Enhancing Credit Card Fraud Detection: A Multifaceted Approach Using Machine Learning and Adaptation to Non-Gaussian Distributions

Ablay Mustafa

February 2024

1 Abstract

This paper addresses the urgent issue of credit card fraud by using existing techniques for fraud detection, including Linear Regression, Support Vector Machines (SVM), Random Forests, Statistical Analysis, and Behavioral Analytics. My goal, as an undergraduate student, is to replicate and extend the findings presented in the "Data mining for credit card fraud: A comparative study" paper. To implement Linear Regression, I will model the relationship between variables to identify potential fraud indicators. Support Vector Machines (SVM) will involve classifying transactions into normal and fraudulent categories based on distinct patterns. Random Forests will be employed to construct an ensemble of decision trees, enhancing accuracy in detecting anomalous transactions. For Statistical Analysis, I will utilize techniques like hypothesis testing and probability distributions to analyze transaction data for irregularities. Behavioral Analytics will involve studying user behavior over time, identifying deviations from typical patterns as potential fraud signals. Considering the assumption of Gaussian distribution in existing research, I aim to expand these techniques to datasets with non-Gaussian distributions. This involves adapting algorithms to handle different statistical properties, ensuring robustness across diverse datasets. By doing so, I intend to address a gap in current literature and enhance the applicability of credit card fraud detection methods in real-world scenarios.

