Politecnico di Torino

Analysing Adversarial Attacks on Tabular Data Classifiers

Trad Gianfranco s323713

01DSMUV Machine Learning for Networking

January 2024

Contents

Chapter 1

Data exploration and preprocessing

1.1 Data Preprocessing

The German Credit Risk dataset consists of both categorical and numerical features. Categorical features are encoded with alphanumeric codes. The 20^{th} feature represents the result, where 1 indicates a positive outcome (eligible for credit), and 2 indicates a negative outcome (not eligible for credit). To conduct data exploration, identifying patterns through descriptive statistics as well as visualizations is essential. First and foremost, a meaningful name to each feature is assigned, specifying the numerical attributes for subsequent explorations.

Upon associating each specific feature with its corresponding name, the resulting dataset appears as follows:

1-10 1000 rows × 21 columns

	existing_account	month_duration	credit_history	purpose	credit_amount	saving_bonds
0	A11	6	A34	A43	1169	A65
1	A12	48	A32	A43	5951	A61
2	A14	12	A34	A46	2096	A61
3	A11	42	A32	A42	7882	A61
4	A11	24	A33	A40	4870	A61
5	A14	36	A32	A46	9055	A65
6	A14	24	A32	A42	2835	A63
7	A12	36	A32	A41	6948	A61
8	A14	12	A32	A43	3059	A64
9	A12	30	A34	A40	5234	A61

Figure 1.1: DF with features assigned

To convert the categorical features into numerical representations, the one-hot encoding encoding was applied. During this process, the label indicating the result was transformed by converting its values from $\{1, 2\}$ to $\{0, 1\}$. Therefore, indicating as 0 positive loans while with 1 negative loans.

Firstly, the dataset was split into train-validation and test set and later normalization was performed. This approach is taken to prevent data leakage between the split of the training-validation (80% of the starting dataset) and the test set (20%). This is known as a best practice method when approaching ML tasks. The test set was immediately normalized, as it will remain untouched until it will be used to test the classifiers, while each train and validation splits will be separately normalized in the cross-validation, in order to train and validate the classifiers without incurring in data leakage. Another crucial aspect is that the initial train-validation and test split is stratified, as the starting dataset is unbalanced. Later on, the cross-validation was also performed in a stratified fashion (StratifiedK-Fold).

The statistics displayed underneath are computed over eligible and non-eligible data points.

	month_duration	credit_amount	installment_rate	resident_since	age	credit_number	people_liability
count	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000
mean	19.207143	2985.457143	2.920000	2.842857	36.224286	1.424286	1.155714
std	11.079564	2401.472278	1.128078	1.108373	11.381145	0.584721	0.362844
min	4.000000	250.000000	1.000000	1.000000	19.000000	1.000000	1.000000
25%	12.000000	1375.500000	2.000000	2.000000	27.000000	1.000000	1.000000
50%	18.000000	2244.000000	3.000000	3.000000	34.000000	1.000000	1.000000
75%	24.000000	3634.750000	4.000000	4.000000	42.250000	2.000000	1.000000
max	60.000000	15857.000000	4.000000	4.000000	75.000000	4.000000	2.000000

Figure 1.2: Statistics computed over eligible data points

	month_duration	credit_amount	installment_rate	resident_since	age	credit_number	people_liability
count	300.000000	300.000000	300.000000	300.000000	300.000000	300.000000	300.000000
mean	24.860000	3938.126667	3.096667	2.850000	33.963333	1.366667	1.153333
std	13.282639	3535.818955	1.088395	1.094605	11.222379	0.559702	0.360911
min	6.000000	433.000000	1.000000	1.000000	19.000000	1.000000	1.000000
25%	12.000000	1352.500000	2.000000	2.000000	25.000000	1.000000	1.000000
50%	24.000000	2574.500000	4.000000	3.000000	31.000000	1.000000	1.000000
75%	36.000000	5141.500000	4.000000	4.000000	40.000000	2.000000	1.000000
max	72.000000	18424.000000	4.000000	4.000000	74.000000	4.000000	2.000000

Figure 1.3: Statistics computed over non-eligible data points

We can observe that the features "month_duration" and "credit_amount" generally have higher values in the non-eligible entries compared to the eligible ones.

1.2 Exploratory Data Analysis

1.2.1 Target Distribution

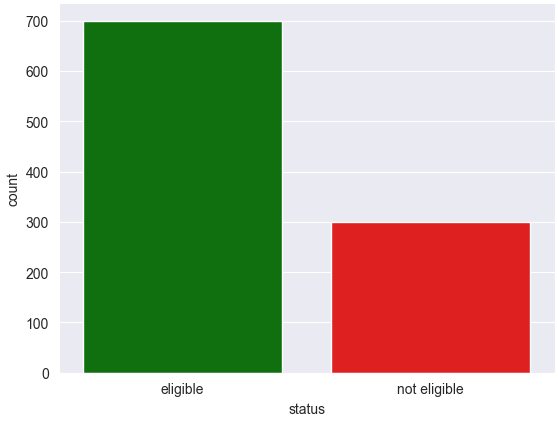


Figure 1.4: Target Distribution

As we can see from the graph above the dataset is unbalanced, in fact, 70% of observations belong to the eligible class while only 30% belong to the not eligible one. To deal with it, as we already mentioned,stratified split on the "result" column and stratified cross-validation will be carried out in the Supervised Learning.

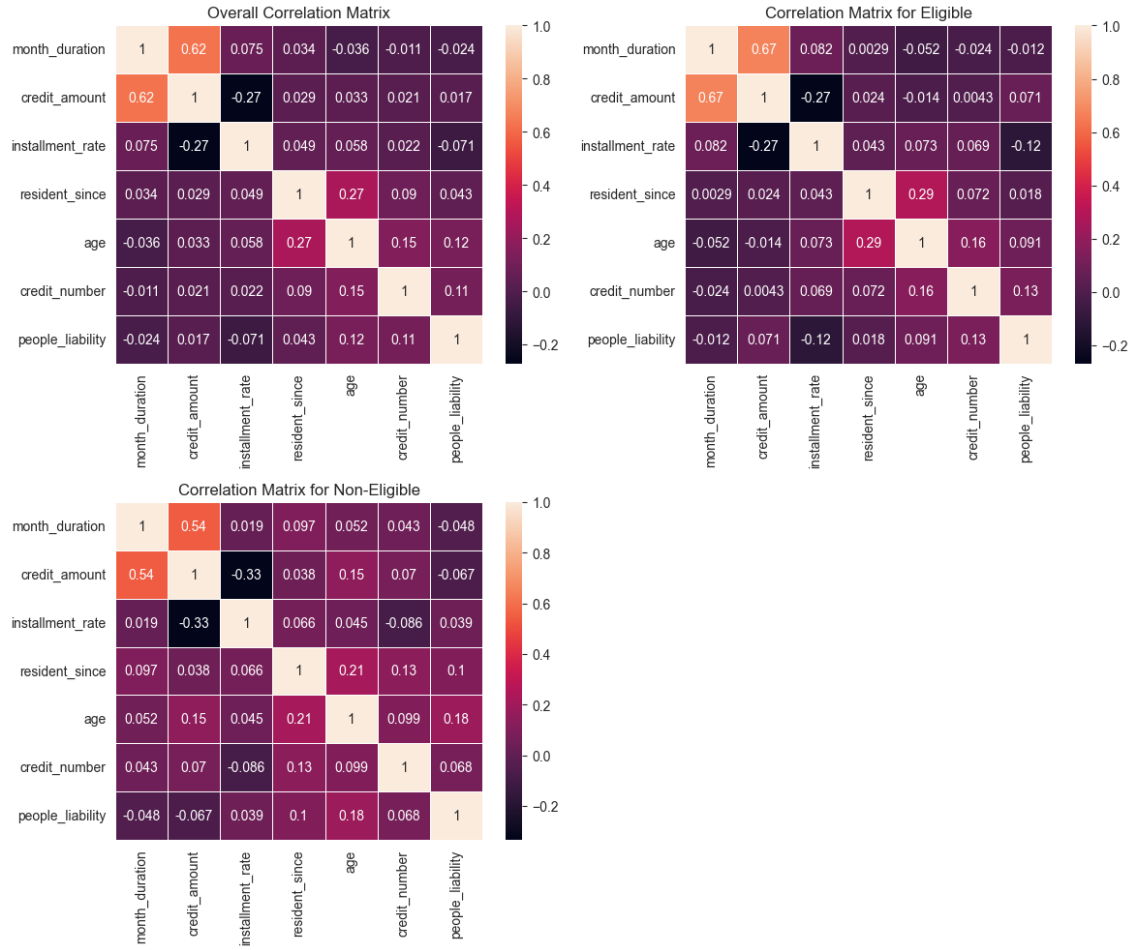


Figure 1.5: Correlation Matrixes

The correlation matrixes above show the correlation between the numerical variables for the whole dataset and for both eligible and non-eligible data points.

Interestingly, the *credit_amount* and *month_duration* features present moderately positive correlation.

In the following pages a subset of the performed data analysis with graphical plots is presented.

1.2.2 Age Distribution

Figure 1.6: Age Distribution Plots

The first plot displays the overall age distribution, while the second and third plots illustrate the distribution of Bad Credit and Good Credit across different ages. In the final plot, through **feature engineering** all dataset entries were categorized into four groups (Student, Young, Adult, and Senior) based on their age, in order to investigate whether age groups may influence credit outcomes.

The ratios suggest that in the Student group, aged from 18 to 25, encountering bad credit is more likely compared to the general 70/30 split of the dataset. Conversely, in the Senior category (age greater than or equal to 60), a positive outcome is more probable. Additionally, the ratio appears to slightly increase with the age group. Hence, a question arises:

Could it be that older individuals, with potentially more stable jobs in comparison to students, are more likely to be good creditors?

To address this question, the act of exploring whether age is influenced by other variables such as existing_account, saving_bonds, job, and housing_status is performed.

1.2.3 Loan Duration

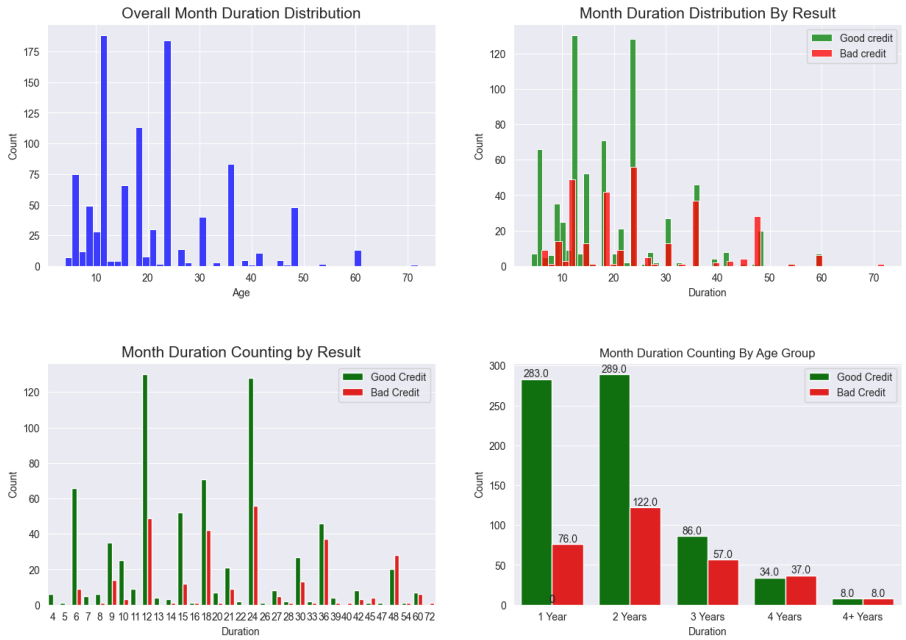


Figure 1.7: Loan Duration

In the graphs shown above it's interesting to notice how the longer the loan duration is the more likely it could be a non-eligible loan. This prolonged exposure increases the risk for both the borrower and the lender. Economic conditions, personal circumstances, and creditworthiness may change over time, potentially impacting the borrower's ability to meet the financial obligations.

1.2.4 Credit Amount

The following plots assess the **Credit Amount** features which states the amount of money borrowed by each applicant.

Figure 1.8: Enter Caption

The last plot shows how in the 4^{th} quartile the ratio between good and bad loans drops to 58%

compared to the starting 70/30 split. Therefore, we can denote how the higher the credit amount is the more likely it could present a bad loan.

1.2.5 Saving Bonds

This section delves into considering the *Saving Bonds* variable, which describes the amount of savings of each customers. Additionally, also a study to understand if the *Saving Bonds* and *Age* features are modulated between each other is carried out. The same goes for the *Saving Bonds* and *Credit Amount* variables.

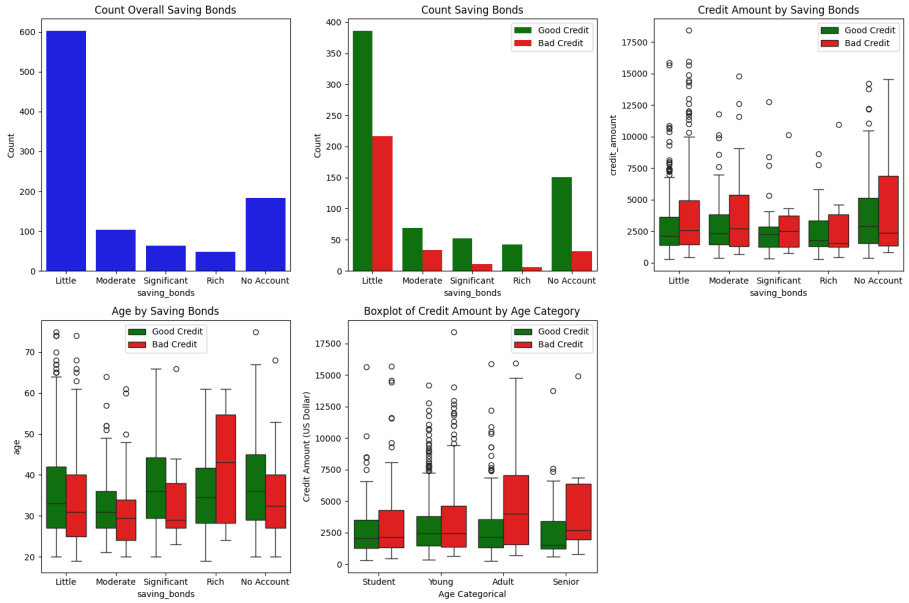


Figure 1.9: Count Savings

We observe that good credits tend to exhibit a similar distribution of credit amounts across all age classes, whereas bad credits show a distinct distribution in the Adult and Senior classes compared to the Student and Young classes. This observation suggests that age might be a significant factor in determining creditworthiness. Consequently, financial institutions might need to customize their risk assessment and lending strategies based on the age of the applicants.

1.2.6 Job

This section considers if the information regarding the job category of each applicant carries meaningful insights on the credit risk. Additionally, studies regarding if the variables *Job* and *Age* are modulated between each other are carried out. Same goes for the *Job* and *Credit Amount* variables.

