.
- Polynomial features of `V17` (squared, cubed) also resulted in performance degradation.
- Dropping less correlated features like `V22`, `V23`, `V24`, and `V25` gave minor improvements but did not lead to substantial gains.
- The only notable improvement was achieved by adding the interaction feature `V17_V14` = `V17` $\times$ `V14`, which consistently improved model metrics.
- Various other interaction features using the top 10 important features identified by Random Forest were tested, but none surpassed the performance gain achieved by `V17_V14`.

Using this interaction feature alone, the model performance was:

Table 33: Model Performance Using Interaction Feature `V17_V14` (Threshold = 0.283)

Dataset	**F1 Score (%)**	**Precision (%)**	**Recall (%)**	**PR-AUC (%)**
Training	100.00	100.00	100.00	100.00
Validation	89.02	92.77	85.56	87.20

This represents a substantial gain in generalization performance and highlights the importance of carefully selected feature interactions in enhancing model effectiveness. Attempts to combine this interaction feature with other transformations or feature drops typically resulted in decreased performance, so it is recommended to use `V17_V14` alone for boosting the model.

It is important to note that constructing a Voting Classifier (VC) using the three diverse base models—Neural Network, K-Nearest Neighbors, and Random Forest—poses a challenge when attempting to achieve significant new performance improvements. Each model effectively learns and captures different aspects of the data distribution and decision boundaries, making their ensemble robust and comprehensive. Consequently, the scope for further enhancement through additional feature engineering or complex transformations is limited.

In this context, the notable increase in performance achieved by introducing the simple interaction feature `V17_V14` is particularly significant. While individually tuning or transforming features yielded marginal or negative results, this interaction alone boosted the testing F1 score by approximately 0.65% compared to the baseline. This improvement underscores the value of targeted feature engineering even in advanced ensemble models and suggests that carefully chosen interactions can still provide meaningful gains despite the ensemble's inherent robustness.

Final Model Training and Evaluation

After retraining the final model on the combined training and validation datasets, we evaluated its performance on the unseen test set using a threshold optimized for the F1 score on the validation split.

Table 34: Final Model Performance on Training+Validation and Test Sets (Threshold = 0.283)

Dataset	F1 Score	Precision	Recall	PR-AUC
Training + Validation	89.02%	92.77%	85.56%	87.20%
Test	86.73%	85.86%	87.63%	87.73%

The Voting Classifier (VC) maintained strong generalization when evaluated on the test set, achieving an F1 score of **86.73%**, with balanced **precision (85.86%)** and **recall (87.63%)**. These results reflect a robust fraud detection model that strikes a strong balance: it effectively identifies the majority of fraudulent transactions while minimizing false positives—critical in real-world deployment to avoid wrongly blocking legitimate users.

Notably, the PR-AUC score of **87.73%** further confirms the model's strong discriminative ability, especially important in imbalanced classification settings such as fraud detection.

Comparison with Base Models

To better understand the effectiveness of the Voting Classifier, we compared its performance to that of its constituent base models: Random Forest (RF), K-Nearest Neighbors (KNN), and a Neural Network (NN). Each model was evaluated on the test set after retraining and threshold optimization for F1 score on validation data.

Performance on the Test Set:

Model	F1 Score	Precision	Recall	PR-AUC
RF	84.69%	83.84%	85.57%	85.01%
KNN	**87.05%**	87.50%	86.60%	82.81%
NN	85.25%	**90.70%**	80.41%	85.70%
VC	86.73%	85.86%	**87.63%**	**87.73%**

Each base classifier brought unique strengths:

- **Random Forest** offered balanced precision and recall, achieving a solid F1 score with a reliable PR-AUC. It showed consistent performance but slightly underperformed the ensemble in F1 and PR-AUC.
- **KNN** achieved the highest F1 score (87.05%) among all models individually, indicating strong point estimates. However, its comparatively lower PR-AUC (82.81%) suggests weaker probabilistic calibration, potentially affecting performance when confidence scores are critical (e.g., threshold tuning, business rules).

- **Neural Network** excelled in precision (90.70%), demonstrating a cautious approach that minimizes false positives. However, this came at the cost of lower recall (80.41%), meaning it may miss more fraudulent cases—a serious drawback in this domain.

Why the Voting Classifier?

The Voting Classifier ensemble demonstrated **the best overall consistency** across all datasets and metrics. It successfully combined the strengths of RF, NN, and KNN, resulting in a model that not only performed well on the validation set but also generalized effectively to unseen test data. Its ability to stabilize performance and mitigate weaknesses in individual learners makes it ideal for deployment in fraud detection systems, where precision, recall, and robustness are all critical.

Looking at the individual base models, if we were to select a final model based solely on training and validation performance, **Random Forest** appeared to be the best performer, followed by KNN, with the Neural Network slightly trailing behind. However, this ranking did not fully hold when evaluated on the unseen test set, where surprising shifts in performance occurred—highlighting that relying on a single model based on validation results alone could lead to suboptimal choices.

Thus, the ensemble approach of the Voting Classifier helps mitigate such unpredictable behaviors by leveraging the complementary strengths of the base models, resulting in improved generalization and reliability on new data.

Conclusion

The final deployed model—a **Voting Classifier optimized for F1 score using validation data**—achieved a strong trade-off between sensitivity and specificity. Its superior balance, consistency across datasets, and high discriminative power (as shown by PR-AUC) make it a suitable and reliable choice for real-world fraud detection applications.