Fraud Detection in Digital Transactions: Stage 1 Report

Tom Delahaye, Gabriel Carlotti, Pierre Briand, Kentin Guillemot

November 15, 2024

Abstract

This report presents the first stage of a multi-phase project focused on fraud detection in digital payments. The goal of this stage is to implement baseline machine learning models to identify fraudulent transactions using various features extracted from transaction data. The outcomes of this stage will serve as a foundation for more advanced models in subsequent phases.

Contents

1 Business Scope

The rapid expansion of digital payments has revolutionized the way transactions are conducted, offering unparalleled convenience to consumers and businesses alike. However, with the increasing volume of online and card transactions, fraud has become a significant threat. Financial institutions face a growing challenge to accurately identify and prevent fraudulent activities while ensuring a seamless experience for legitimate users.

Fraud detection systems play a crucial role in safeguarding digital transactions, aiming to minimize financial losses and maintain customer trust. The challenge lies in building models that can effectively detect fraudulent transactions in real-time without compromising user convenience. This project aims to leverage data analysis techniques and machine learning approaches to develop a robust fraud detection framework, starting with baseline models in Stage 1.

2 Problem Formalisation and Methods

2.1 Algorithm Description

In this stage, we implemented three fundamental machine learning algorithms to serve as a baseline for detecting fraudulent transactions. These algorithms include:

- **Logistic Regression**: A linear classification model that predicts the probability of an event occurring (in this case, a fraudulent transaction) based on the input features. It uses the logistic function to model the probability of the binary outcome and finds the best-fitting linear boundary between the classes.
- **K-Nearest Neighbors (KNN)**: A non-parametric algorithm that classifies a data point based on the majority vote of its nearest neighbors. It assigns a class to the new data point by finding the k closest points in the training data, making it particularly useful for problems where the data distribution is not well-defined.

2.2 Limitations

While the algorithms implemented in this stage provide a solid foundation for detecting fraudulent transactions, each comes with its own set of limitations that may affect the quality of the expected results:

- **Logistic Regression**: This model assumes a linear relationship between the features and the target variable, which may not hold true in complex datasets. It can struggle with non-linear data and is sensitive to outliers, which can significantly impact its performance. Given the linear nature of this model, we may expect suboptimal performance if the relationships between features and the target variable are non-linear.
- **K-Nearest Neighbors (KNN)**: KNN can be inefficient for large datasets because it requires calculating the distance between the new data point and every point in the training set. It is also sensitive to the choice of k (the number of neighbors) and the distance metric used. Moreover, KNN performs poorly when the data has many irrelevant features or when the classes overlap significantly. Due to its reliance on

distance calculations, KNN may struggle with high-dimensional data and may not perform well in the presence of noisy or irrelevant features.

3 Methodology

3.1 Data Description and Exploration

The dataset used for this stage contains **1 million rows** with **no missing values**, ensuring a complete and consistent dataset for analysis. The features included in the dataset were thoroughly examined and preprocessed in the exploratory data analysis (EDA) notebook, `explo_EDA.ipynb`, where initial data cleaning and feature analysis were performed.

- **distance_from_home**: The distance between the user's home location and where the transaction took place.
- **distance_from_last_transaction**: The distance from the location of the previous transaction.
- **ratio_to_median_purchase_price**: The ratio of the transaction amount to the median purchase price.
- **repeat_retailer**: Indicates whether the transaction was made at the same retailer as the previous one (1 = Yes, 0 = No).
- **used_chip**: Indicates if the transaction was processed through a chip-enabled credit card (1 = Yes, 0 = No).
- **used_pin_number**: Indicates whether a PIN number was used during the transaction (1 = Yes, 0 = No).
- **online_order**: Indicates whether the transaction was an online order (1 = Yes, 0 = No).
- **fraud**: The target variable indicating if the transaction is fraudulent (1 = Fraud, 0 = Not Fraud).

The dataset shows a clear case of **imbalanced data**, as only **9%** of the transactions are labeled as fraudulent. This imbalance poses a significant challenge, as standard machine learning models may tend to favor the majority class (non-fraudulent transactions), potentially leading to suboptimal performance in detecting fraudulent transactions. Addressing this issue is crucial to ensure that the models are trained effectively and can generalize well to unseen data.