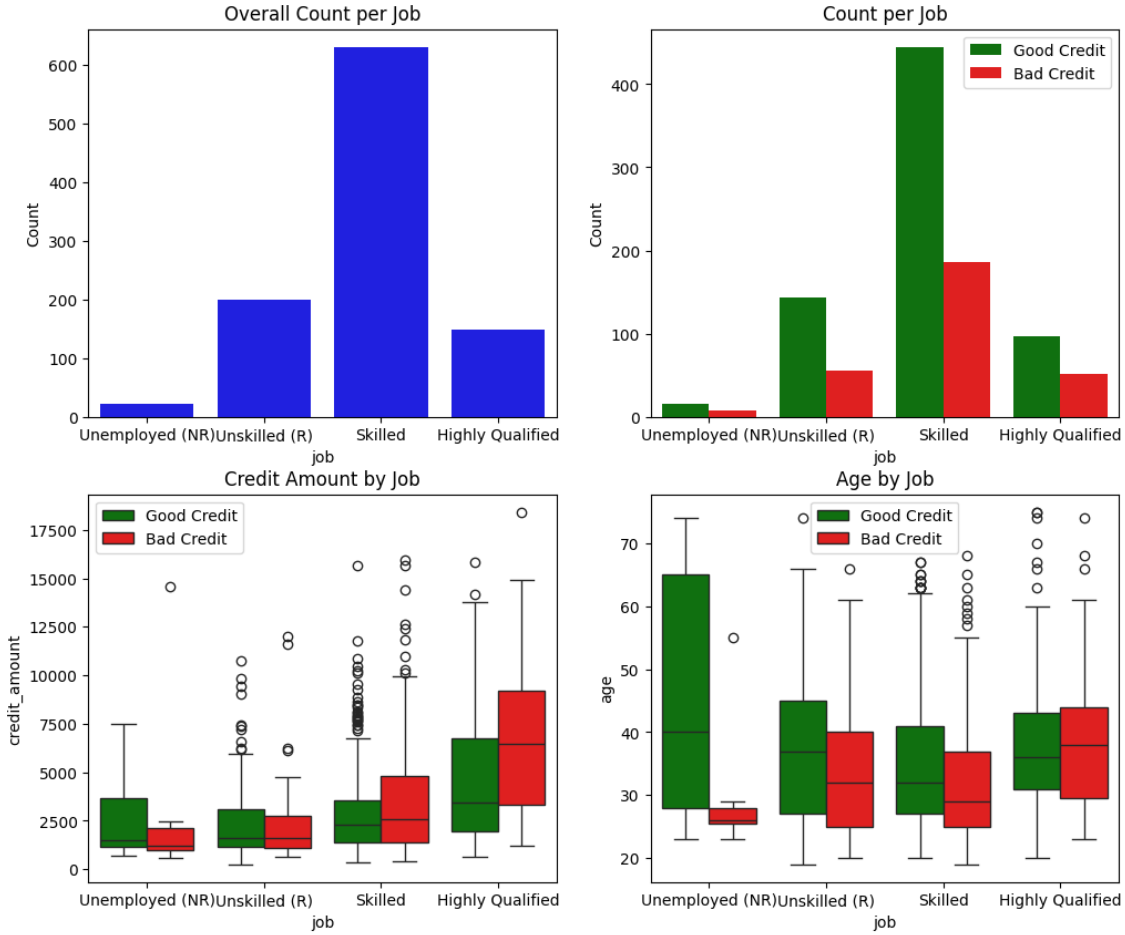


Figure 1.10: Job

First of all, we can note how in the unemployed category the ratio between positive and negative loans is smaller than the 70/30 general split. Whereas, both skilled, unskilled and highly qualified workers preserve the 70/30 ratio, if not even exceeding it as in the case of the Skilled category.

1.2.7 Employment Status

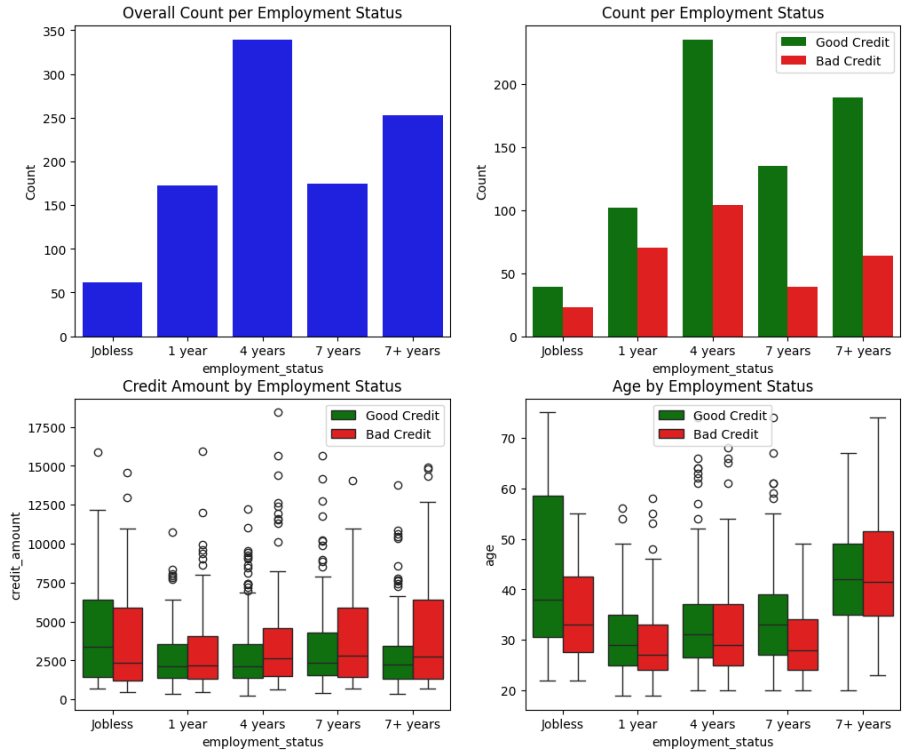


Figure 1.11: Employment Status

From these graphs, it is evident that the bank tends to extend credit to individuals with longer employment duration. The notable distinction is apparent from the 4-year range on, as opposed to those who are jobless or have been working for just one year.

1.2.8 Purpose

This section showcases the different purposes behind credit requests and their association with being good credits or bad credits.

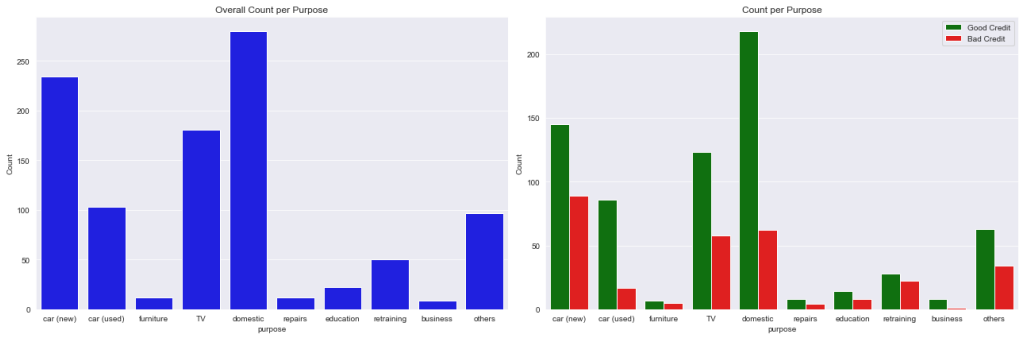


Figure 1.12: Purpose

The majority of loans are made for domestic purposes, but there is also a large number of loans for TVs and cars. The latter purpose also has the highest rate of bad credits.

1.2.9 Housing

This section demonstrates how different types of housing can influence the likelihood of being a good borrower or representing a bad loan.

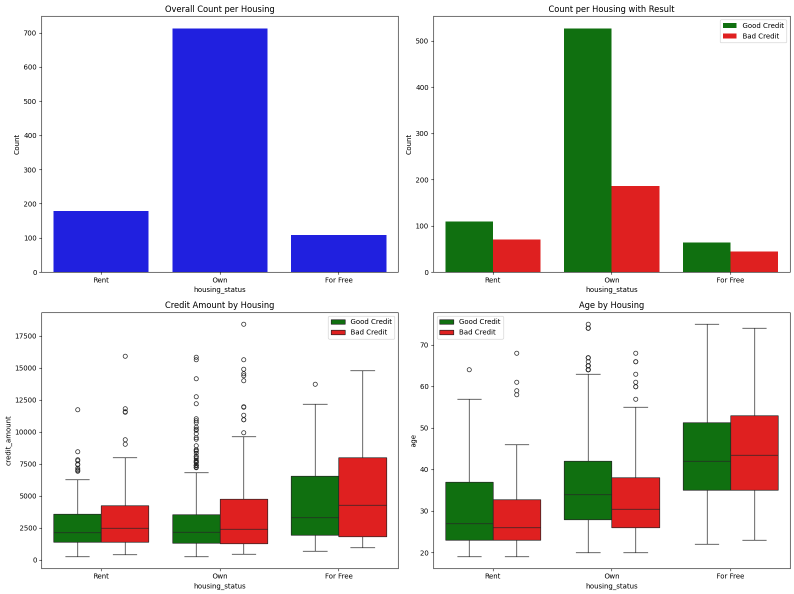


Figure 1.13: Purpose

It is evident that the majority of people applying for credit owns a house, and it is also clear how owning a house positively influences having good credit.

Conclusion

The exploratory data analysis has revealed key insights about the loan risk classification. Analyzing almost every feature in conjunction with age-based categorization brought to realize that the most influential features in determining loan risk classification include the month duration of the loan, the credit amount and the amount of saving bonds. But also, the purpose of the loan, the information regarding the employment of each borrower and the housing status.
As the analysis proceeds, these identified features will be very useful in developing and understanding the patterns and factors influencing loan risk classification.

Chapter 2

Unsupervised exploration and clustering

The second phase involved the dimensionality reduction of the dataset and the effective utilization of unsupervised clustering algorithms to group different data points based on similar characteristics. Two dimensionality reduction techniques are used: t-Distributed Stochastic Neighbor Embedding (t-SNE) and Principal Component Analysis (PCA). The objective is to identify the optimal hyperparameters for clustering, evaluate the quality of clusters, and characterize them based on the most common features.

2.1 Dimensionality reduction for data visualization

Dimensionality reduction is a critical step in data analysis to visualize and process high-dimensional data effectively.

2.1.1 Principal Component Analysis (PCA)

As an initial step, to correctly performing PCA, normalization was performed on the one-hot encoded dataset. Subsequently, labels were removed from the normalized dataset, since they are unnecessary for PCA and unsupervised learning objectives. To perform Principal Component Analysis (PCA), the cumulative explained variance was computed and a plot was generated to determine the optimal number of principal components (PCs) that retain 90% of the variance, which was chosen as the threshold. In this way it is possible to obtain a good balance between dimensionality reduction and information preservation, ensuring the effectiveness of data representation.

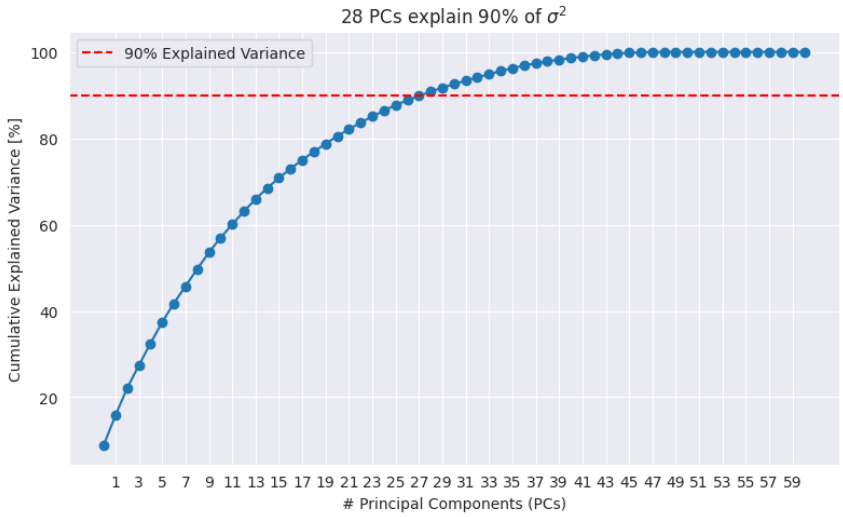


Figure 2.1: 90% Explained Variance

In this case, the one-hot-encoded dataset initially had more than 60 features. By applying PCA, we reduced it to 28, aiming to eliminate all features that were dependent and correlated with each other. The resulting number of components is then used to perform dimensionality reduction, and the reduced dataset is visualized.

2.1.2 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE, or t-Distributed Stochastic Neighbor Embedding, is a dimensionality reduction technique commonly used for visualizing high-dimensional data in lower-dimensional spaces, often 2D or 3D. The primary purpose of t-SNE is to capture the underlying structure of complex datasets, emphasizing the preservation of local relationships between data points. In the implementation provided by scikit-learn *sklearn.manifold.TSNE*, there are several important hyperparameters that require careful consideration. Particular focus was put on adjusting the perplexity, the learning rate, and the number of iterations to apply. The tuned parameters that best represents two clusters are the following ones:

```
model = TSNE(n_components=2, learning_rate=100, perplexity=45, n_iter
    =2000, metric="cityblock", random_state=15, n_jobs=-1)
```

These parameters have been chosen to appropriately represent the clusters present in the dataset. The resulting visualization is presented.

Figure 2.2: Final t-SNE plot

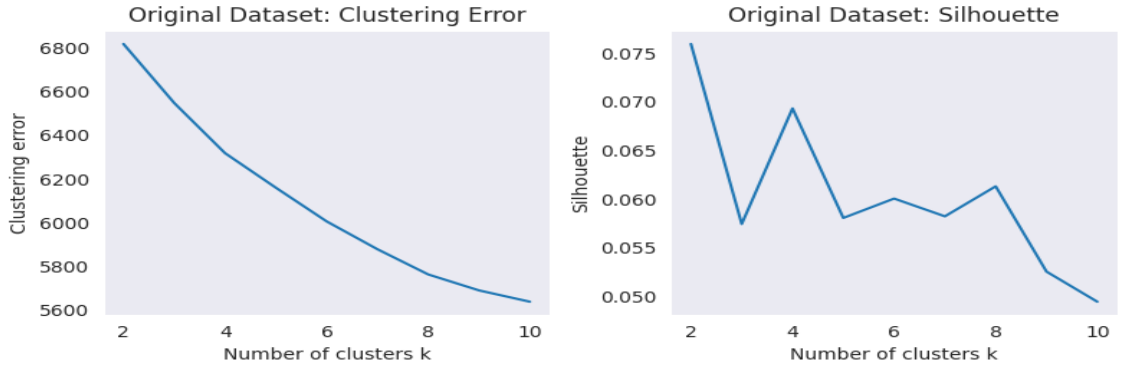
2.2 Unsupervised Data Analysis

After reducing the dimensionality of the dataset, Unsupervised Analysis was performed by implementing two distinct clustering algorithms: K-Means for hard clustering and GMM (Gaussian Mixture Model) for soft clustering.