2 Dataset

This paper examines a dataset provided by the Kaggle community, documenting credit card transactions from September 2013 involving European cardholders. The dataset encompasses two days of transactions, totaling 284,807, of which 492 are identified as fraudulent. The dataset primarily consists of numerical input variables derived from a Principal Component Analysis (PCA). Due to confidentiality constraints, the original features and detailed background information are not disclosed. The PCA-transformed features are labeled V1 through V28. The only attributes not transformed by PCA are 'Time' and 'Amount'. 'Time' records the seconds elapsed since the first transaction in the dataset, while 'Amount' denotes the transaction value, useful for cost-sensitive learning models. The 'Class' variable indicates the transaction status, where a value of 1 signifies a fraudulent transaction and 0 indicates a non-fraudulent one. The dataset comprises the following columns:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'], dtype='object')
```

(Kaggle, 2021)

Train-Test Split

The `train_test_split` function algorithmically partitions the dataset into training and testing sets. This process, crucial for evaluating the performance of a machine learning model, can be described as follows:

1. **Shuffling:** The dataset is randomly shuffled to ensure the training and testing sets are representative of the data's overall distribution.
2. **Splitting:** The shuffled data is then divided into training and testing sets based on a specified ratio in this study the data is split as 80% for training and 20% for testing.

1. Random Forest Algorithm

The random forest approach produces highly accurate predictions and can handle a very large number of input variables without overfitting, which is essential for the study of credit card fraud detection. Leo Breiman, who is considered a father of Random Forest, suggested that each tree in the forest is constructed by selecting a random set of features from the dataset at nodes to split and test it on the training set. (Breiman, 1984) The approach involves the construction of a decision tree, which is a non-linear predictive model to differentiate the entities into distinct categories, in our case fraudulent and non-fraudulent.

1.1 Decision Tree Construction

The mathematical foundation of construction of a decision tree involves optimization of impurity measures and recursive splitting enable them to effectively capture patterns in the data. However, considerations around overfitting, model complexity, and the handling of feature interactions are essential for building effective decision tree models. Decision trees in a Random Forest use a random subset of features at each split, chosen based on Gini impurity or Entropy, which will be discussed in the next subsection.

1.1.1 Mathematical Foundation

The construction of a decision tree can be viewed as an optimization problem, where the objective is to find the tree structure that best separates the instances into homogeneous groups or classes.

$$\min_{w\neq 0,\gamma} \left(\frac{1}{m} ||(-Aw + e\gamma)_+||_1 + \frac{1}{k} ||(Bw - e\gamma)_+||_1 \right) \tag{1}$$

where, A and B are two points in the n-dimensional space R^n, w is an n-dimensional "weight" vector representing the normal to an optimal "separating" plane, γ is a real number. (Bennett, 1992)

This objective is typically achieved through the maximization of a purity measure or the minimization of an impurity measure at each split. (Breiman, 1984) As was mentioned in the previous subsection two common measures are:

- **Gini Impurity:** Gini impurity measures how often a randomly chosen entity from the data would be incorrectly identified if it was randomly labeled according to the distribution of labels in the subset. Defined for a set of items as $1 - \sum_{i=1}^{J} p_i^2$, where p_i is the fraction of items labeled with class i in the set, and J is the number of classes. It measures the frequency at which a randomly chosen element would be incorrectly labeled. (Zhang, Gionis, 2022)
- **Entropy:** Entropy is a measure from information theory that quantifies the uncertainty involved in predicting the class of an instance. Information gain is then the decrease in entropy as a result of the split. Measures the amount of uncertainty or disorder, defined as $-\sum_{i=1}^{n} p_i \log_2 p_i$, where p_i is the proportion of elements belonging to class i. (Thomas, 1991)

The algorithm selects the split that results in the largest decrease in Gini impurity or entropy, aiming for homogeneity.

The selection of the best feature and value to split on at each node is based on the calculation of the Gini impurity or information gain for each potential split. The algorithm evaluates each feature and each possible value of these features as a potential split point, selecting the one that results in the highest gain (or lowest impurity).

1.1.2 Construction Methodology

Recursive Binary Splitting

The process starts at the root of the tree and involves recursively partitioning the data into subsets. This binary splitting continues until a stopping criterion is met, which might include:

- Reaching a maximum specified depth of the tree. - The number of instances in a node falling below a threshold. - The improvement in impurity falling below a threshold, indicating additional splits do not significantly increase purity.

Pruning

To address the issue of overfitting, where the tree models the noise in the training data rather than the underlying distribution, trees are often pruned. Pruning involves cutting back parts of the tree that have little contribution to predictive power. One common approach is cost complexity pruning, which introduces a trade-off between the tree's complexity and its fit to the data, penalized by a complexity parameter α.

1.1.3 Extension of the theory on non-Gaussian distribution