To avoid data leakage, the problems of imbalanced data and outliers will be addressed in the preprocessing step (Section 3.3). This approach ensures that synthetic data generated using techniques like SMOTE is not influenced by the presence of extreme outliers, leading to more robust and representative models.

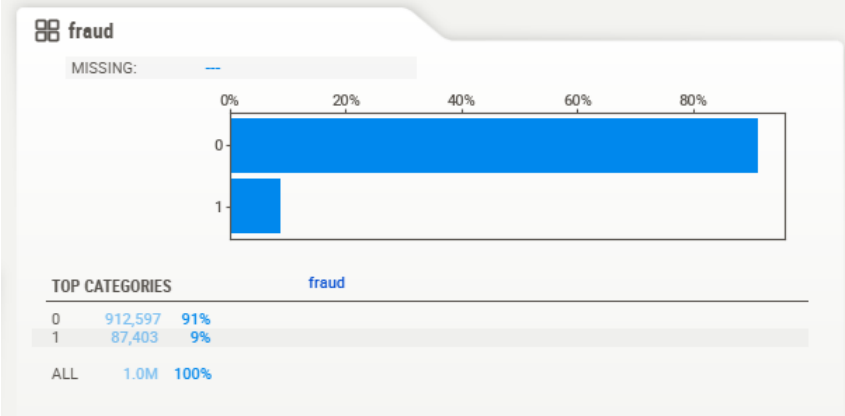

Figure 1: Distribution of Fraud in the Dataset

3.2 Feature Analysis and Justification

To better understand the relationship between our features and the target variable (fraud), we conducted an exploratory data analysis using **Sweetviz** and visualized the distribution of fraud across the binary features.

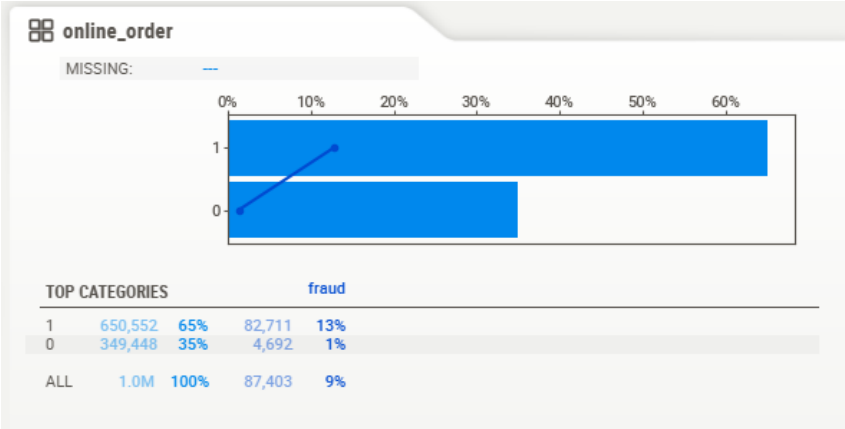

Figure 2: Fraud Rate by Online Order

In Figure 2, we observe that **13% of online transactions** are fraudulent, compared to only **1%** of in-store (physical) transactions. This significant difference suggests that online transactions are more susceptible to fraud, making the 'online_order' feature a strong predictor for fraudulent behavior.

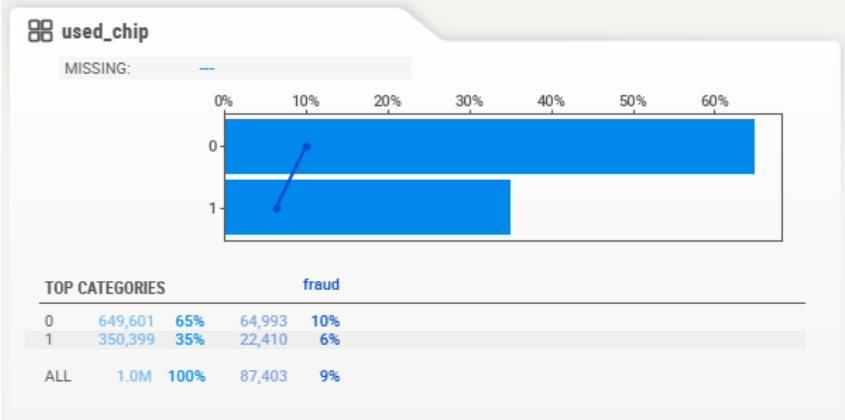

Figure 3: Fraud Rate by Chip Usage

As shown in Figure 3, **10% of transactions without chip usage** are fraudulent, while only **6%** of transactions using a chip are fraudulent. This indicates that chip usage reduces the likelihood of fraud, highlighting the importance of the 'used_chip' feature for our model.