2.2.1 Hard-Clustering: K-Means

The algorithm initiates by selecting 'k' initial centroids, where 'k' denotes the number of clusters. This parameter is important, and to achieve optimal results silhouette and inertia were selected as

metrics. The silhouette score gauges how well-separated the clusters are, while the clustering error (inertia) indicates the tightness of the clusters. The algorithm was applied to both the original dataset and the one reduced through the PCA technique. Clustering was executed on the original dataset with 'k' ranging from 2 to 10, and for the PCA-reduced dataset, clustering was performed for 'k' ranging from 2 to 10.



(a) K-Means results on the original dataset

(b) K-Means results on the PCA reduced dataset

Figure 2.3: Comparison of K-Means results

In order to evaluate the clustering quality and describe the clusters based on the predominant feature, when dealing with only 2 clusters, insights on the top 3 features with highest distance from each centroid of the 2 clusters are provided. On this basis, the top 3 features per distance were computed for both k-Means and GMM clusters. This analysis is specifically meaningful, aiming to discern the key features that play a significant role in differentiating between the two clusters.

The best performances are obtained with k equal to two. The results obtained on the original dataset are as follows:

- **k-Means with 2 clusters**

  Size of each cluster: [596, 404]

  k_means clustering error: 6816.85

  Silhouette: 0.08

  RI: 0.5

  Adjusted Rand Index (ARI): 0.0

  Top 3 features with the highest distance between the 2 centroids: ['job_A174' 'telephone_A191' 'telephone_A192']

While in the PCA-reduced dataset, here are the results:

- **k-Means with 2 clusters**

  Size of each cluster: [596, 404]

  k_means clustering error: 6067.13

  Silhouette: 0.08

  RI: 0.5

  Adjusted Rand Index (ARI): 0.0

  Top 3 features with the highest distance between the 2 centroids: ['Component 3' 'Component 2' 'Component 1']

In this initial phase, however, the results obtained were not the best. The K-Means algorithm can be influenced by the variable with the largest scale. Specifically, the Euclidean distance, commonly used in K-Means, is strongly influenced by the scales of variables. Therefore, the decision of using different normalization techniques than those applied during the preprocessing step was carried out. In particular, the second normalization technique scales input vectors individually to unit norm (vector length) in order to achieve higher metrics performance. The results obtained with the different normalization are shown below:

Figure 2.4: K-Means results on normalized dataset using Sklearn

The results obtained on the original dataset are as follows:

- k-Means with 2 clusters
  - Size of each cluster: [851, 149]
  - Clustering error (inertia): 0.17
  - Silhouette: 0.65

– Rand Index (RI): 0.53

– Adjusted Rand Index (ARI): -0.02

– Top 3 features with the highest distance between the 2 centroids: ['installment_rate' 'month_duration' 'age']

The results obtained with the second normalization technique show significantly better outcomes. Subsequently, K-Means was applied only to the numerical features of our dataset to which the first normalization method had been applied.

Figure 2.5: K-Means results on numerical feature dataset

Same operation is performed on PCA reduced dataset (only numerical features).

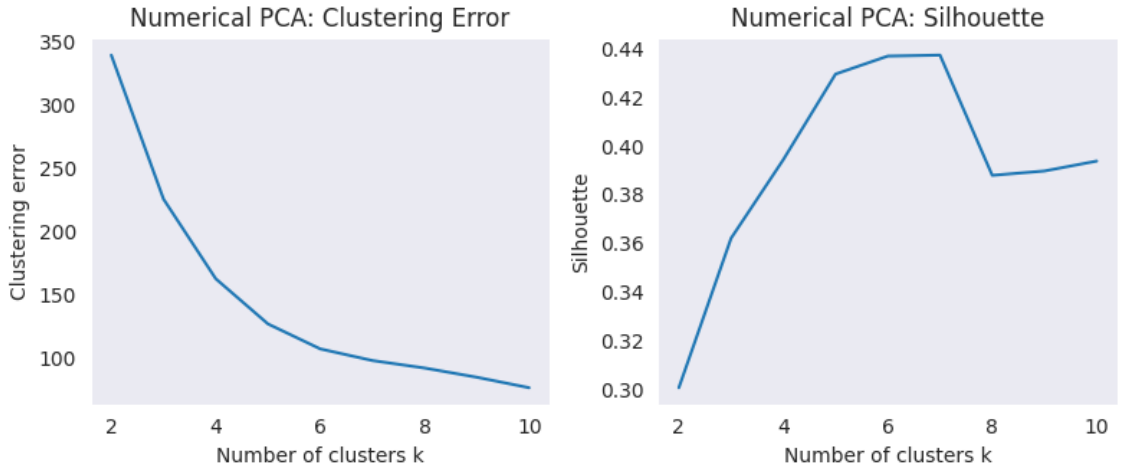


Figure 2.6: K-Means results on numerical feature dataset reduced with PCA

After this series of analyses performed on the K-Means algorithm, it is possible to observe how the proper pre-processing of the dataset significantly influences the clustering. In particular, the obtained results indicate:

- Silhouette Score: this metric measures how similar an object in a cluster is to the other objects in the same cluster compared to objects in the nearest cluster (immediately succeeding cluster). It varies from -1 to 1, where values close to 1 indicate that the object fits well into its cluster and poorly into neighboring clusters. The results obtained, especially with different normalization, are considered adequate.
- ARI (Adjusted Rand Index): this metric measures the similarity between two clusterings, taking into account all pairs of samples and evaluating how well the sample assignments to clusters agree. ARI ranges from -1 to 1, where 1 indicates a perfect match between

clusterings. In this case, the obtained result is slightly below 0, indicating less than perfect agreement between clusters.

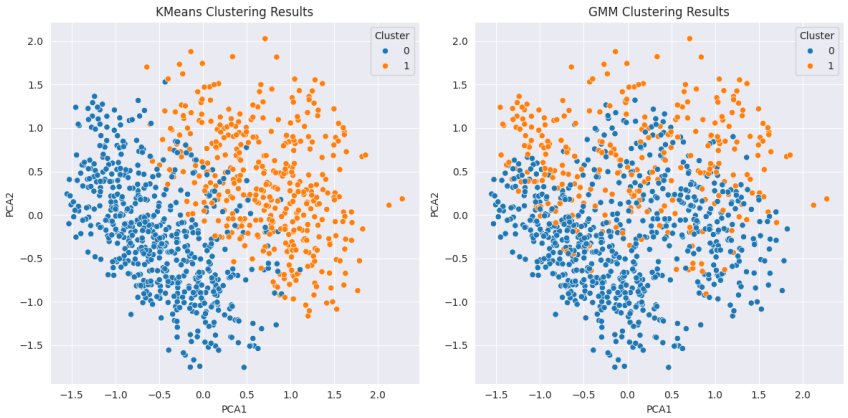
- RI (Rand Index): this metric assesses the similarity between two clusterings by considering pairs of samples and whether they are assigned to the same or different clusters. RI varies from 0 to 1, where 1 indicates a perfect match between clusterings, and 0 indicates a lack of agreement between clusterings. In our case, the obtained results are considered adequate, being positive and above 0.5.

2.2.2 Soft Clustering: GMM

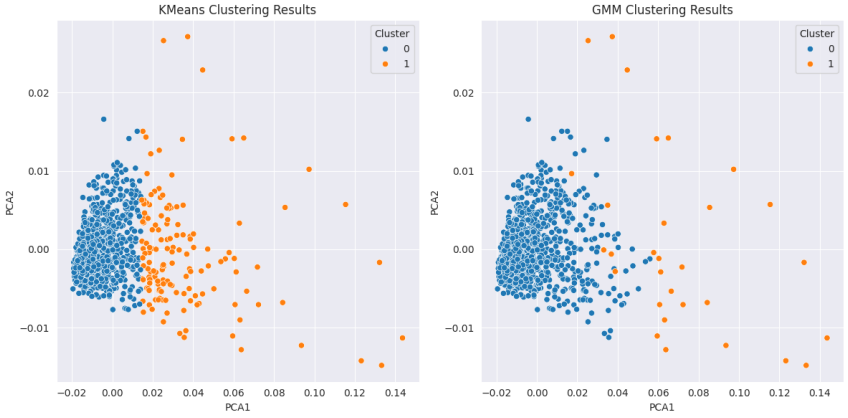
In the implementation of the soft clustering algorithm, specifically the Gaussian Mixture Model (GMM), similar analyses were conducted as with KMeans. Analogous to KMeans, the function employed to execute GMM evaluates the quality of clusters and characterizes them based on the prevailing features. Additionally, insights on the top 3 features per distance from each cluster centroids are reported.
In particular, the following observations were made:

- Silhouette values are slightly higher when using the dataset reduced through PCA. They increase significantly when using the differently normalized dataset. A similar analysis can be conducted when adopting the dataset with only numerical features; the results obtained are better compared to the entire dataset.

To visualize the results obtained through the use of the two clustering algorithms, we employed PCA, setting the number of components to two, allowing easy visualization in a 2D space. The visualization was conducted on the original dataset, the dataset reduced through PCA, the dataset normalized with the second normalization technique and the dataset containing only numerical features.
Here below,only the most significant plots are showcased.



(a) K-Means and GMM clustering on the original dataset



(b) K-Means and GMM clustering on different normalization technique dataset

(c) K-Means and GMM clustering on the numerical feature dataset

2.2.3 Purity assessment

Purity measures helps to understand how well-separated and internally consistent the clusters are with respect to the actual labels. The purity is calculated for each dataset used in the previous steps. Here are the results obtained for the original dataset.

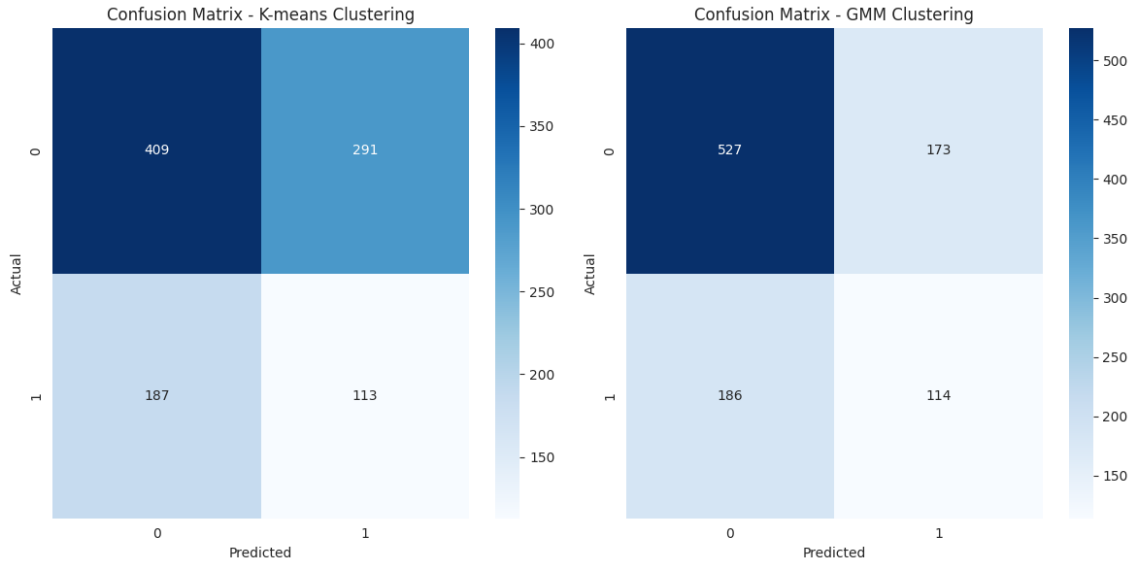


Figure 2.8: Clustering on the original dataset

- Purity of cluster 0 (K-means): 0.69
- Purity of cluster 1 (K-means): 0.72
- Purity of cluster 0 (GMM): 0.74
- Purity of cluster 1 (GMM): 0.60

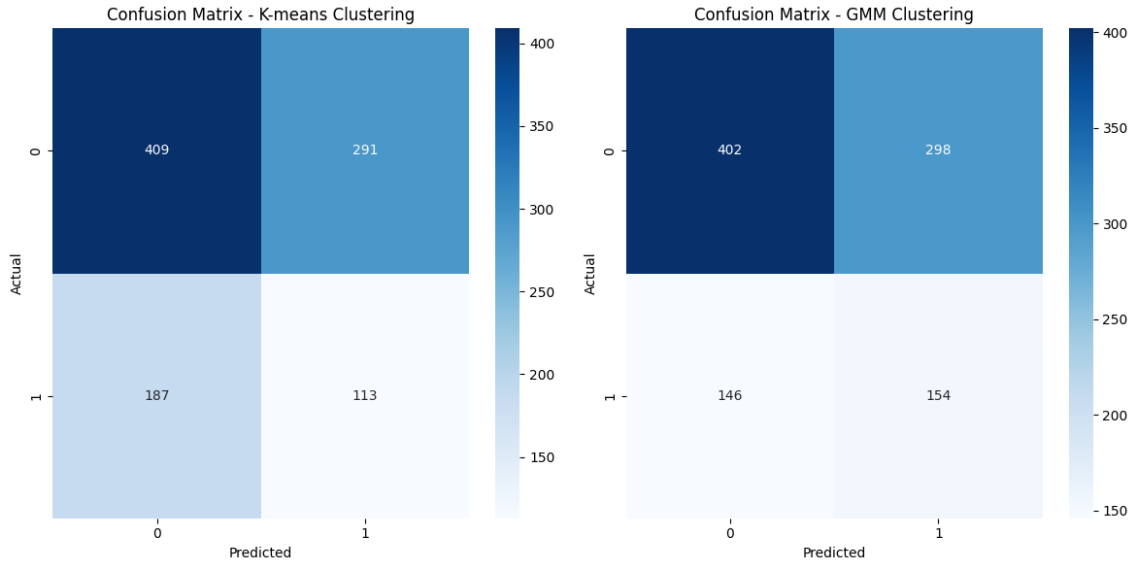


Figure 2.9: Clustering on the reduced dataset

- Purity of cluster 0 (K-means): 0.69
- Purity of cluster 1 (K-means): 0.72
- Purity of cluster 0 (GMM): 0.73
- Purity of cluster 1 (GMM): 0.66

Below, there are also the results of the dataset normalized with the other normalization technique proposed.