In the context of Random Forests, the non-Gaussian distribution fitting into the algorithm is highlighted as part of the broader objective to handle different statistical properties across diverse datasets. Random Forests are an ensemble learning technique that builds several decision trees during the training process. For classification tasks, this method outputs the most frequent class among the trees (mode), and for regression tasks, it computes the average prediction (mean) from all the trees. Random Forests inherently do not assume any specific distribution for the input variables, making them versatile in managing datasets with non-Gaussian distributions.

The adaptation to non-Gaussian distributions involves modifying algorithms to accommodate different statistical properties to ensure robustness. This is particularly relevant for Random Forests in the fraud detection context, as financial transaction data often exhibit skewed distributions, heavy tails, or multi-modal characteristics that deviate from the Gaussian assumption. The non-Gaussian distribution's fit in the Random Forest algorithm signifies an advanced approach to making the model more flexible and applicable to real-world data, which may not adhere to the Gaussian distribution.

The adjustment for non-Gaussian distributions in Random Forests could involve preprocessing steps, such as transformation of variables to normalize data distributions or employing techniques that make the algorithm more sensitive to the distribution of the data. However, since Random Forests do not make strong assumptions about the distribution of the input data, the primary fit for non-Gaussian distributions might not involve a fundamental change to the Random Forest algorithm itself but rather an emphasis on its natural compatibility with diverse data distributions and the potential for preprocessing strategies to maximize this compatibility.

Considering these adaptations for non-Gaussian distributions ensures the detection method remains effective across various datasets, enhancing the credit card fraud detection methods' applicability in real-world scenarios, as aimed by the report.

This analysis demonstrates the significance of adapting fraud detection algorithms, like Random Forests, to accommodate non-Gaussian distributions, addressing a gap in current literature and aiming to improve the robustness and applicability of these methods in detecting anomalous transactions within credit card data.

1.2 Bootstrap Aggregating (Bagging)

Bootstrap Aggregating, or bagging, is a machine learning ensemble technique designed to improve the stability and accuracy of machine learning algorithms. It involves training multiple models using randomly drawn subsets of the training data and then aggregating their predictions to form a final result. The method is particularly effective in reducing variance and avoiding overfitting. Thus, this paper uses it to enhance the random forest algorithm. (Farid, Rahman, Rahman, 2011)

1. **Bootstrap Sampling**

   Given a dataset D of size N, bagging generates new training sets D_i (for $i = 1$ to B, where B is the number of bootstrap samples), each of size N', by sampling from D uniformly and with replacement.

This process is known as bootstrap sampling. The sampled observations in each D_i are used to train a new model M_i. Because sampling is with replacement, some observations may appear multiple times in a single D_i, while others may not appear at all.

2. **Model Training**

   For each bootstrap sample D_i, a model M_i is trained. The training process is independent for each model, allowing it to be performed in parallel. The choice of model can vary, but decision trees are a common choice for bagging because of their tendency to overfit on their training data, a trait that bagging can help mitigate.

3. **Aggregation**

   Once all models M_i are trained, their predictions are aggregated to form a final prediction. The aggregation method depends on the type of prediction being made: For regression problems, the final prediction is often the average of all model predictions:

$$\hat{y} = \frac{1}{B} \sum_{i=1}^{B} M_i(x) \tag{2}$$

   where x is a new input and $\hat{y}$ is the predicted output.

   For classification problems, the final prediction is typically the mode (i.e., the most common prediction) of all model predictions:

$$\hat{y} = \text{mode}\{M_1(x), M_2(x), \ldots, M_B(x)\} \tag{3}$$

   where x is a new input and $\hat{y}$ is the predicted class.

   The final prediction is made by aggregating the predictions from all individual trees:

   - For classification tasks, this is typically done by majority voting.
   - For regression tasks, the predictions are averaged.

   These processes ensure the Random Forest algorithm's effectiveness in both classification and regression tasks, making it a powerful tool in the machine learning toolkit.

   After applying random forest algorithm, the following results were obtained

	Predicted
Actual	0
1	
0	56854
7	
1	22
79	

Table 1: Confusion Matrix

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56861
1	0.92	0.78	0.84	101
accuracy			1.00	56962
macro avg	0.96	0.89	0.92	56962
weighted avg	1.00	1.00	1.00	56962

4. **Mathematical Justification**

   The effectiveness of bagging comes from its ability to reduce variance without increasing bias. If the models M_i are unbiased estimators, the aggregated model $\hat{y}$ will also be unbiased. The reduction in variance is due to the averaging of multiple predictions, which tends to cancel out errors from individual models as long as the errors are not correlated. Mathematically, if the errors made by the models are independent and have variance σ^2, the variance of the aggregated predictions is $\frac{\sigma^2}{B}$ under simple averaging for regression.

   Bagging is most effective when the base models are highly accurate yet exhibit substantial variability in their predictions across different training sets, making it a powerful tool for improving the performance of models that are sensitive to the specificities of the training data.