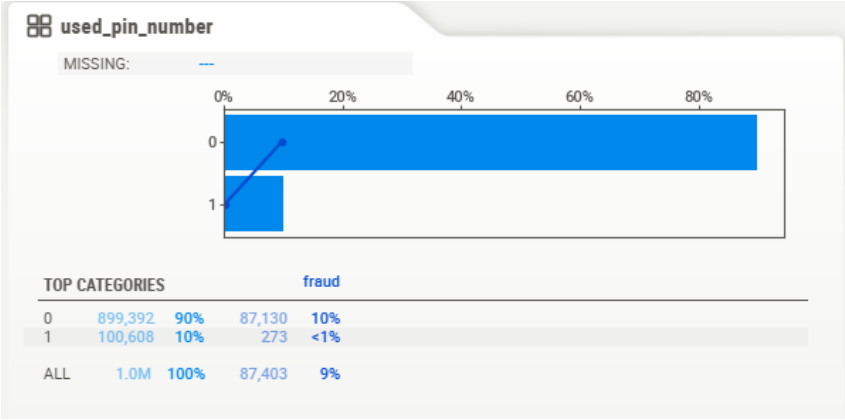

Figure 4: Fraud Rate by PIN Number Usage

In Figure 4, we see that transactions without PIN usage have a fraud rate of **10%**, compared to less than **1%** for transactions where a PIN was used. This stark contrast demonstrates that using a PIN significantly decreases the risk of fraud, making the 'used_pin_number' feature a valuable addition to our model.

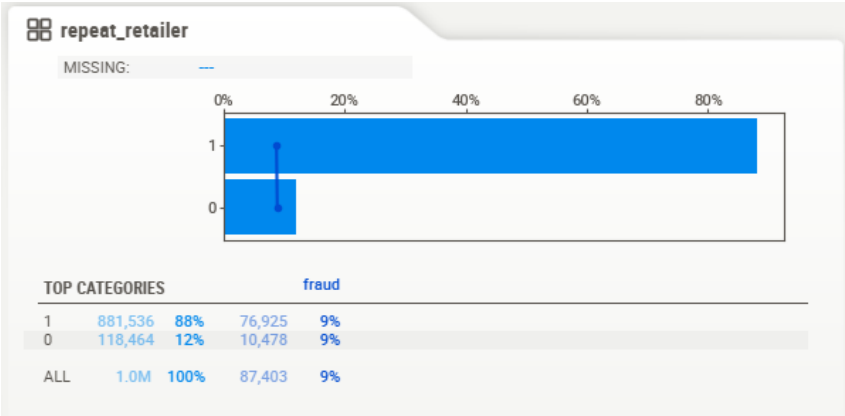

Figure 5: Fraud Rate by Repeat Retailer

The analysis in Figure 5 shows no significant difference in the fraud rate between repeated and non-repeated transactions (both showing a fraud rate of **9%**). This suggests that the 'repeat_retailer' feature does not provide meaningful predictive power for identifying fraud and may be considered for removal in future stages.

Conclusion: Based on the analysis, the binary features 'online_order', 'used_chip', and 'used_pin_number' show a strong correlation with fraudulent activities and will be retained. In contrast, 'repeat_retailer' does not appear to be informative and could be dropped from the model in subsequent analyses.

3.3 Data Splitting for Train/Test

To evaluate our models effectively, we used a standard **80/20 train-test split**. This approach ensures a robust assessment of the model's performance on unseen data. Additionally, we applied "`stratify = y`" during the split to maintain the same proportion of fraudulent transactions (9%) in both the training and testing sets, preserving the class distribution and preventing bias in the evaluation.

Before applying any sampling technique, we first removed the outliers from the training data. We used the `z-score` method from `scipy.stats` to identify and remove outliers. The **z-score** measures how many standard deviations a data point is from the mean. Data points with a z-score greater than 3 (in absolute value) were considered outliers and removed.