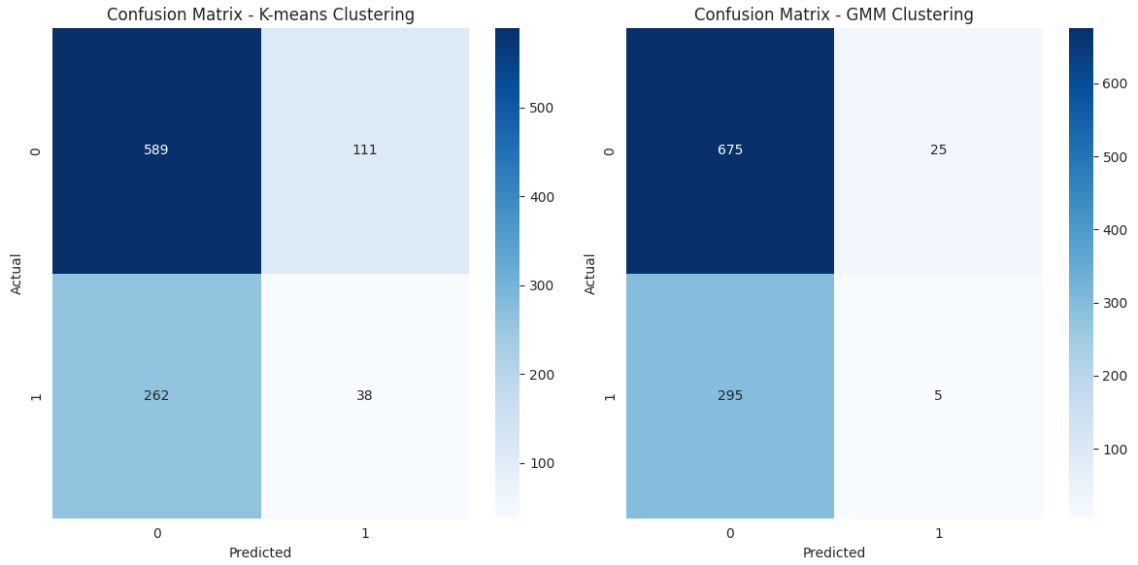


Figure 2.10: Clustering on the scikit-learn standardized dataset

- Purity of cluster 0 (K-means): 0.69
- Purity of cluster 1 (K-means): 0.74
- Purity of cluster 0 (GMM): 0.70
- Purity of cluster 1 (GMM): 0.83

Conclusion

The analysis conducted using clustering algorithms and evaluating purity has led to a series of assessments: in order not to lose information, the original dataset will be retained to explore supervised learning tasks. Indeed, the results using a reduced dataset do not change significantly from the original one. Additionally, to address the mentioned attacks in 4^{th} point effectively and with the aim of enhancing the interpretability of the attacks, namely in the point 4.2 where single features will be attacked, the starting dataset was retained.

Chapter 3

Supervised Data Analysis

3.1 Classifier Selection

As mentioned earlier,a pipeline was defined for each selected classifier. Firstly, in the pipelines the scaling operation of the different training and validation folds is performed, by ensuring no leak of information between the different splits. The scaling was performed via a min-max scaler.

5 classifiers were selected in order cover most of the classifiers treated during the course:

- Logistic Regression
- Random Forest
- Artificial Neural Network
- Support Vector Machine
- Complement Naive Bayes

Regarding the **Artificial Neural Network**, after a rigorous analysis 1 hidden layer with 32 neurons was chosen, as by trying distinct values for the number of neurons and the number of hidden layers it can be found out that this solution provides an adequate hypothesis for this task. By doing so, a rule of thumb that states how the number of hidden neurons should be between the size of the input layer and the output layer was followed. The activation function for the hidden layer is the *relu* function. Additionally, since this is a binary classification task, the output layer exploits the *sigmoid* function, while *binary-crossentropy* is the loss function. Lastly, the Artificial Neural Network classifier was generated via the Keras library.

On the other hand, the **Complement Naive Bayes** classifier was picked, instead of the Naive Bayes Classifier that we analyzed during the course, since it is better suited for imbalanced datasets.

3.2 Cross Validation

First, stratified cross-validation was applied as it accounts for the unbalanced structure of the dataset by preserving the relative occurrence of each class across the single folds.

The **cross_val_features** function was designed in order to implement a rigorous **5-10 nested stratified cross-validation**. For all five outer folds the train-validation set was split into separate instances of training and validation sets, containing 80% and 20% of the data points, respectively. For each outer fold, the current training set was given to the inner cross-validation, for which we used a grid search cross-validation. The grid search cross-validation evaluates all possible combinations of the hyperparameters to find the best-performing one.

After a careful and meticulous analysis the **rationale** for the evaluation of the models was built by taking into consideration not only the validation accuracy, but also the **f1-score**. On this basis, for the inner cross-validation, "best-performing" was defined as "highest F1 score" since the

unbalanced nature of the dataset may easily lead to high accuracies, while other metrics such as F1, precision and recall, estimate the model's performance more realistically.

The tuned hyperparameters are defined for each model in the *param_dict* dictionary. Optimal hyperparameters of the inner folds were given back to the corresponding outer fold, which then fitted the model with those hyperparameters. Then, the model was evaluated using **accuracy** to adhere to the project's specifications. For the outer folds, accuracy can be used as an evaluation metric without concerns. This is rooted in the fact that the inner folds tuning for the highest F1 score led in all models to balanced class weights as a hyperparameter.

Additionally, the pipeline was employed for both inner and outer folds to avoid data leakage. Data leakage causes information from the train dataset to leak into the validation dataset, possibly creating misleading results. It should therefore be strictly avoided. Through the scaling of the data, data leakage would have occurred if data was not scaled separately for each unique train and validation dataset within each outer and inner fold. Hence, the pipeline first scales the data and then applies it to the model.

Lastly, regarding the adversarial attacks that will be conducted later, information about which features are more relevant for each model were obtained by utilizing the **feature permutation** technique to determine the feature importance for each model.

Tuned Hyper-Parameters

As we are dealing with an unbalanced dataset the tuning of the **class weight** parameter for all models that allow it was performed. This hyper-parameter was designed to control the trade-off between precision and recall, which in other terms, is the same as handling class imbalance.

Additionally, the following hyper-parameters were tuned for the different classifiers:

- **Logistic Regression**:
  1. Inverse of regularization strength (C).
  2. Maximum iterations for the solvers to converge.
  3. Solver algorithm to use in the optimization problem.
  4. Class Weight due to the unbalanced nature of the dataset.
- **Random Forest**:
  1. Number of estimators (trees) in the forest.
  2. Maximum Depth of each tree.
  3. Maximum Features to consider when looking for the best split.
  4. Minimum Samples Split required to split an internal node.
  5. Class Weight due to the unbalanced nature of the dataset.
- **Artificial Neural Network**:
  1. Batch Size for the stochastic optimization.
  2. Epochs number.
  3. Solver: which was selected as the Adam solver. This choice also tunes the learning rate.
  4. Class Weight due to the unbalanced nature of the dataset.
  5. As a recall, the tuning of the number of hidden layer, hidden neurons and the different activation functions was already previously addressed.
- **Support Vector Machine**:
  1. Inverse of regularization strength (C).
  2. Kernel type.
  3. Degree of the polynomial kernel function.

4. Gamma kernel coefficient.
5. Class Weight due to the unbalanced nature of the dataset.

- **Complement Naive Bayes**:
  1. Alpha: smoothing parameter

3.2.1 Logistic Regression

With the aforementioned set of tuned hyper-parameters the average validation metrics for the Logistic Regression model are presented in the following table:

Accuracy	0.71
Precision	0.51
Roc-Auc	0.78
Recall	0.70
F1-score	0.59

Table 3.1: Logistic Regression: Average Validation Metrics

The outer fold with the highest validation accuracy was the 2nd fold with validation accuracy of 0.72. The correspondent hyper-parameters are shown in the following table:

C	1
Penalty	l2
Max Iterations	25
Solver	liblinear
Class Weight	balanced

Table 3.2: Logistic Regression: Hyper-Parameters of the Highest Validation Accuracy Fold

Additionally, the information regarding the top 5 features by mean feature importance and the lowest 5 ones of the 2nd fold will come in hand later on when performing the adversarial attacks.

3.2.2 Random Forest

With the aforementioned set of tuned hyper-parameters the average validation metrics for the Random Forest model are presented in the following table:

Accuracy	0.72
Precision	0.53
Roc-Auc	0.79
Recall	0.65
F1-score	0.59

Table 3.3: Random Forest: Average Validation Metrics

The outer fold with the highest validation accuracy was the 2nd fold with validation accuracy of 0.73. The correspondent hyper-parameters are shown in the following table:

Max Depth	5
Number of Estimators	400
Min Samples Split	5
Max Features	sqrt
Class Weight	balanced

Table 3.4: Random Forest: Hyper-Parameters of the Highest Validation Accuracy Fold

Additionally, the information regarding the top 5 features by mean feature importance and the lowest 5 ones of the 2nd fold will come in hand later on when performing the adversarial attacks.

3.2.3 Artificial Neural Network

With the aforementioned set of tuned hyper-parameters the average validation metrics for the Artificial Neural Network model are presented in the following table:

Accuracy	0.72
Precision	0.53
Roc-Auc	0.77
Recall	0.69
F1-score	0.60

Table 3.5: Artificial Neural Network: Average Validation Metrics

The outer fold with the highest validation accuracy was the 5th fold with validation accuracy of 0.74. The correspondent hyper-parameters are shown in the following table:

Solver	Adam
Batch Size	64
Epochs	40
Class Weight	balanced

Table 3.6: Neural Network: Hyper-Parameters of the Highest Validation Accuracy Fold

Additionally, the information regarding the top 5 features by mean feature importance and the lowest 5 ones of the 5th fold will come in hand later on when performing the adversarial attacks.

3.2.4 Support Vector Machine

With the aforementioned set of tuned hyper-parameters the average validation metrics for the Support Vector Machine model are presented in the following table:

Accuracy	0.69
Precision	0.49
Roc-Auc	0.78
Recall	0.75
F1-score	0.59

Table 3.7: Support Vector Machine: Average Validation Metrics

The outer fold with the highest validation accuracy was the 2nd fold with validation accuracy of 0.72. The correspondent hyper-parameters are shown in the following table:

C	1.5
Kernel	linear
Degree	2
Gamma	scale
Class Weight	balanced

Table 3.8: Support Vector Machine: Hyper-Parameters of the Highest Validation Accuracy Fold

Additionally, the information regarding the top 5 features by mean feature importance and the lowest 5 ones of the 2nd fold will come in hand later on when performing the adversarial attacks.

3.2.5 Complement Naive Bayes

With the aforementioned set of tuned hyper-parameters the average validation metrics for the Complement Naive Bayes model are presented in the following table:

Accuracy	0.70
Precision	0.50
Roc-Auc	0.77
Recall	0.70
F1-score	0.59

Table 3.9: Complement Naive Bayes: Average Validation Metrics

The outer fold with the highest validation accuracy was the 5^{th} fold with validation accuracy of 0.73. The correspondent hyper-parameter is shown in the following table:

Alpha	2.0

Table 3.10: Complement Naive Bayes: Hyper-Parameter of the Highest Validation Accuracy Fold

Additionally, the information regarding the top 5 features by mean feature importance and the lowest 5 ones of the 5^{th} fold will come in hand later on when performing the adversarial attacks.

3.3 Classifier Evaluation

As requested by the specifications the performances for each classifier with the correspondent tuned hyper-parameters were evaluated on the test set.

3.3.1 Logistic Regression

The Logistic Regression classifier effectiveness when predicting loan attribution on the test set is determined by the following metrics:

Accuracy	0.70
Precision	0.49
Roc-Auc	0.78
Recall	0.70
F1-score	0.58

Table 3.11: Logistic Regression: Metrics on Test Set

Furthermore, the ROC curve and the Confusion Matrix obtained are below presented

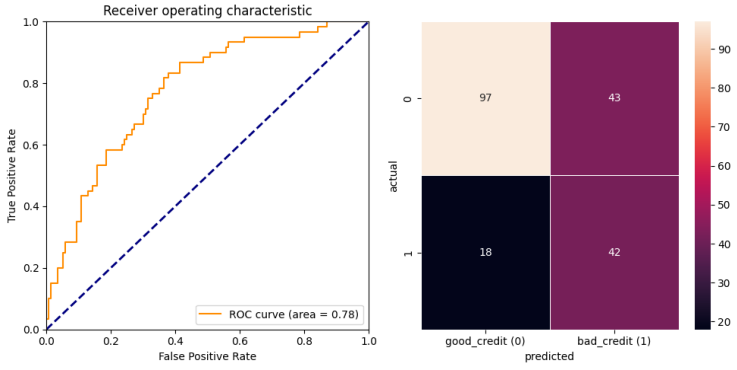


Figure 3.1: Logistic Regression: ROC Curve and Confusion Matrix

3.3.2 Random Forest

The Random Forest classifier effectiveness when predicting loan attribution on the test set is determined by the following metrics:

Accuracy	0.72
Precision	0.53
Roc-Auc	0.78
Recall	0.68
F1-score	0.59

Table 3.12: Random Forest: Metrics on Test Set

Furthermore, the ROC curve and the Confusion Matrix obtained are below presented

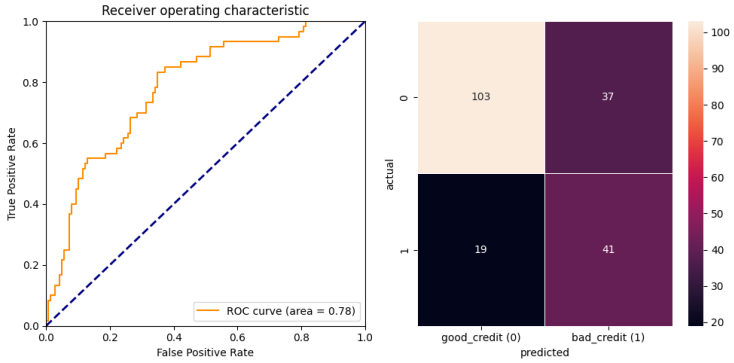


Figure 3.2: Random Forest: ROC Curve and Confusion Matrix

3.3.3 Artificial Neural Network

The Artificial Neural Network classifier effectiveness when predicting loan attribution on the test set is determined by the following metrics:

Accuracy	0.74
Precision	0.55
Roc-Auc	0.80
Recall	0.77
F1-score	0.64

Table 3.13: Artificial Neural Network: Metrics on Test Set

Furthermore, the ROC curve and the Confusion Matrix obtained are below presented

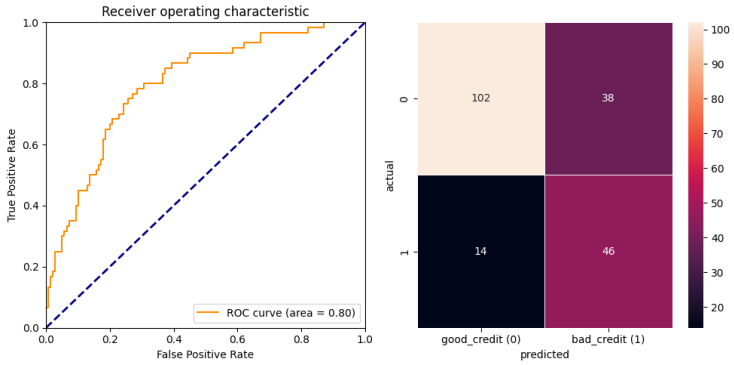


Figure 3.3: Artificial Neural Network: ROC Curve and Confusion Matrix

3.3.4 Support Vector Machine

The Support Vector Machine classifier effectiveness when predicting loan attribution on the test set is determined by the following metrics:

Accuracy	0.69
Precision	0.49
Roc-Auc	0.77
Recall	0.70
F1-score	0.58

Table 3.14: Support Vector Machine: Metrics on Test Set

Furthermore, the ROC curve and the Confusion Matrix obtained are below presented

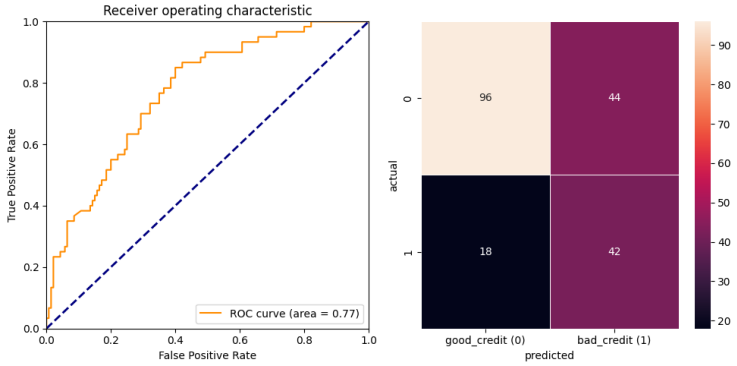


Figure 3.4: Support Vector Machine: ROC Curve and Confusion Matrix

3.3.5 Complement Naive Bayes

The Complement Naive Bayes classifier effectiveness when predicting loan attribution on the test set is determined by the following metrics:

Accuracy	0.71
Precision	0.51
Roc-Auc	0.79
Recall	0.78
F1-score	0.62

Table 3.15: Complement Naive Bayes: Metrics on Test Set

Furthermore, the ROC curve and the Confusion Matrix obtained are below presented

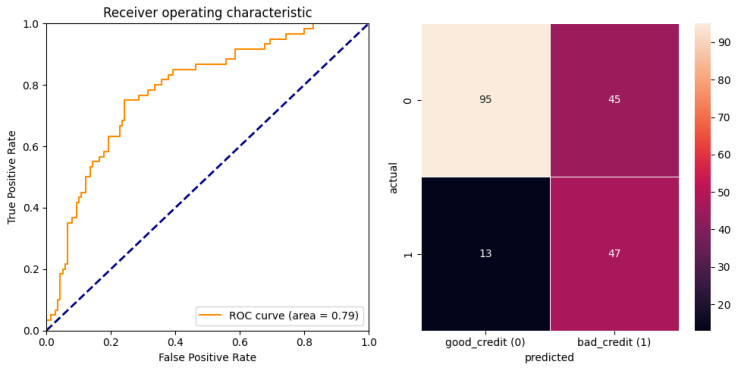


Figure 3.5: Complement Naive Bayes: ROC Curve and Confusion Matrix

3.3.6 Classifiers Comparison

To recap, a thorough comparison on the different classifiers' evaluation was implemented that is best summarized in the following figure

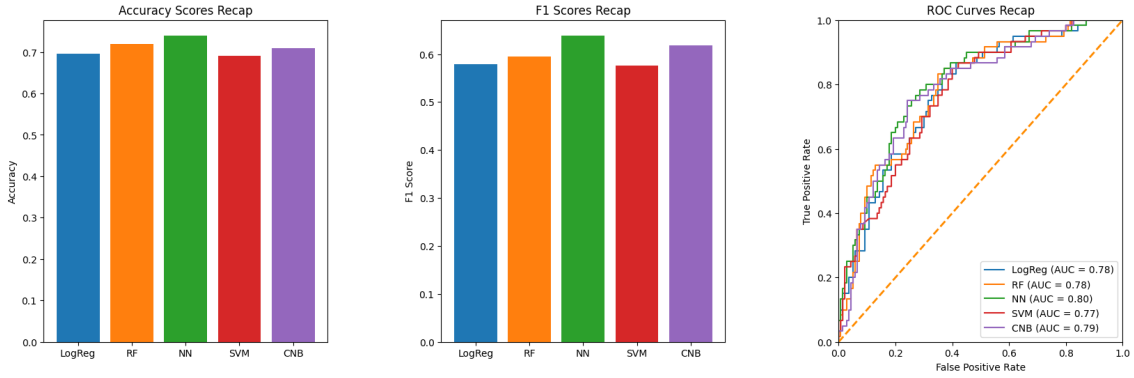


Figure 3.6: Classifiers Comparison

Accuracy Scores

Regarding the Accuracy Scores all classifiers performed in the $[0.69 \rightarrow 0.74]$ range.
The Artificial Neural Network classifier prevailed, while the Random Forest and the Complement Naive Bayes models achieved, respectively, the 2^{nd} and the 3^{rd} place. Lastly, the Logistic Regression and Support Vector Machines presented the lowest results.

F1-Scores

Given the unbalanced nature of the dataset it is important to stress on the F1-score metric, on this basis, all classifiers performed in the $[0.58 \rightarrow 0.64]$ range.
Again, the Artificial Neural Network classifier overcame the other models. In this case, the Complement Naive Bayes prevailed in respect to the Random Forest. Whereas, both the Logistic Regression and Support Vector Machine models carried out the lowest results.

ROC Curves

For the ROC curves, all classifiers presented areas under the curve (AUCs) in the $[0.77 \rightarrow 0.80]$ range.
Similarly to the previous metrics, the Artificial Neural Network prevailed. Whereas, the remaining models performed similar results.

Conclusion

To conclude, the designed **rationale** of both considering the f1-score and accuracy metrics previously explained, led to optimal results both in terms of accuracy and in terms of the ability of each model to better deal with the unbalanced nature of the dataset.

Finally, the fact that the Artificial Neural Network classifier outperformed in all relevant metrics the other models leaves a positive outlook for the adversarial attacks that will be carried out on it later on.

Chapter 4

Adversarial Attacks

Nowadays it has become crucial to not only assess the overall effectiveness of Machine Learning models but also examine their true robustness and reliability. The focus has recently shifted towards adversarial robustness. This entails investigating whether we can create classifiers that remain resilient to intentional input perturbations executed by adversaries to deceive the classifier.

The common approach for training a classifier involves Empirical Risk Minimization, which aim is to minimize the average loss across a given training set. This optimization problem is typically solved using (stochastic) gradient descent.

In Neural Networks the gradient is efficiently computed using back-propagation. If we calculate the gradient of the loss with respect to the input x_i, the result provides insights into how small changes in the data impact the loss function. In the context of adversarial examples, rather than adjusting the parameters to minimize the loss, the objective is to adjust them to **maximize** the loss. In other words we aim to ensure that our adversarial example closely resembles the original input by optimizing the perturbation applied to x.

4.1 Random Noise

A first approach to better understand this adversarial context is to apply random noise to the test data and measure its impact on classifier accuracy.

To achieve that, random noise from a Gaussian distribution was generated having the same shape as the test data and normalized the noise to have a predefined Euclidean distance. Which means that, when applying this random noise to the starting dataset, the resulting *noisy* dataset has euclidean distance clipped to a certain value in respect to the starting dataset.

Finally this noise is applied by adding it to the test data and evaluate its impact on each classifier presented in the previous chapter.

	Accuracy (clean)	F1 (clean)	Robust Accuracy	Robust F1	L2 Distance (**sf**)
LR	0.70	0.58	0.55	0.39	150 (2.77)
RF	0.72	0.59	0.64	0.29	150 (2.77)
SVM	0.69	0.58	0.65	0.27	150 (2.77)
NN	0.74	0.64	0.58	0.25	200 (3.7)
CNB	0.71	0.62	0.64	0.21	200 (3.7)

From the obtained results, it is evident that the average degradation of accuracy and F1-measure occurs around an L2 distance of 150, except for the NN and CNB classifier, which both degrades only towards a distance value of 200.

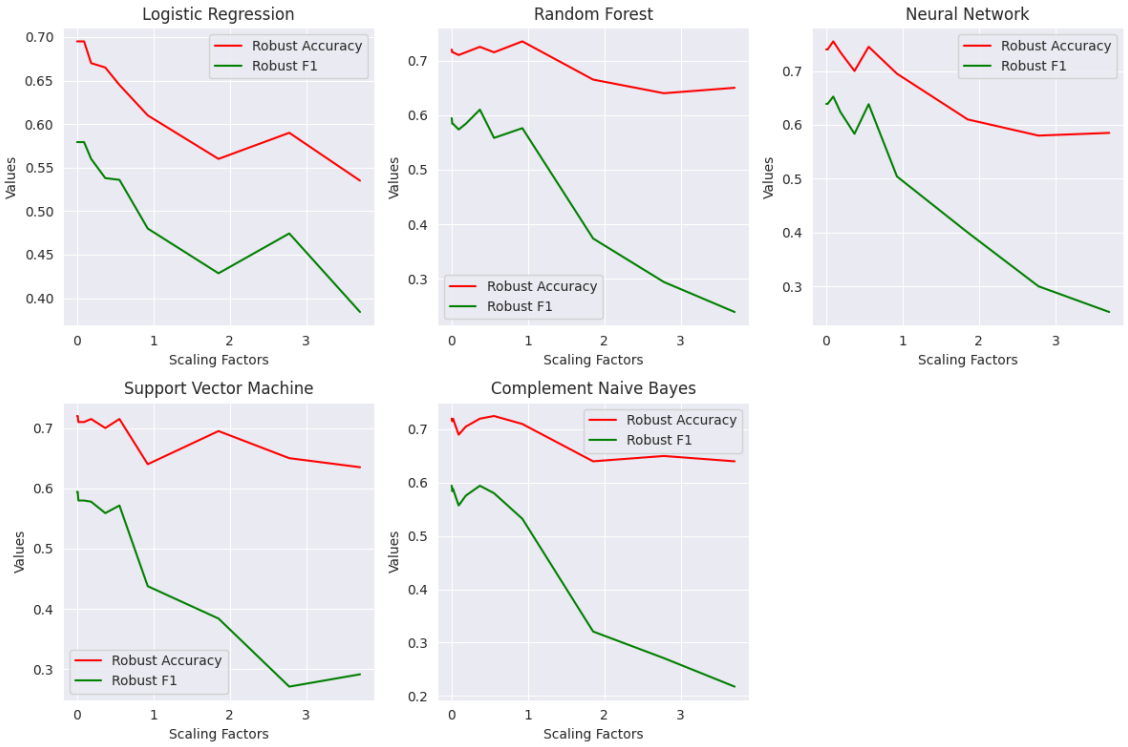


Figure 4.1: Robust Accuracy and Robust F1 score comparison for each classifier

4.2 Feature specific noise

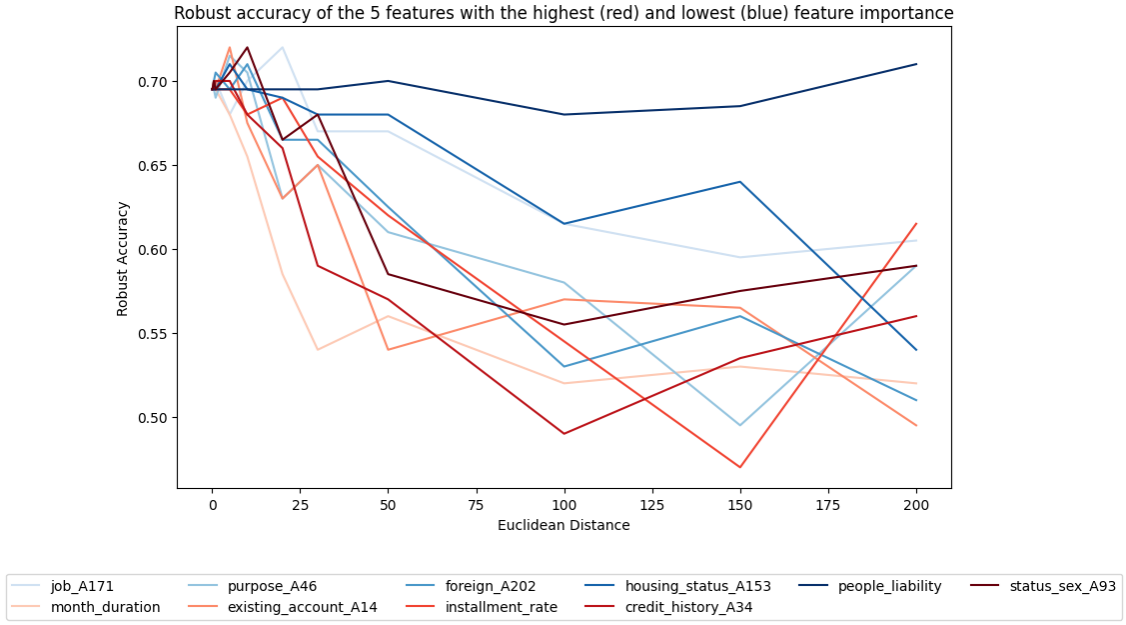
The next step in the study focuses on investigating the impact of random noise on individual features for each classifier.

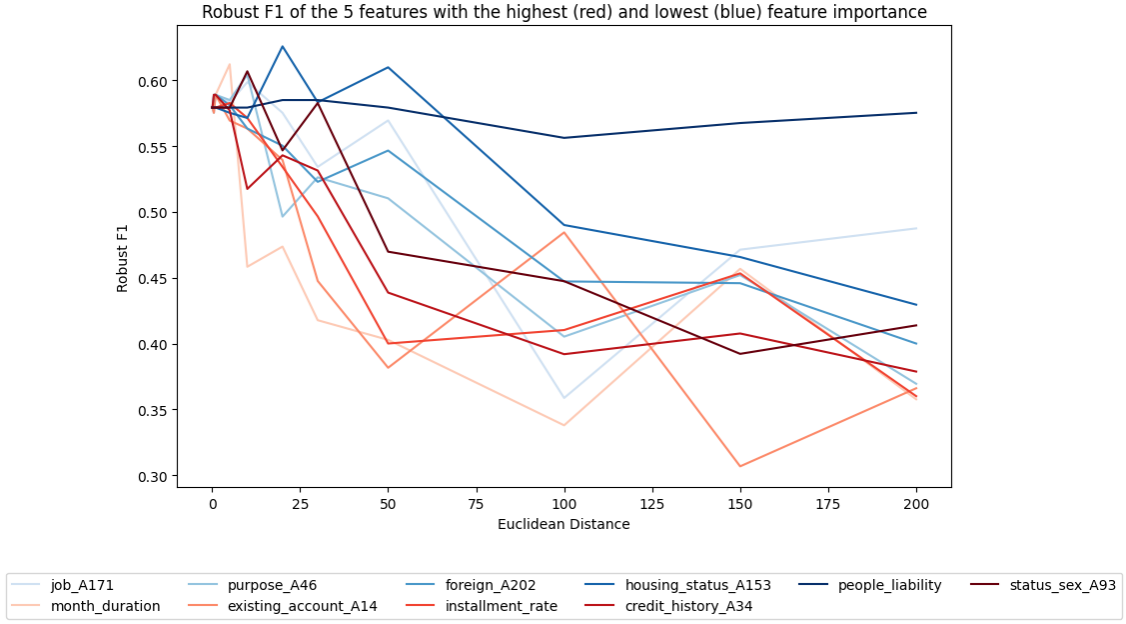
This analysis is carried out on the *top 5* features and *low 5* features identified during the feature importance assessment in the preceding chapter. We iterate over these features calculating the robustness (either accuracy or F1 score) for each of them.

The analysis reveals, as expected, that top 5 important features significantly impact classifier accuracy, with a notable decrease observed at specific distance values. Random Forest consistently outperforms other classifiers, maintaining a good accuracy even with low-importance features. Neural Network, SVM and CNB show similar behavior, with low-importance features affecting accuracy and F1 score at specific distances.

Overall, the impact of features against random noise varies across classifiers, especially highlighting the average L2 distance of 100 as a critical point for feature degradation, with Random Forest demonstrating higher robust performance.