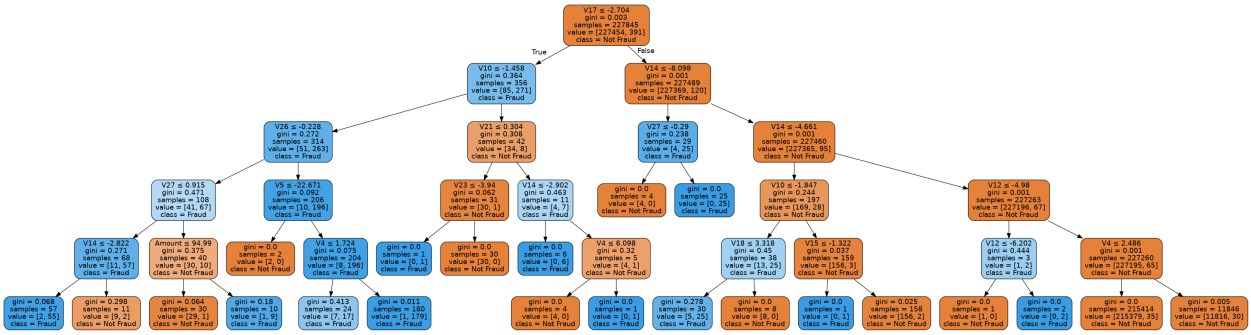

Figure 1: Decision tree

1.3 Model Evaluation

1.3.1 Model Accuracy

- **0.9994908886626171**: This is the accuracy of the model, representing the proportion of true results (both true positives and true negatives) among the total number of cases examined. The model's accuracy is approximately 99.95%, indicating that it correctly predicts nearly all transactions as either fraudulent or legitimate.
- **True Negatives (TN) = 56854**: The number of legitimate transactions correctly identified as legitimate.
- **False Positives (FP) = 7**: The number of legitimate transactions incorrectly identified as fraudulent.
- **False Negatives (FN) = 22**: The number of fraudulent transactions incorrectly identified as legitimate.
- **True Positives (TP) = 79**: The number of fraudulent transactions correctly identified as fraudulent.

1.3.2 Confusion Matrix

The confusion matrix is provided as a 2x2 array:

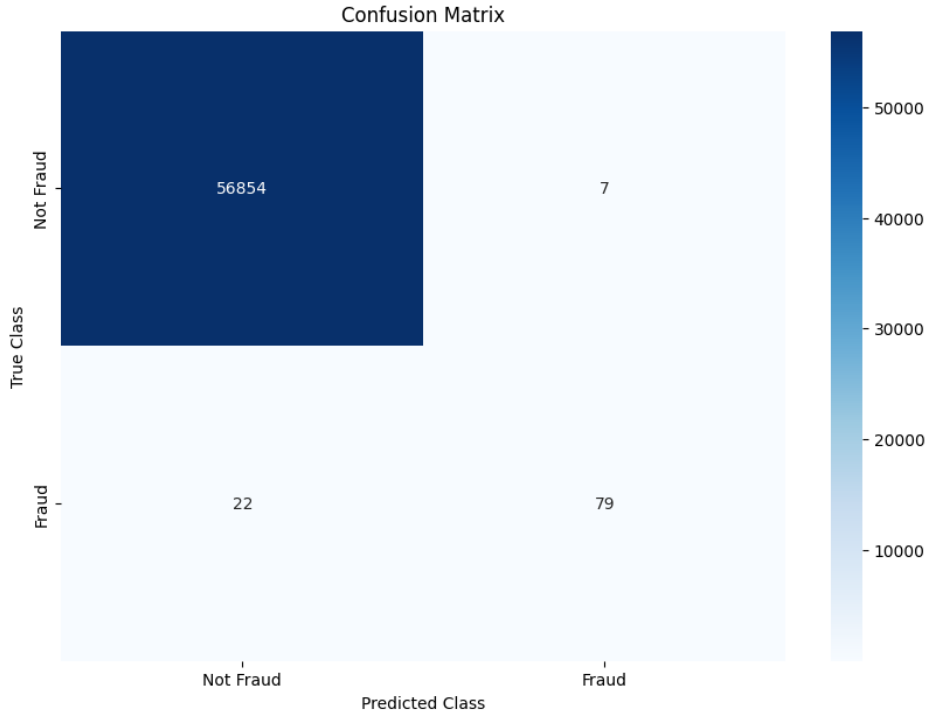

Figure 2: Confusion Matrix

1.3.3 Classification Report

The classification report provides detailed metrics for each class (0 for legitimate transactions and 1 for fraudulent transactions) as well as averages:

- **Precision (Class 0 = 1.00, Class 1 = 0.92)**: The precision for legitimate transactions is perfect, indicating no legitimate transactions were incorrectly labeled as fraudulent. The precision for fraudulent transactions is 0.92, meaning that when the model predicts a transaction is fraudulent, it is correct 92% of the time.
- **Recall (Class 0 = 1.00, Class 1 = 0.78)**: The recall for legitimate transactions is also perfect. For fraudulent transactions, the recall is 0.78, indicating that the model correctly identifies 78% of all fraudulent transactions.
- **F1-Score (Class 0 = 1.00, Class 1 = 0.84)**: The F1-score is the harmonic mean of precision and recall. For fraudulent transactions, the F1-score is 0.84, suggesting a balance between precision and recall for this class.
- **Support**: The support for each class indicates the number of actual occurrences in the dataset (56861 for legitimate transactions and 101 for fraudulent transactions).

The report also includes macro and weighted averages for precision, recall, and the F1-score:

Macro Avg :The average precision, recall, and F1-score without taking class imbalance into account.

Weighted Avg :The average precision, recall, and F1-score, weighted by the support of each class, which accounts for class imbalance.

The report also includes macro and weighted averages for precision, recall, and the F1-score:

- **Macro Avg**: The average precision, recall, and F1-score without taking class imbalance into account.
- **Weighted Avg**: The average precision, recall, and F1-score, weighted by the support of each class, which accounts for class imbalance.

This detailed output shows that the model performs exceptionally well in identifying both legitimate and fraudulent transactions, with high accuracy and balanced precision and recall for fraudulent transactions, despite the inherent class imbalance often present in fraud detection datasets.

Here are the results from the application of Random Forest via python code:

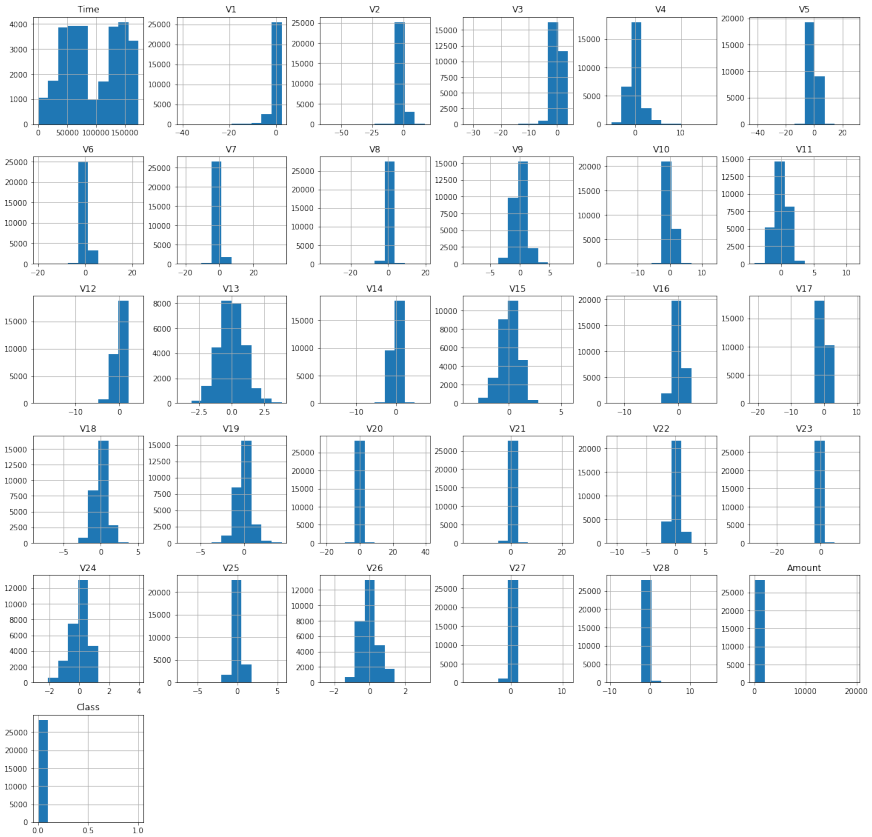

Figure 3: Results

Summary Statistics

Summary statistics for the dataset are as follows:

	Time	V1	V2	V3
count	28481.000000	28481.000000	28481.000000	28481.000000
mean	94705.035216	-0.001143	-0.018290	0.000795
std	47584.727034	1.994661	1.709050	1.522313
min	0.000000	-56.407510	-72.715728	-48.325589
25%	54022.000000	-0.920373	-0.598550	-0.890365
50%	84692.000000	0.018109	0.065486	0.179846
75%	139320.000000	1.315642	0.803724	1.027196
max	172792.000000	2.454930	22.057729	9.382558

2. Support Vector Machines (SVM)

Support Vector Machines (SVMs) represent a group of supervised learning techniques applicable to classification, regression, and outlier detection tasks. The goal of the SVM algorithm is to identify a hyperplane within an N-dimensional space (where N is the number of features) that clearly separates the data points. (Cortes, Vapnik, 1995)

2.1 Linear SVM

The goal of the linear SVM algorithm is to find the optimal separating hyperplane which maximizes the margin between two classes. The decision function is defined as:

$$f(x) = \mathbf{w}^T\mathbf{x} + b \tag{4}$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the feature vector, and b is the bias. The optimal hyperplane can be found by solving the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \tag{5}$$

subject to the constraint that all data points are correctly classified, i.e.,

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall i \tag{6}$$

where y_i is the label of the i-th data point.

2.2 Support Vectors

Support vectors are the data points that are closest to the hyperplane. These points are critical in defining the hyperplane because the hyperplane is positioned based on these points. The margin is defined by the distance between the hyperplane and the nearest data point from either class and is calculated as $\frac{2}{||\mathbf{w}||}$.

2.3 Kernel SVM

In the case of this study, the data is not linearly separable in the original feature space. Kernel SVMs allow the algorithm to operate in a transformed feature space where a linear separation is possible without explicitly computing the transformation. The kernel function can be defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) \tag{7}$$

where ϕ is the transformation function to the higher-dimensional space. Common kernel functions include:

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$

- Polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d$, where γ, r, and d are parameters.
- Radial Basis Function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2)$, where γ is a parameter.
- Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$

By using kernel functions, SVMs can efficiently perform a non-linear classification, leveraging the kernel trick to avoid the explicit computation of the transformation to high-dimensional space. (Hasan, Xu, Kabir, 2016)