After handling the outliers, we applied **SMOTE (Synthetic Minority Over-sampling Technique)** to address the problem of imbalanced data. SMOTE generates synthetic samples for the minority class (fraud) by interpolating between existing data points. This technique helps balance the dataset, allowing the models to learn better from the minority class without simply duplicating existing samples.

We chose to remove the outliers first and then apply SMOTE to avoid generating synthetic samples based on extreme values. If outliers were included in the SMOTE process, the synthetic data could be biased towards these outlier points, making the generated

samples less representative of typical fraudulent transactions.

It is important to note that these preprocessing steps were only applied to the **training set** and not to the test set. This approach prevents **data leakage**, which occurs when information from the test set influences the training process. By ensuring that the test set remains untouched, we can fairly evaluate the model's performance on unseen data, reflecting its true generalization capability.

3.4 Algorithm Implementation and Hyperparameters

The two models selected for this stage are **Logistic Regression** and **K-Nearest Neighbors (KNN)**. Both algorithms were implemented using the `scikit-learn` library, ensuring clean and modular code that adheres to best practices for reproducibility. Helper functions were developed for data preprocessing, model training, and evaluation, providing a streamlined and maintainable implementation process.

For **Logistic Regression**, we used the `LogisticRegression` class. This model was trained on the features to predict the probability of a transaction being fraudulent. The primary hyperparameter, the regularization parameter `C`, controls the trade-off between fitting the training data well and keeping the model simple (i.e., avoiding overfitting).

For **K-Nearest Neighbors (KNN)**, we used the `KNeighborsClassifier` class. The main hyperparameter for KNN is the number of neighbors `k`, which determines how many nearest neighbors are considered when making a prediction.

Hyperparameter Optimization: We performed hyperparameter tuning using **GridSearchCV** for both models. This technique systematically searches through a specified set of hyperparameters and evaluates each combination using cross-validation. For Logistic Regression, we searched for the optimal value of `C` (e.g., `[0.001, 0.01, 0.1, 1, 10]`). For KNN, we tested different values of `k` (e.g., `k: [1, 3, 5, 7, 9]`). The cross-validation process helps to select the hyperparameters that yield the best generalization performance, reducing the risk of overfitting.

Monitoring of Underfitting/Overfitting: To ensure that our models (Logistic Regression and KNN) did not overfit the data, we applied several evaluation techniques:

- **Train-test performance comparison**: We compared the models' performance on both the training set and the test set. If the models performed significantly better on the training data than on the test data, it indicated a risk of overfitting.
- **Learning curves**: We plotted the learning curves for both models, observing how their performance evolved with increasing training data size. A persistent gap between the training and test curves suggested a potential overfitting issue.
- **Cross-validation**: We utilized cross-validation during hyperparameter tuning with `GridSearchCV`, which provided an evaluation of the models across different data subsets. A low variance between the cross-validation scores indicated good generalization ability.

By applying these techniques, we were able to detect and mitigate the risk of overfitting while optimizing the hyperparameters.

4 Results

4.1 Metrics

In this section, we present the evaluation metrics for the two models, **Logistic Regression** and **K-Nearest Neighbors (KNN)**. The performance was assessed using precision, recall, F1-score, and accuracy metrics, as well as confusion matrices for a detailed analysis. Additionally, we used the **F2-score** to guide the hyperparameter tuning in our GridSearch. The choice of the F2-score was motivated by the need to place more emphasis on recall, as our primary goal is to minimize false negatives in the context of fraud detection, where failing to detect a fraudulent transaction is more costly than a false positive.

4.1.1 Logistic Regression

The classification report for Logistic Regression is shown in Figure 6. The model achieved a high overall accuracy of 93%. The precision for the positive class (fraudulent transactions) was 0.54, while the recall was very high at 0.97, resulting in an F1-score of 0.69. This indicates that the model is highly sensitive in detecting fraudulent transactions but sacrifices precision, leading to a higher number of false positives.

The confusion matrix in Figure 7 provides a clearer view of the model's predictions. We observe:

- True Negatives (0 correctly classified): 168,042
- False Positives (0 classified as 1): 14,477
- False Negatives (1 classified as 0): 465
- True Positives (1 correctly classified): 17,016

The high recall for the positive class confirms the model's focus on minimizing false negatives, which is crucial in a fraud detection context.

	precision	recall	f1-score	support
0	1.00	0.92	0.96	182519
1	0.54	0.97	0.69	17481
accuracy			0.93	200000
macro avg	0.77	0.95	0.83	200000
weighted avg	0.96	0.93	0.93	200000

Figure 6: Classification report for Logistic Regression.