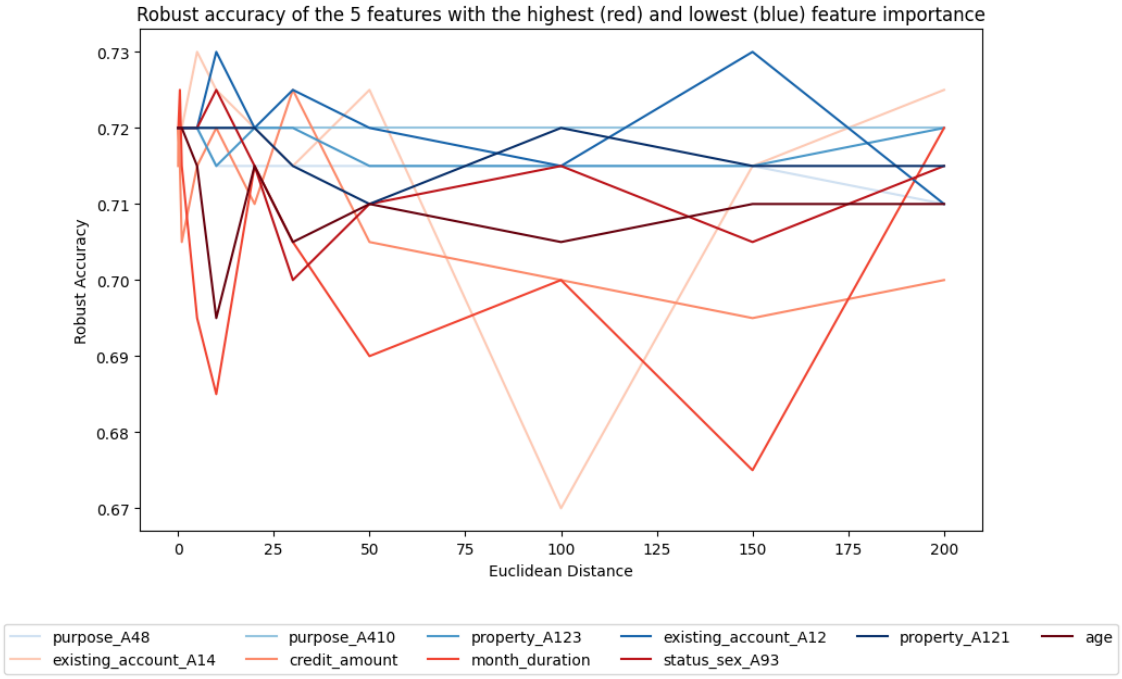
Logistic Regression

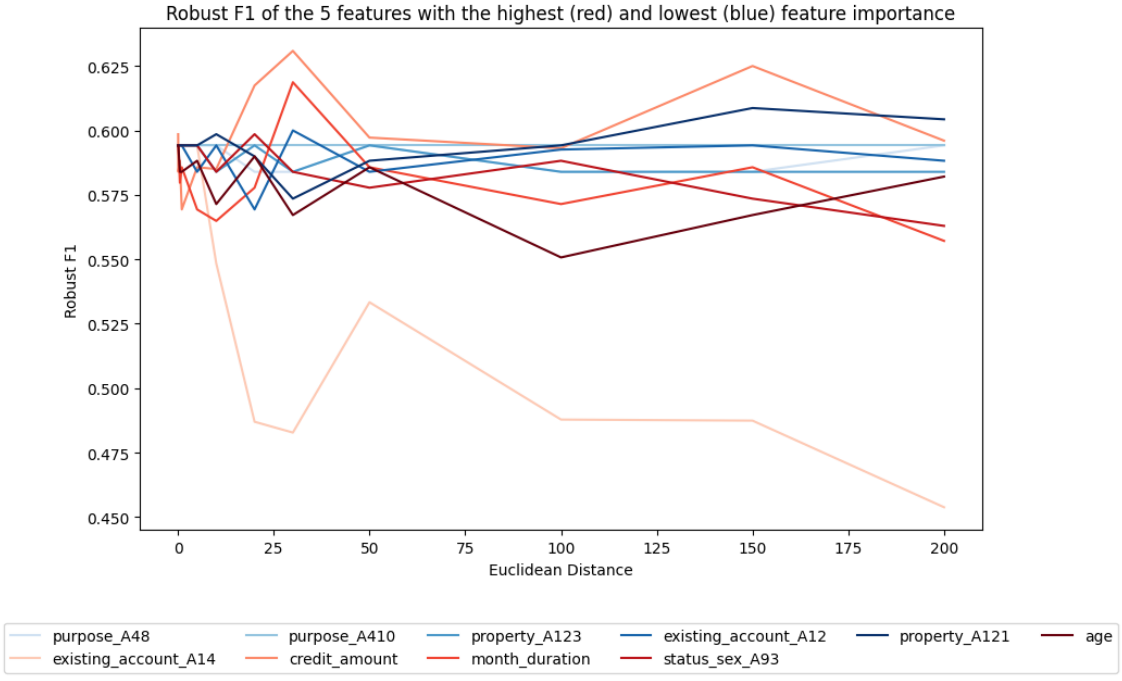




As indicated by the plot, the top 5 important features have a significant impact on accuracy compared to the low 5 important features. Particularly worth to note is the "**credit_history**" feature, which decreases the initial classifier accuracy from 0.70 to 0.49 at a distance value of 100, and the feature "**installment_rate**" decreasing even less, 0.47 accuracy at distance value of 150. Regarding the low-importance features, only the "**purpose**" feature results in the lowest robust accuracy of 0.49, and this occurs at a very long distance of 200. The same pattern for the mentioned features is observed in F1 measurements, but at a slightly larger distance of 150.

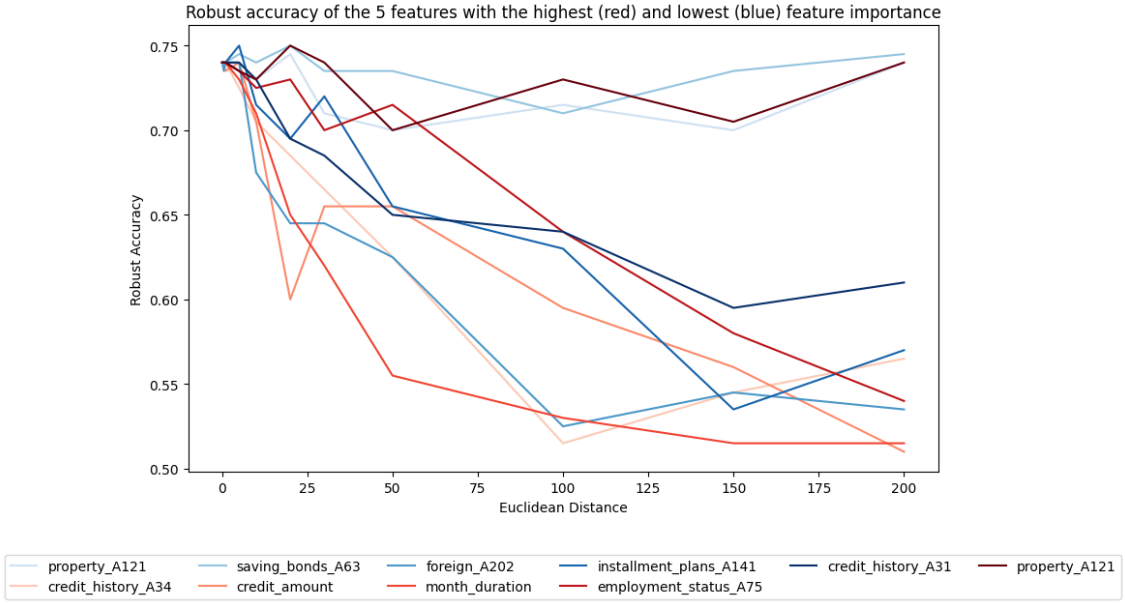
Random Forest Logistic Regression

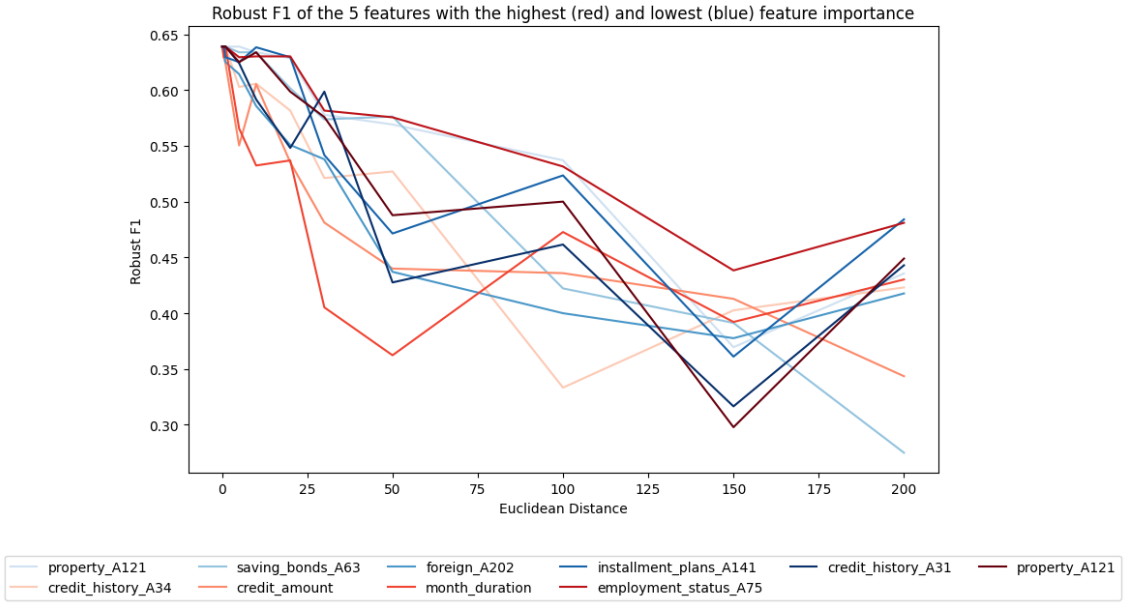




Random Forest demonstrates consistently better results, both on the top 5 important features and, notably, on the low 5 important ones. It is evident that even the lowest accuracy, starting from 0.72 and associated with the feature "**existing_account**", remains above 0.67. In contrast, all the low-importance features do not appear to degrade at all the classifier's strength. In terms of F1 score, apart from "existing_account", the other features do not significantly decrease the score.

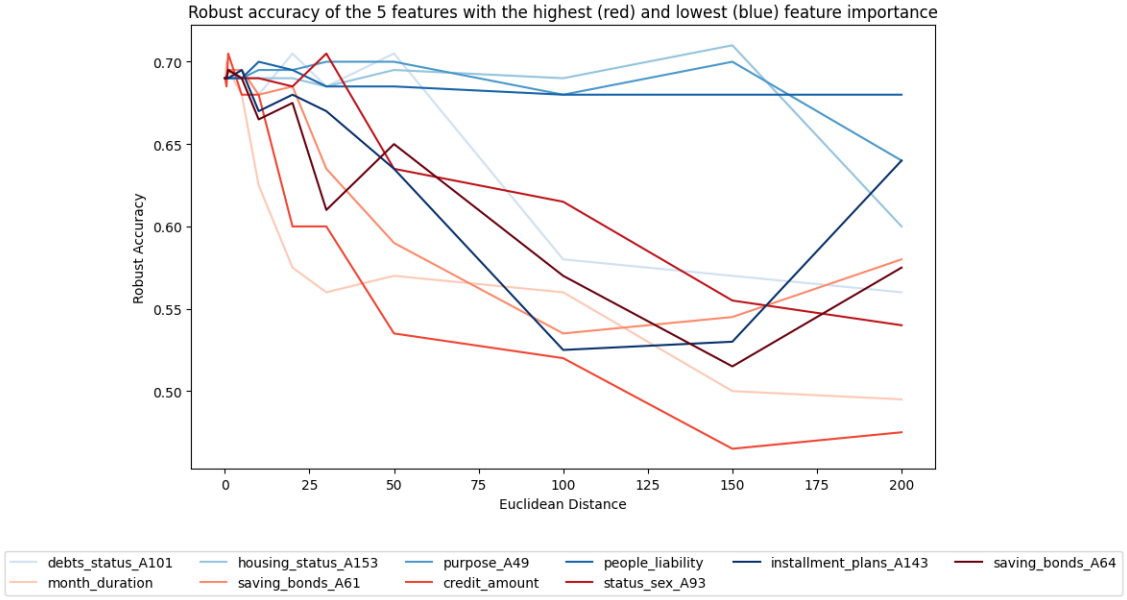
Neural Network

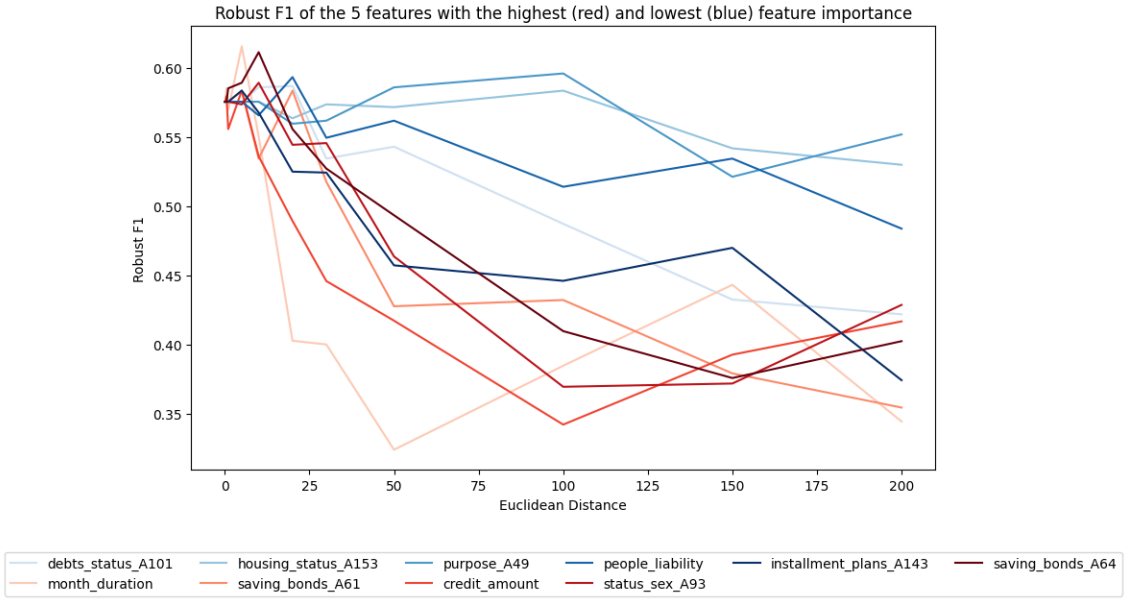




The Neural Network produces similar output to the logistic classifier. However, it is remarkable that the low-importance features demonstrate better performance by reducing both accuracy and F1 score at a distance value of 150. The feature "**credit_amount**" and "**month_duration**" decrease the accuracy of the Neural Network from 0.74 to a minimum value of 0.51, which occurs at a long distance value of 200.