2.4 Radial Basis Function (RBF) Kernel

The RBF kernel, or Gaussian kernel, is defined as:

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$

where $||x - x'||$ is the Euclidean distance between the vectors, and σ is a parameter that determines the spread of the kernel. (Cao, Naito, Ninomiya, 2008)

The RBF kernel's effectiveness in non-Gaussian contexts lies in its non-linear transformation, capable of handling complex relationships within the data. The flexibility and locality of the RBF kernel make it suited for the skewed distributions typical in fraud detection scenarios.

In non-Gaussian distributions, the RBF kernel allows for capturing the intricate structures by implicitly mapping the input space to a high-dimensional feature space where classes are more likely to be linearly separable.

The parameters C and γ (where $\gamma = 1/(2\sigma^2)$) control the trade-off between smooth decision boundary and classifying training points correctly. A grid search with cross-validation is commonly used to find the optimal values:

$$C \in \{10^{-3}, 10^{-2}, \ldots, 10^3\}, \quad \gamma \in \{10^{-3}, 10^{-2}, \ldots, 10^3\}$$

Python implementation results

	precision	**recall**	**f1-score**	**support**
0	1.00	1.00	1.00	56864
1	0.97	0.66	0.79	98
accuracy			1.00	
macro avg	0.98	0.83	0.89	56962
weighted avg	1.00	1.00	1.00	56962

Table 3: Classification Report

Average Precision Score: 0.6440456753513998

3. Influence of Minkowski Distance Parameter p

3.1 Introduction

The analysis of transaction data, particularly in distinguishing between fraudulent and legitimate transactions, is crucial for developing effective fraud detection systems. The geometry of transaction data refers to the shape, distribution, and arrangement of data points in a high-dimensional space. This section discusses how the geometric properties of fraudulent and legitimate transaction data might differ and explores the effectiveness of varying the parameter p in the Minkowski distance formula for capturing these differences more adeptly than using a fixed p, such as $p = 2$ for Euclidean distance.

3.2 Differences in Transaction Data

The features of fraudulent transactions usually allow bank security to set them apart from legitimate transactions. These differences include density, clustering, and distribution:

1. **Density**: Fraudulent transactions might congest in the same areas in an n-dimensional space. This might be due to behavioral similarity in fraudsters actions, which lead to dense pockets of fraudulent data points within the broader transaction space.

2. **Clustering**: In a real world environment, the datasets are highly imbalanced with legitimate transactions being a dominant class. Thus, it is expected that there will be more diffuse clusters reflecting the wide variety of genuine transaction behaviors, whereas fraudulent clusters may be smaller and more distinct.

3.3 Impact of Varying p in Minkowski Distance

The Minkowski distance is a metric used for measuring the distance between two points in a normed vector space and is defined as:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Where X and Y are two points in the space, and p is the order parameter of the distance. Varying p can significantly affect how the distances between points are calculated:

1. $p = 1$ **(Manhattan Distance)**: This distance measures the sum of the absolute differences of their Cartesian coordinates. It is more sensitive to anomalies that manifest as large deviations in one or a small number of dimensions, typical of categorical and discrete numerical data. (Thant, Aye, Mandalay, 2020)

2. $p = 2$ **(Euclidean Distance)**: This is the most common form used, measuring the shortest path between points. It treats all dimensions equally and is effective in capturing the overall distance but can be less sensitive to outliers in individual dimensions.

3. **Higher values of** p: Increasing p tends to highlight the largest single element difference between the two points, thus becoming more sensitive to outliers in any single dimension.

4. **Varying** p: By adapting p, one can fine-tune the sensitivity of the distance metric to different types of geometric distributions. For instance, a smaller p could better cluster dense fraud pockets with small deviations, while a larger p might better isolate outliers.

3.4 Implementation of custom Minkowski Distance in SVM

Creating a custom kernel for use with Support Vector Machines (SVM) that incorporates the Minkowski distance can enhance the handling of datasets where standard kernels may not capture the intricacies effectively. Below is a suggestion for a custom kernel based on the Minkowski distance:

3.4.1 Custom Minkowski Kernel Formulation

To transform the Minkowski distance into a kernel function, we can use a similarity function that decreases with increasing distance, such as an exponential function. The kernel is defined as:

$$K(X,Y) = \exp\left(-\frac{D(X,Y)^q}{\sigma}\right)$$

where, D(X, Y) is the Minkowski distance, q is a positive scaling factor that adjusts the sensitivity of the kernel to changes in the distance D(X, Y), and *sigma* is a free parameter controlling the width of the kernel, similar to the *gamma* parameter in the RBF kernel.

3.4.2 Parameters

1. p **in Minkowski Distance**: Adjusting p allows the kernel to be more sensitive to different kinds of deviations in the data.
2. **Scaling Factor** q: Modifies how the distance influences the kernel function. Values greater than 1 increase sensitivity to larger distances.
3. **Bandwidth** σ: Controls the smoothness of the decision boundary. Smaller values lead to faster decay and potential overfitting, while larger values smooth out the decision boundary.

3.4.3 Implementation in Python

To implement the idea of using the custom kernel with the Minkowski distance, the study will be using python environments with manually defined functions. For instance, here is a snippet of the code that was used:

```
from sklearn.svm import SVC
import numpy as np

def minkowski_kernel(X, Y, p=2, q=1, sigma=1):
    dist = np.power(np.sum(np.abs(X[:, None] - Y[None, :])**p, axis=2), 1/p)
    return np.exp(-np.power(dist, q) / sigma)

clf = SVC(kernel=lambda X, Y: minkowski_kernel(X, Y, p=2, q=1, sigma=1))
clf.fit(X_train, y_train)
```

Since there was no installed function for the new custom kernel, it needed to be made manually. The algorithm included precomputing the gram matrix using the python environment and then using this results for further computations

3.4.4 Results

Accuracy: 0.997 Cross-validation scores: [0.998 0.9985 0.9985 0.9985 0.9985] Average score: 0.9984 Best parameters: 'C': 0.1, 'gamma': 0.01 Best cross-validation score: 0.9984

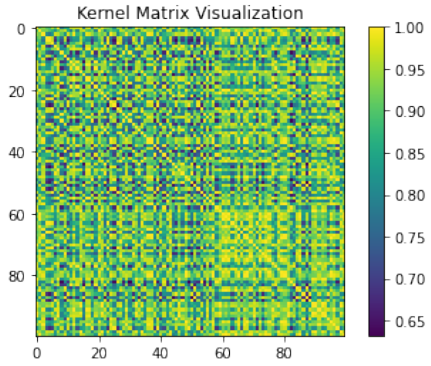

Figure 4: Kernel Matrix Visualization

The figure above is a heatmap representation of a Kernel Matrix or a Gram Matrix. Each cell in the matrix corresponds to the computed kernel between two instances in a dataset. A kernel function measures the similarity or a defined computation over pairs of inputs. The colors that are on a brighter scale on the right represents more similarity.

The figure above represents the non-linearly separable case, since visually it can be seen that there is no line that can separate Class 0 and Class 1.

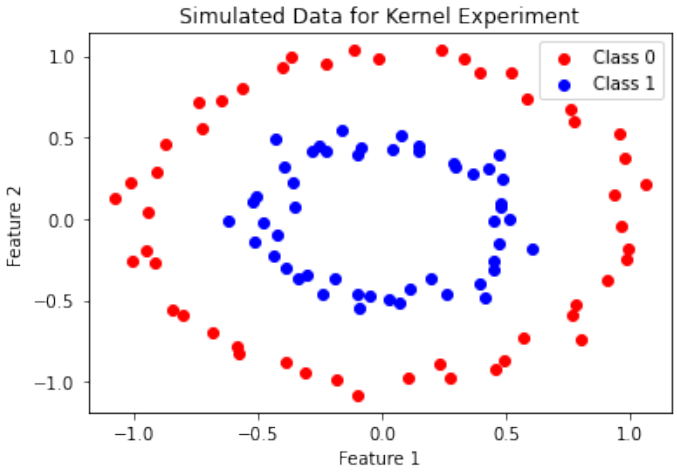

Figure 5: Simulated Data for Custom Kernel Experiment

3.4.5 Conclusion

Implementation of the Minkowski distance in a custom kernel function allow us to choose p, which will enhance the detection capabilities of fraud detection systems. An Euclidean distance, or fixed $p = 2$ in the Minkowski distance, might not provide reliable results when dealing with datasets with non-Gaussian distributions. However, in our case where p is not fixed, the solutions might be more robust, so the fraudulent and legitimate transactions can be identified more accurately. The empirical study provided in this paper shows promising results, but cannot fully guarantee the security of credit card transactions.

4. Limitations

Due to the sensitive nature of the data that is needed to train the model, it is impossible to get the information on the real-time transactions of existing bank customers. Therefore, this study is limited to the artificially made datasets, which does not allow one to be fully confident in the results provided by the models that are being tested. Moreover, in order to train and test the model very large datasets are being used, which requires a lot of computational power. For instance, during the testing period the PC gave the warnings and shut down the processes, so the code did not execute. One of the common mistakes that were shown were: "Unable to allocate 11.3 TiB for an array with shape (227845, 227845, 30) and data type float64."

References

"Credit Card Fraud Detection." Kaggle, 2021. Available: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?resource=download

L. Breiman, J. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees*, 1st ed., Chapman and Hall/CRC, 1984. DOI: https://doi.org/10.1201/9781315139470

K. P. Bennett, "Decision Tree Construction via Linear Programming." 1992. Retrieved from https://minds.wisconsin.edu/bitstream/handle/1793/59564/TR1067.pdf?sequence=

G. Zhang and A. Gionis, "Regularized impurity reduction: accurate decision trees with complexity guarantees." *Data Min Knowl Disc*, vol. 37, pp. 434–475, 2023. DOI: https://doi.org/10.1007/s10618-022-00884-7

T. M. Cover and J. A. Thomas, "Entropy, relative entropy and mutual information," in *Elements of Information Theory*, 1991. Available: http://www.stat.columbia.edu/~liam/teaching/neurostat-fall17/papers/EM/Cover\&Thomas-Ch2.pdf

D. M. Farid, M. Z. Rahman, and C. M. Rahman, "An ensemble approach to classifier construction based on bootstrap aggregation." *International Journal of Computer Applications*, 2011. Available: https://citeseerx.ist.psu.edu/document?repid=rep1\&type=pdf\&doi=11b9f20b8ad759225c9151defbec72b4caa7166c

C. Cortes and V. Vapnik, "Support-vector networks." *Mach Learn*, vol. 20, pp. 273–297, 1995. DOI: https://doi.org/10.1007/BF00994018

H. Cao, T. Naito, and Y. Ninomiya, "Approximate RBF Kernel SVM and Its Applications in Pedestrian Classification." The 1st International Workshop on Machine Learning for Vision-based Motion Analysis-MLVMA'08, Oct 2008, Marseille, France. INRIA, inria-00325810.

M. A. M. Hasan, S. Xu, and M. M. J. Kabir, "Performance evaluation of different kernels for support vector machine used in intrusion detection system." *International Journal of Network Security & Its Applications*. Available: https://www.academia.edu/download/60155294/8616cnc0420190729-75198-qx4ykx.pdf

A. A. Thant and A. S. M. Aye, "Euclidean, Manhattan and Minkowski Distance Methods For Clustering Algorithms," *International Journal of Scientific Research in Science, Engineering and Technology* (IJSRSET), vol. 7, no. 3, pp. 553-559, May-June 2020. DOI: https://doi.org/10.32628/IJSRSET2073118

3 Apendix

Code for the Random Forest:

```
import pandas as pd
file_path = 'creditcard.csv'
data = pd.read_csv(file_path)
data.head(), data.shape
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
X = data.drop(['Class'], axis=1) # Features
y = data['Class'] # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

random_forest = RandomForestClassifier(n_estimators=100, random_state=0)

random_forest.fit(X_train, y_train)

y_pred = random_forest.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

accuracy, conf_matrix, report
feature_importances = pd.Series(random_forest.feature_importances_, index=X.columns).sort_values(ascending=False)

plt.figure(figsize=(10,8))
feature_importances.plot(kind='bar')
plt.title('Feature Importance')
plt.show()

conf_mat = confusion_matrix(y_test,y_pred)
sns.heatmap(conf_mat, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

Code for the Isolation Forest

```
# -*- coding: utf-8 -*-
!pip install numpy pandas matplotlib seaborn scipy scikit-learn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sys
import scipy

# load the dataset using pandas
data = pd.read_csv('creditcard.csv')

# dataset exploring
print(data.columns)

# Print the shape of the data
data = data.sample(frac=0.1, random_state = 1)
print(data.shape)
```

```
print(data.describe())

# V1 - V28 are the results of a PCA Dimensionality reduction to protect user identities and sensitive features

# Plot histograms of each parameter
data.hist(figsize = (20, 20))
plt.show()

# Determine number of fraud cases in dataset

Fraud = data[data['Class'] == 1]
Valid = data[data['Class'] == 0]

outlier_fraction = len(Fraud)/float(len(Valid))
print(outlier_fraction)

print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))

# Correlation matrix
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))

sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()

# Get all the columns from the dataFrame
columns = data.columns.tolist()

# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]

# Store the variable we'll be predicting on
target = "Class"

X = data[columns]
Y = data[target]

# Print shapes
print(X.shape)
print(Y.shape)

from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

# define random states
state = 1

# define outlier detection tools to be compared
classifiers = {
    "Isolation Forest": IsolationForest(max_samples=len(X),
                                        contamination=outlier_fraction,
                                        random_state=state),
    "Local Outlier Factor": LocalOutlierFactor(
        n_neighbors=20,
        contamination=outlier_fraction)}

plt.figure(figsize=(9, 7))
n_outliers = len(Fraud)


for i, (clf_name, clf) in enumerate(classifiers.items()):

    # fit the data and tag outliers
```

```
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_pred = clf.negative_outlier_factor_
    else:
        clf.fit(X)
        scores_pred = clf.decision_function(X)
        y_pred = clf.predict(X)

    # Reshape the prediction values to 0 for valid, 1 for fraud.
    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1

    n_errors = (y_pred != Y).sum()

    # Run classification metrics
    print('{}: {}'.format(clf_name, n_errors))
    print(accuracy_score(Y, y_pred))
    print(classification_report(Y, y_pred))
```

Code for Support Vector Machine:

```
import pandas as pd

# Load the dataset
file_path = 'creditcard.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset to understand its structure
data.head(), data.shape

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline

# Load the dataset
file_path = 'creditcard.csv'
data = pd.read_csv(file_path)
df = pd.read_csv(file_path)

# Prepare the data
X = df.drop('Class', axis=1)  # Features
y = df['Class']               # Target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

# Create and train the SVM model with RBF kernel
model = make_pipeline(StandardScaler(), SVC(kernel='rbf', C=1, gamma='scale'))
model.fit(X_train, y_train)

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

# Define the pipeline
pipeline = make_pipeline(StandardScaler(), SVC())

# Assuming X_train and y_train are already defined and contain your training data and labels:
pipeline.fit(X_train, y_train)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

pipeline.fit(X_train, y_train)

from sklearn.metrics import classification_report, average_precision_score
```

```
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))
print("Average Precision Score:", average_precision_score(y_test, y_pred))
```