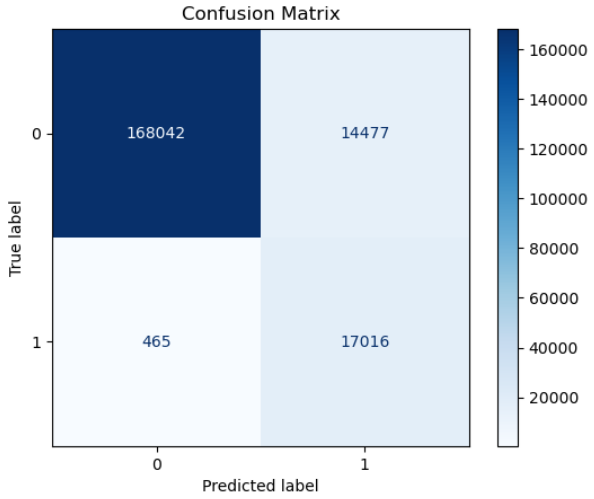

Figure 7: Confusion Matrix for Logistic Regression.

4.1.2 K-Nearest Neighbors (KNN)

The classification report for the KNN model is shown in Figure 8. The KNN model achieved a significantly higher accuracy of 100%. The precision for the positive class was 0.96, and the recall was perfect at 1.00, resulting in an F1-score of 0.98. This suggests that the KNN model is very effective at distinguishing between fraudulent and non-fraudulent transactions.

The confusion matrix in Figure 9 highlights the KNN model's performance:

- True Negatives (0 correctly classified): 181,864
- False Positives (0 classified as 1): 655
- False Negatives (1 classified as 0): 83
- True Positives (1 correctly classified): 17,398

The near-perfect performance across all metrics indicates that the KNN model generalizes well and is highly reliable for fraud detection.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	182519
1	0.96	1.00	0.98	17481
accuracy			1.00	200000
macro avg	0.98	1.00	0.99	200000
weighted avg	1.00	1.00	1.00	200000

Figure 8: Classification report for K-Nearest Neighbors (KNN).

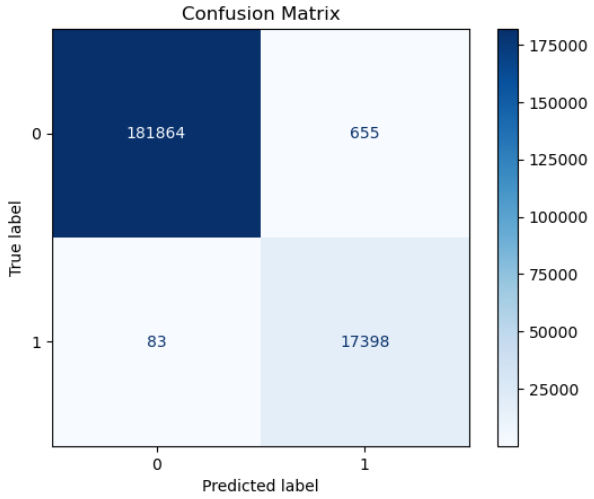

Figure 9: Confusion Matrix for K-Nearest Neighbors (KNN).

4.2 Overfitting Analysis

To evaluate the risk of overfitting in our models, we utilized **learning curves**. Learning curves plot the model's performance (using the F2-score in our case) against the size of the training set, providing insights into how well the model generalizes to new data. They are crucial for detecting overfitting, as they allow us to observe the relationship between training and validation scores. A large gap between these scores would indicate potential overfitting, while similar scores suggest good generalization.

4.2.1 Logistic Regression

The learning curve for Logistic Regression (Figure 10) shows no clear signs of overfitting for the following reasons:

- **No perfect training score**: The training F2-score is below 1.0, indicating that the model does not perfectly fit the training data.
- **Small gap between training and validation scores**: The validation score is close to the training score, suggesting strong generalization to unseen data.
- **Consistent increase in scores**: Both the training and validation scores increase as the size of the training set grows, demonstrating that the model continues to learn effectively without overfitting.

Overall, the Logistic Regression model shows balanced learning and robust generalization.