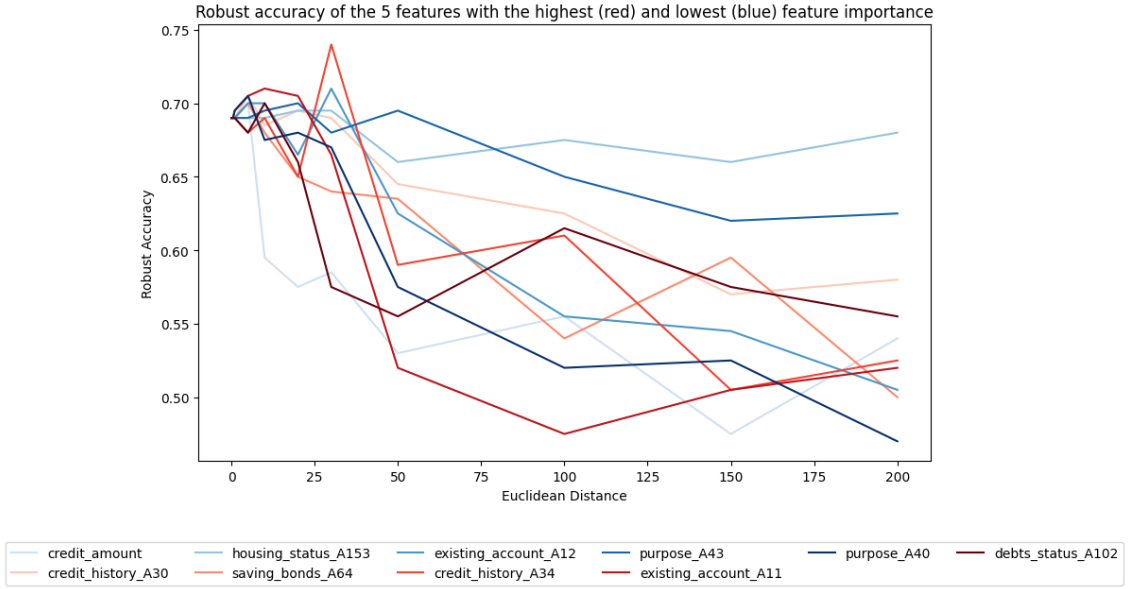
Support Vector Machine

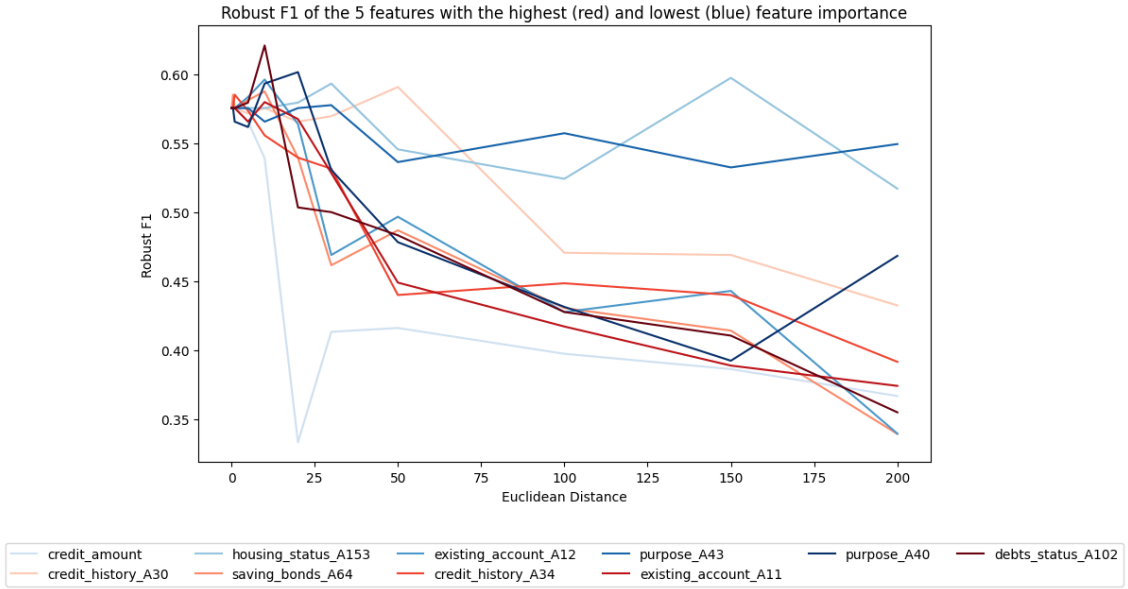




On the other hand, SVM shows a behavior similar to random forest, where low-importance features do not significantly weaken the classifier. However, the feature "**installment_plans_A143**" stands out as it considerably reduces accuracy at a distance of 100 and negatively impacts the F1 score after a distance of 150.

Complement Naive Bayes





CNB, aligning with the overall behavior of the previous classifiers, demonstrates a distinctive pattern. Notably, the low importance feature "**purpose_A43**" degrades the accuracy to 0.47, similar to the impact of the top importance feature "**existing_account_A11**" but at twice the distance (200 instead of 100).

4.3 Adversarial Attack with ART

Before delving on the actual attacks let's start by considering the inner maximization problem

$$\underset{\|\delta\| \leq \epsilon}{\text{maximize}}\, L(h_\theta(x), y)$$

where $h_\theta(x)$ represents our Neural Network. How do we solve this optimization problem? We can find a **lower bound** on the optimization objective $h_\theta(x + \delta)$. Because (by definition) any feasible δ will give us a lower bound, i.e., this is equivalent to find an adversarial example.

$$\underset{\|\delta\| \leq \epsilon}{\text{maximize}}\, L(h_\theta(x + \delta), y)$$

The basic idea is that, using back-propagation, we can compute the gradient of the loss function with respect to the perturbation δ itself. Therefore, a logical approach is to leverage gradient descent on δ to maximize our objective. However, it is crucial to maintain δ within the specified norm bound ϵ, which describes the magnitude of our perturbation.

The underlying philosophy behind adversarial examples lies in the idea that we can identify a subset within the allowed space of perturbations. This subset is characterized by the property that, according to any "reasonable" definition, the semantic content of our data should remain unchanged despite the applied perturbation. A common perturbation set to use is the L^∞ ball or the L^2 ball.

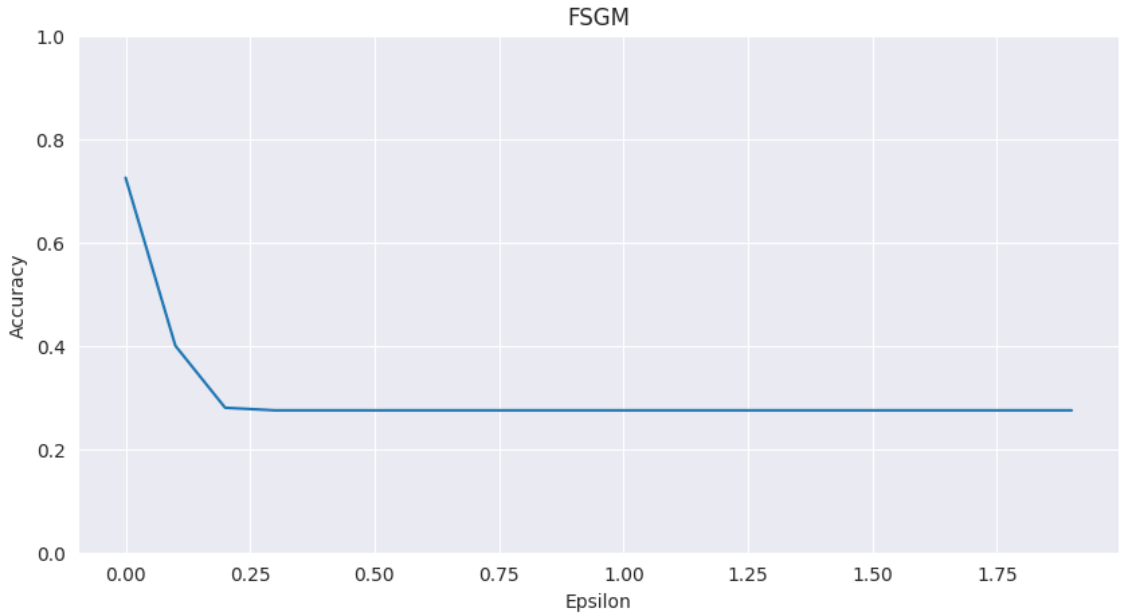
These attacks are known as **white-box** attacks, documented within the ART library as **evasion** attacks. It's important to note that executing such attacks might require access to the targeted model.

4.3.1 FGSM

Fast Gradient Sign Method is a white box, **non-targeted** evasion attack. Non-targeted means that we are trying to maximize the loss for the true class label.

In FGSM the objective is to maximize the loss by adjusting the perturbation, denoted as δ, in the direction of the gradient. This involves taking a step and subsequently projecting back into the norm ball, where $||\delta||$ is constrained by ϵ. To achieve maximum loss increase, it's crucial to take the largest step possible.

For significant values of α, the update simplifies to $\delta := \epsilon \cdot \text{sign}(g)$, where ϵ controls the magnitude of the perturbations and g is the gradient. It's worth noting that FGSM, by design, computes the gradient only once and is not explicitly tailored for finding minimal perturbations. Consequently, this method tends to result in larger perturbations and requires manual adjustment of the ϵ parameter to achieve the desired outcomes.



For this type of attack, as well for the future attacks, we established an ϵ range $[0, 2]$ incrementing in steps of 0.1. This range allow us to explore what happens to the robust accuracy following the attack across different magnitudes of perturbation. Starting with an initial accuracy of 0.72, the robust accuracy dramatically decreases to 0.27 when ϵ is set to 0.4.

FGSM is very powerful against linear binary classification but since we know that neural networks are not in fact linear even over a relatively small region, if we want a stronger attack we likely want to explore better methods at maximizing the loss function.

4.3.2 PGD L^∞

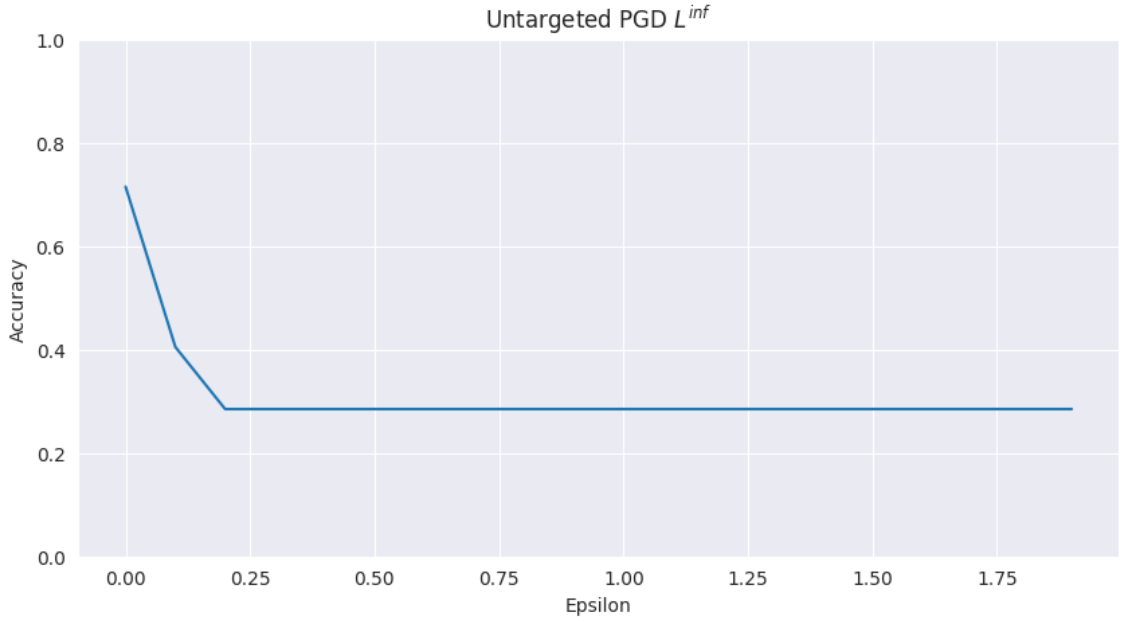
The basic approach of **Projected Gradient Method** is to re-iterate the updates that we have seen with FGSM. With PGD we have more choices such as the actual stepsize itself, and the number of iterations. PGD is also a white-box, **non-targeted** evasion attack. White-box, since we need to know the gradients of the model to create the adversarial examples.

PGD calculates the gradient multiple times and attempts to find the perturbation that maximizes the loss of a model on a particular input while keeping the size of the perturbation smaller than ϵ. This is good because it helps preserving similar appearance to the original data.

Repeat:

$$\delta := \mathcal{P}(\delta + \alpha \nabla_\delta L(h_\theta(x + \delta), y))$$

where $\mathcal{P}$ denotes the project onto the ball of interest, in this case L^∞.
PGD performance is limited by the possibility of local optima within the objective. To partial overcome this problem, we don't just run PGD once, but we run it multiple times from different random locations within the L^∞ ball (*"n_random_init=1"*).

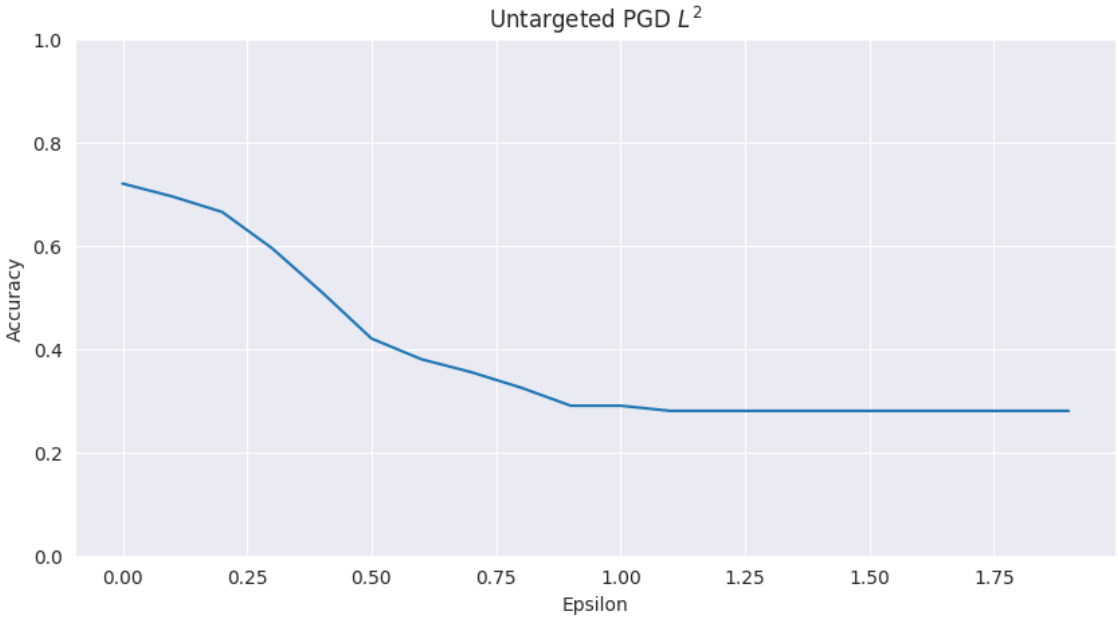


The results demonstrate a behavior similar to FGSM, yet achieving a comparable accuracy of 0.285 with only half the magnitude of ϵ, precisely 0.2.