Code for the Custom Minkowski Kernel

```
pip install pandas scikit-learn scipy
import numpy as np
import pandas as pd
from sklearn import svm
from scipy.spatial.distance import cdist

file_path = 'creditcard.csv'
data = pd.read_csv(file_path)

subset_size = 10000
subset = data.sample(n=subset_size, random_state=42)

features = subset.drop('Class', axis=1).values
labels = subset['Class'].values

print("Features shape:", features.shape)

def minkowski_rbf_kernel(X, Y=None, p=1, gamma=None):
    if Y is None:
        Y = X
    if gamma is None:
        gamma = 1 / X.shape[1]
    M = cdist(X, Y, metric='minkowski', p=p)
    K = np.exp(-gamma * (M ** p))
    return K

p_value = 1
gamma_value = 1 / features.shape[1]
K = minkowski_rbf_kernel(features, p=p_value, gamma=gamma_value)

clf = svm.SVC(kernel='precomputed')
clf.fit(K, labels)
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, average_precision_score

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

K_train = minkowski_rbf_kernel(X_train, p=p_value, gamma=gamma_value)

clf.fit(K_train, y_train)

K_test = minkowski_rbf_kernel(X_test, Y=X_train, p=p_value, gamma=gamma_value)

predictions = clf.predict(K_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
from sklearn.model_selection import cross_val_score

K = minkowski_rbf_kernel(features, p=p_value, gamma=gamma_value)

scores = cross_val_score(clf, K, labels, cv=5)
print("Cross-validation scores:", scores)
print("Average score:", np.mean(scores))
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.01, 0.1, 1, 10]
}

clf = svm.SVC(kernel='precomputed')
```

```
K = minkowski_rbf_kernel(features, p=p_value, gamma=None)

grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(K, labels)
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)

from scipy.spatial.distance import cdist

def minkowski_rbf_kernel(X, Y=None, p=2, gamma=0.1):
    if Y is None:
        Y = X
    M = cdist(X, Y, metric='minkowski', p=p)
    K = np.exp(-gamma * (M ** p))
    return K
    K = minkowski_rbf_kernel(X, p=2, gamma=0.1)

plt.imshow(K, interpolation='none')
plt.title('Kernel Matrix Visualization')
plt.colorbar()
plt.show()
import numpy as np
import pandas as pd
from sklearn import svm
from scipy.spatial.distance import cdist
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split

# Load and prepare the dataset
file_path = 'creditcard.csv'
data = pd.read_csv(file_path)

subset_size = 10000
subset = data.sample(n=subset_size, random_state=42)

features = subset.drop('Class', axis=1).values
labels = subset['Class'].values

# Define Minkowski RBF Kernel function
def minkowski_rbf_kernel(X, Y=None, p=2, gamma=None):
    if Y is None:
        Y = X
    if gamma is None:
        gamma = 1 / X.shape[1]  # Default gamma
    M = cdist(X, Y, metric='minkowski', p=p)
    K = np.exp(-gamma * (M ** p))
    return K

# Set kernel parameters
p_value = 2
gamma_value = 1 / features.shape[1]
K = minkowski_rbf_kernel(features, p=p_value, gamma=gamma_value)

# Initialize and train the SVM classifier
clf = svm.SVC(kernel='precomputed')
clf.fit(K, labels)  # Note: fitting on the entire dataset is not typically advisable

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

# Compute kernel matrices for training and testing sets
K_train = minkowski_rbf_kernel(X_train, p=p_value, gamma=gamma_value)
K_test = minkowski_rbf_kernel(X_test, Y=X_train, p=p_value, gamma=gamma_value)

# Fit model and make predictions
clf.fit(K_train, y_train)
predictions = clf.predict(K_test)

# Calculate and print classification report
report = classification_report(y_test, predictions)
print("Classification Report:")
print(report)
```

```
# Display accuracy
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

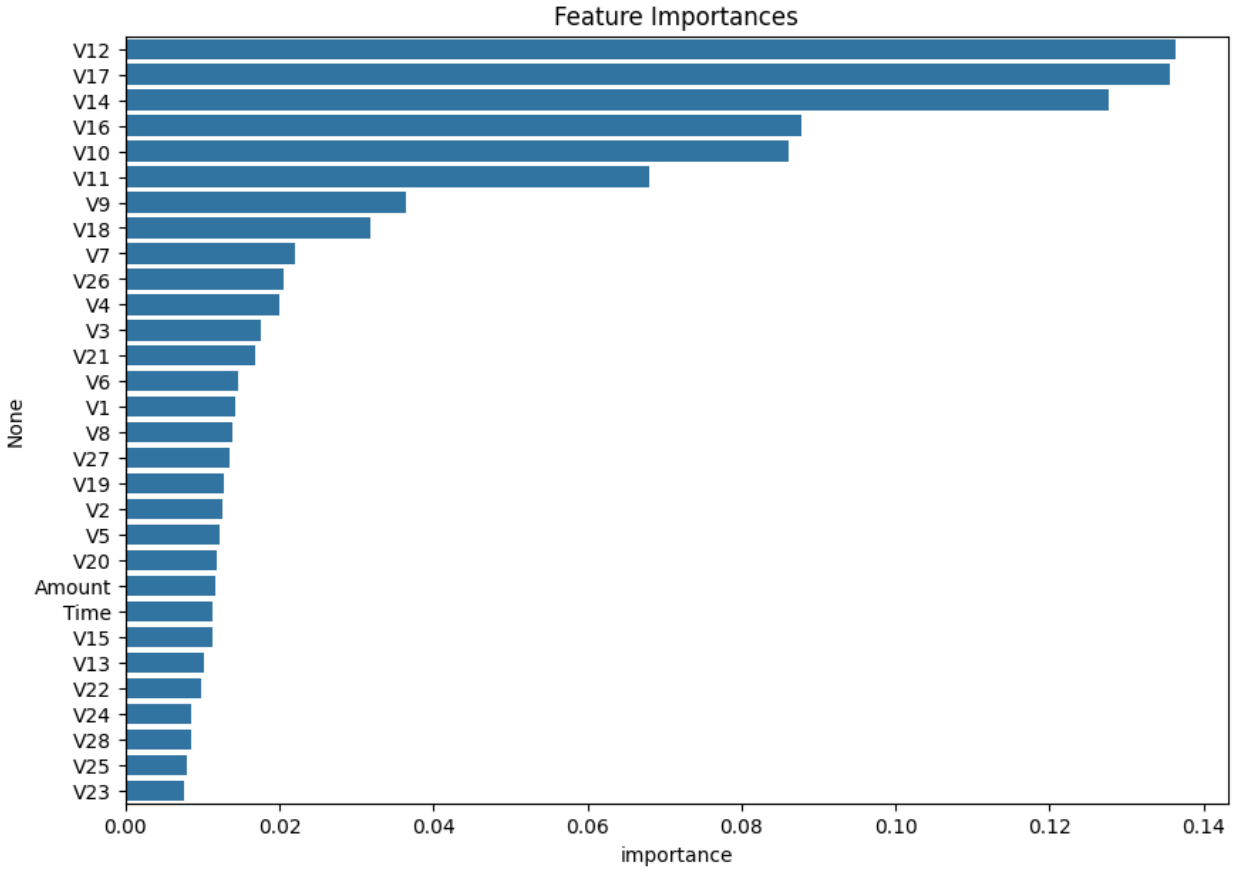

Figure 6: Feature Importances