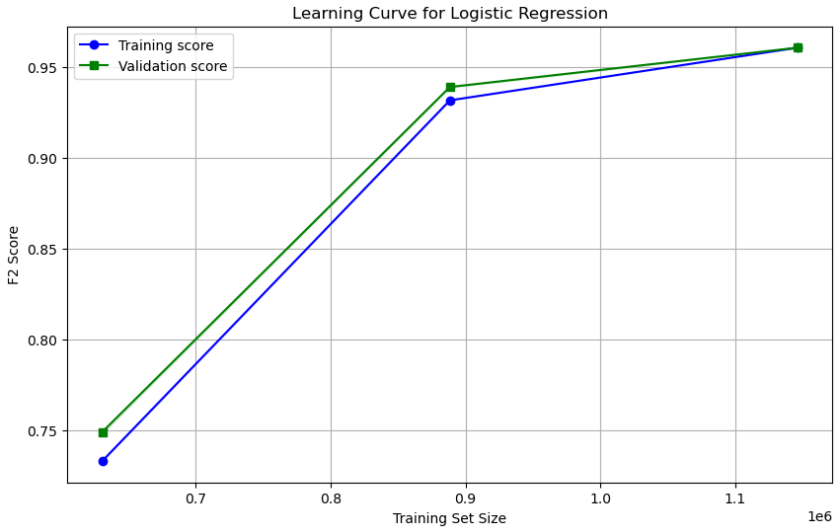

Figure 10: Learning Curve for Logistic Regression using F2-score.

4.2.2 K-Nearest Neighbors (KNN)

Initially, the learning curve for KNN (Figure 11) raised concerns about potential overfitting:

- **Perfect training score**: The training F2-score is consistently 1.0, indicating that the model fits the training data flawlessly.
- **Slight gap in validation score**: Although the validation F2-score is slightly lower (0.998), the gap is minimal, suggesting good generalization on the validation set.

Despite the high validation scores, the perfect training score hinted at potential overfitting. To mitigate this risk, we increased the number of neighbors (`n_neighbors`) from 3 to 5 and re-evaluated the model. The updated learning curve (Figure 12) confirmed that the risk of overfitting was reduced, as shown by:

- **Minimal gap between training and validation scores**: The small difference indicates good generalization capability.
- **Consistent performance across different values of `n_neighbors`**: This suggests model stability and reduces the likelihood of overfitting.
- **High validation scores close to training scores**: This confirms that the model is not overly fitted to the training data.

Overall, the KNN model demonstrates robust generalization with a low risk of overfitting after adjusting the hyperparameters.

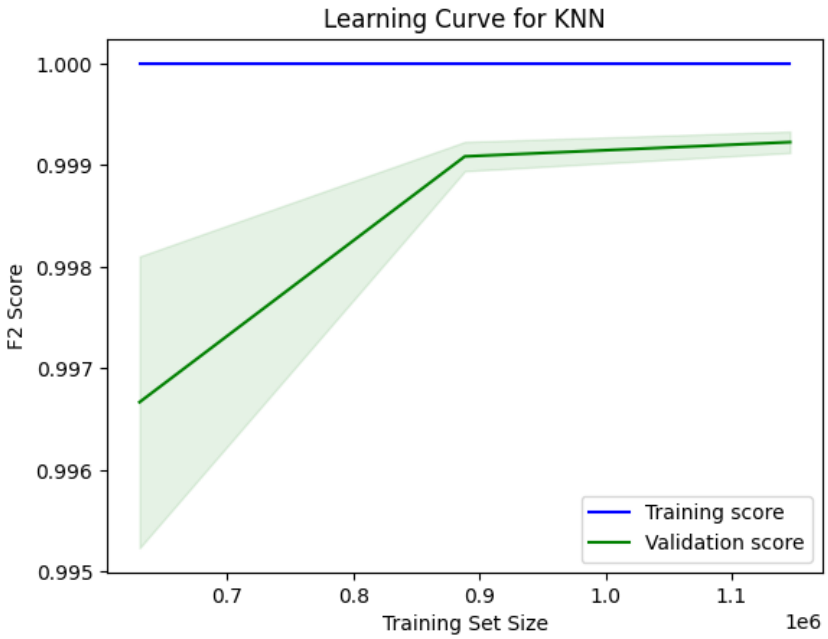

Figure 11: Initial Learning Curve for KNN (n_neighbors=3) using F2-score.

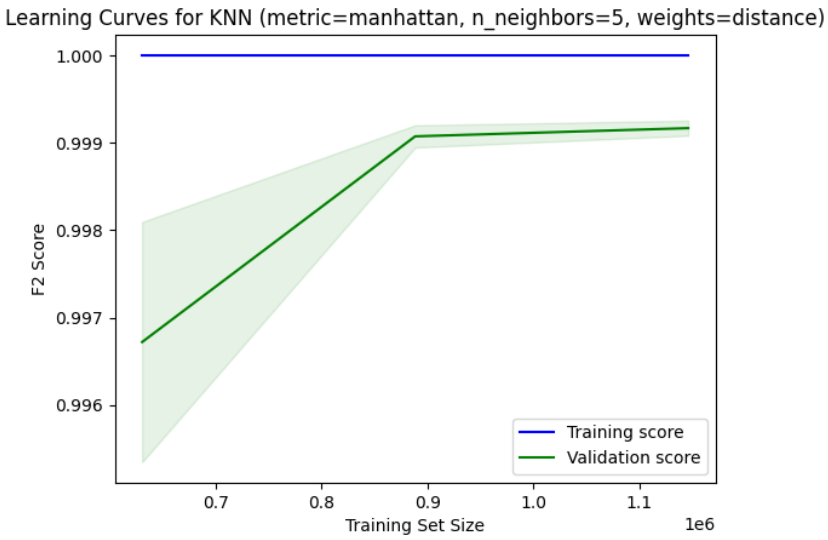

Figure 12: Updated Learning Curve for KNN (n_neighbors=5) using F2-score.

4.3 Evaluation

We evaluated the performance of both models, **Logistic Regression** and **K-Nearest Neighbors (KNN)**, using precision, recall, F1-score, F2-score, and confusion matrices.

4.3.1 Logistic Regression

Logistic Regression achieved an accuracy of 93%, with strong recall for the fraudulent class. The model shows a slight trade-off between precision and recall:

- High recall (0.97) for the fraudulent class indicates effective detection of fraudulent transactions.
- Lower precision (0.54) suggests more false positives, but this is acceptable in our context as recall is prioritized.
- The confusion matrix shows a low number of false negatives, highlighting the model's sensitivity in identifying fraud.

4.3.2 K-Nearest Neighbors (KNN)

The KNN model performed exceptionally well, with an accuracy close to 100%. However, caution is needed due to the perfect training score:

- High precision (0.96) and perfect recall (1.0) for the fraudulent class indicate balanced detection and minimal false negatives.
- The confusion matrix shows very few false positives and false negatives, demonstrating strong model performance.
- Despite the high scores, the perfect training score suggests potential overfitting, which was mitigated by adjusting the number of neighbors.

4.3.3 Comparison of Models

Overall, KNN slightly outperforms Logistic Regression, offering better precision and balanced detection. Logistic Regression, while slightly less precise, shows strong recall, making it a reliable choice when prioritizing the detection of fraudulent cases.

Both models demonstrate good generalization, with KNN showing a small risk of overfitting that was addressed during hyperparameter tuning.

5 Discussion and Conclusion

In summary, both models show strong performance, with KNN demonstrating a slight edge in precision and generalization. While Logistic Regression proved to be highly effective in identifying fraudulent transactions, its lower precision indicates a higher rate of false positives. This can be acceptable in a fraud detection context, where the cost of missing fraudulent transactions is greater than investigating false alerts. On the other hand, KNN provided excellent results with near-perfect precision and recall, though there were initial concerns about potential overfitting, which were addressed through careful hyperparameter tuning.

Despite the success of these models, it is important to recognize the limitations of our approach. Logistic Regression may struggle with complex, non-linear patterns in the data, while KNN's computational cost and sensitivity to the choice of `n_neighbors` can impact its scalability, especially on larger datasets.

For the next stage of this project, we will build upon these findings by exploring more sophisticated models, specifically decision tree-based algorithms like **Random Forest** and **XGBoost**. These models are well-suited for handling complex data structures and can offer improved performance by capturing non-linear relationships. Additionally, ensemble methods may provide better robustness and further reduce the risk of overfitting, giving us the opportunity to refine our fraud detection capabilities.

Overall, the results from this stage provide a solid foundation for our next steps, as we continue to enhance the model's predictive power and adaptability in detecting fraudulent transactions.