4.3.3 PGD L^2

We have just been focusing on attacks where δ has bounded L^∞ norm. Now we try to run the same attack but with δ bounded to L^2 norm.
In this case the ϵ we need to consider for L^2 norm perturbations is larger than what we need for L^∞ perturbations, because the volume of the L^2 ball is proportional to $\sqrt{n}$ times the volume of the L^∞ ball. It's important to note that this means that L^∞ attacks lead to small noise (not good for adversarial defence) comparing to L^2 attack.

This type of attack results in perturbations that are more localized in the data. This is achievable because we can trade-off a larger perturbation in one area of the space for less perturbation in another.



As depicted in the figure, we can notice that we get same robust accuracy as PGD L^∞ only at a magnitude of 1.10.

4.3.4 PGD L^∞ Targeted

So far we have seen only untargeted attacks. In untargeted attacks, we try to change the label to any alternative. This time our aim is to change the label to a particular alternative, so instead of maximizing the loss for the true label we try to minimize the loss for the alternative label.

In this section, our objective is to deceive the classifier into misclassifying specifically those data points that initially lacked loan attribution, tricking the model into assigning them a positive loan attribution. To accomplish this, we assess the accuracy of the starting model, which is trained on the whole dataset, to exclusively classify a subset of the training set that comprises only the target data points. In other words, we assess the accuracy of the starting model into classifying only those loans considered as negative loans (label=1). Subsequently, the PGD targeted attack was deployed on the starting classifier, generating adversarial samples exclusively for this specific type of data points in the test set.

The results are very interesting, since the initial accuracy is only 0.53 and the Targeted PGD attack is able to degradate the accuracy to the point of reaching 0. Which means that the Targeted PGD attack is able to trick the starting model into classifying all bad loans into good loans.

4.3.5 DeepFool

DeepFool is a white-box attack, gradient based. This attack is iterative and works by computing the gradient of the classifier's decision boundary with respect to the input features.

DeepFool determines the minimal perturbation required to shift the input point across the decision boundary, progressively adjusting the input in a direction orthogonal to this boundary. The perturbation is then calculated by considering the linear approximation of the decision boundary. DeepFool aims to minimize the distance between the original input and the perturbed input, while ensuring misclassification.

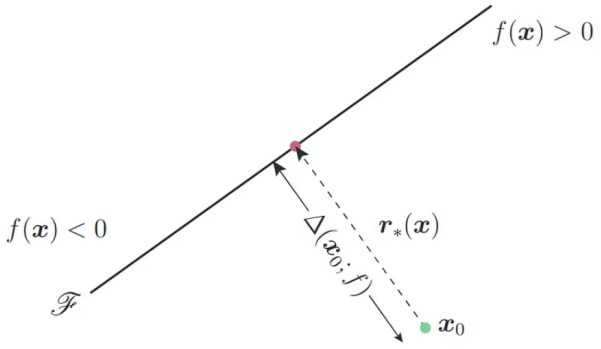


Figure 4.2: Adversarial example for binary classifier

The algorithm computes the perturbation by having the output prediction of the classifier decision function divided by the L^2 norm of the computed gradient of the loss function.

According to the results, starting from an initial accuracy of 0.75, the robust accuracy on adversarial test samples is observed to decrease substantially to 0.25.

4.3.6 Black Box Attack

Up until now, the presented attacks have exclusively relied on the gradients of the classifier. This implies that access to information about the victim classifier is essential for executing these types

of attacks.

In contrast to gradient based attack, a **decision based** attack is a **black-box** attack which generates adversarial samples via iterative optimizations using repeated queries. This is a powerful black-box attack because only requires final class prediction. Let's see an example.

4.3.7 HopSkipJump

HSJ is a **black-box** attack, which goal is to generate adversarial examples based solely on observing output labels returned by the targeted model. The algorithm iteratively refines the perturbation to move the input across the decision boundary while minimizing the perturbation size.

The effectiveness of the HopSkipJumpAttack is evident in the results, where the initial accuracy, starting at 0.71, experiences a substantial drop to 0.28 after just 200 iterations.

4.4 Countermeasure Exploration

4.4.1 Adversarial Training

The simplest strategy for training an adversarially robust model is augmenting the training process by incorporating adversarially examples. This approach is called **Adversarial Training** and seems to be very effective on gradient based attacks.

Adversarial training uses a modified loss function that is a weighted sum of the usual loss function on clean examples and a loss function from adversarial examples.

$$Loss = \frac{1}{(m-k)} \left(\sum_{i \epsilon CLEAN} L\left(X_i | y_i\right) + \lambda \sum_{i \epsilon ADV} L\left(X * adv_i | y_i\right) \right)$$

Adversarial training is often an iterative process. After training on a set of adversarial examples, the model is tested on a new set of adversarial examples to evaluate its robustness. This process may be repeated to further enhance the model's resistance to adversarial attacks. In our case study, after many rigorous attempts, we came to conclusion that **three** iterations are sufficient to demonstrate the effectiveness of this employed technique.

It's important to highlight that, when creating an instance of *AdversarialTrainer* method from ART library, the **ratio** parameter is explicitly set to 0.5. The ratio parameter describes the proportion of samples in each batch to be replaced with their adversarial counterparts. Setting this value to 1 allows to train only on adversarial samples which is not ideal for gaining a balanced adversarial training result.

Explained tabular parameters:

- **Clean Acc**: accuracy on clean test examples with adversarially trained model
- **Original Adv Acc**: accuracy on original adversarial samples with adversarially trained model
- **New Adv Acc**: accuracy on new adversarial samples with adversarially trained model

Accuracy on **benign test** examples: 72.5%

Iteration	Clean Acc(%)	Original Adv Acc (%)	New Adv Acc (%)
0	57.5	69.0	42.5
1	65.0	69.0	56.5
2	54.5	69.5	62.5

Table 4.1: Adversarial Training on FGSM - Accuracy Metrics

Accuracy on **benign test** examples: 71.5%

Iteration	Clean Acc(%)	Original Adv Acc (%)	New Adv Acc (%)
0	69.0	71.5	31.0
1	63.5	71.5	42.5
2	76.0	71.0	57.5

Table 4.2: Adversarial Training on PGD L^{∞} - Accuracy Metrics

Accuracy on **benign test** examples: 72.0%

Iteration	Clean Acc(%)	Original Adv Acc (%)	New Adv Acc (%)
0	62.0	72.0	38.0
1	57.5	61.0	45.5
2	70.5	71.5	62.5

Table 4.3: Adversarial Training on PGD L^2 - Accuracy Metrics

During adversarial training process on Targeted PGD, we made the decision to execute a single iteration, given the relatively small size of the dataset under consideration ($\sim$ 300 samples) $\rightarrow$ less effort during training to get desired results.

Accuracy on **benign test** examples: 53.3%

Iteration	**Clean Acc(%)**	**Original Adv Acc (%)**	**New Adv Acc (%)**
0	62.0	72.0	38.0

Table 4.4: Adversarial Training on Targeted PGD L^∞ - Accuracy Metrics

Accuracy on **benign test** examples: 75.0%

Iteration	**Clean Acc(%)**	**Original Adv Acc (%)**	**New Adv Acc (%)**
0	67.5	72.0	33.5
1	74.5	59.0	26.0
2	71.0	60.0	30.5

Table 4.5: Adversarial Training on DeepFool - Accuracy Metrics

The outcomes of the adversarial training yield important conclusions. With each iteration, the results consistently demonstrate an incremental improvement. Indeed, as new adversarial samples are generated and incorporated into the training process, there is a small but remarkable increase in robust accuracy. Based on this evaluation, a lower final robust accuracy value suggests that the defense mechanisms are less successful in mitigating the impact of the attack, highlighting the effectiveness of the adversarial techniques employed.

Here it is a standings from high to low effectiveness of the attacks:

1. DeepFool
2. PGD L^∞
3. PGD L^2
4. FGSM

The HopSkipJump attack has been intentionally excluded from this standing as it belongs to a distinct class of attack.

4.4.2 Gradient Obfuscation detection on PGD L^∞

Performing security evaluations is not just taking a set of attacks and running them against the defense, but it requires a careful analysis of the problem. **Gradient obfuscation** is the defense mechanism that makes the information contained in the input gradients useless leading gradient based attacks to less capability.

Gradient obfuscation techniques alter the gradients of the model to point to random directions, leading Gradient Descent during PGD attack to fail. The PGD attack is led to failure, or small effectiveness, because the optimizer is unable to continue the descent since it arrived in a region where the norms of the gradients are nearly zero, or the gradients are noisy, and the optimization lands on a bad local optimum. This is what happens when we adversarially train our model, we make no direction to follow for a new attack to find the optimum during Gradient Descent.

With ART library it is possible to plot a **security curve** of our classifier based on the **detection** of the **gradient obfuscation** reached with adversarial training.

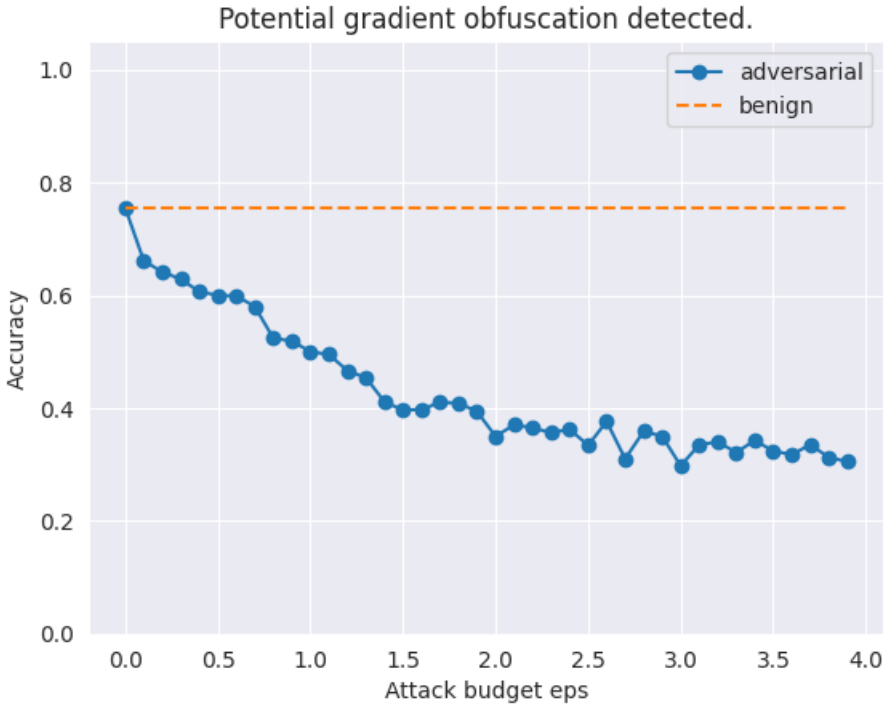


Figure 4.3: Security Curve on PGD L^∞

As we can see the landscape is highly irregular, it means the descent is noisy and the steepest descent might not point to a good solution.

Basically in the security curve, another PGD attack is carried out. The graph presented above illustrates that the accuracy not only fails to reach the minimum observed initially but, despite a larger magnitude of epsilon, remains significantly below the initial minimum accuracy.

Note: the ϵ range has been extended to 4 on purpose.

4.4.3 Input Sanitization

A common best-practice in CyberSecurity is the **Input Sanitization** paradigm. The idea is simple: Input sanitization, also known as data sanitization, refers to the process of cleaning or "sanitizing" user inputs to ensure safety. It doesn't affect input devices, but instead involves removing or modifying data that could potentially lead to system vulnerabilities or errors by checking if the input data respects some properties that the benign data follows.

Regarding the scope of our Project we can note specific properties about the data objects:

1. Since we perform the *MinMaxScaler* operation all features will be mapped in the $[0 \rightarrow 1]$ range. Therefore, whatever data object with any feature value greater than 1 or smaller 0 is detected might be the realization of an adversarial attack.
2. Additionally, the one-hot-encoded features, e.g. the non-numerical starting attributes, those features with "_A" in the final part of the feature name, after being normalized with the *MinMaxScaler* either assume the value 0 or 1.

The *sanitize_input_all_cols* function performs the input sanitization regarding the first property noted, whereas, the *sanitize_input_one_hot_encoded* will carry out the input sanitization on the second property.

By exploiting these functions we will check if the adversarial X_tests generated by each single attack respect both of these properties.

For instance, here by showing the differences between the benign *X_test* and the *Xtest_DF* adversarial generated by the DeepFool attack it is straightforward to note how Input Sanitizing can be used to perform a **first-line of defense**.

installment_rate	resident_since	age	credit_number	people_liability	credit_history_A30	credit_history_A31
1.000000	0.000000	0.312500	0.0	0.0	1.0	0.0
0.333333	0.666667	0.208333	0.0	0.0	0.0	0.0
0.333333	0.666667	0.604167	0.0	1.0	0.0	0.0
0.333333	1.000000	0.229167	0.0	0.0	0.0	0.0
0.333333	0.333333	0.583333	0.0	0.0	1.0	0.0
...	...	...	...	...	...	...
1.000000	0.333333	0.229167	0.0	0.0	0.0	0.0
0.000000	0.333333	0.020833	0.0	0.0	0.0	0.0
1.000000	0.333333	0.270833	0.5	0.0	0.0	0.0
1.000000	1.000000	0.458333	0.5	0.0	0.0	0.0
1.000000	1.000000	0.416667	0.5	0.0	0.0	0.0

Figure 4.4: Original normalized one-hot encoded X_test

installment_rate	resident_since	age	credit_number	people_liability	credit_history_A30	credit_history_A31
0.979612	0.001512	0.326756	-0.014017	-0.002980	0.971696	-0.014838
0.286377	0.672375	0.248702	-0.049193	0.037879	-0.055621	-0.037594
0.494682	0.619025	0.338208	0.058680	1.057478	0.360072	0.193900
0.274657	0.986722	0.275941	-0.022651	-0.009159	-0.076048	-0.039735
0.154517	0.389854	0.662213	-0.110653	-0.008759	0.872122	-0.085792
...	...	...	...	...	...	...
1.019096	0.335437	0.208440	0.008376	0.004274	0.028153	0.016848
0.011683	0.327775	0.009255	0.011738	-0.002117	0.033284	0.021813
1.009840	0.328107	0.248039	0.517112	-0.013382	0.031754	0.018996
0.951049	1.018582	0.484136	0.479331	0.014923	-0.052997	-0.030832
0.975957	1.011334	0.437385	0.479213	-0.000775	-0.035432	-0.022513

Figure 4.5: Adversarial X_test obtained from DeepFool attack

Both properties were not respected by the Deep-Fool, FGSM and PGD attack.

It is interesting to note how for the HopSkipJump attack the respective adversarial X_test_HSJ created still preserves the first property since all features are in the $[0 \rightarrow 1]$ range. Therefore, if we wouldn't check also for the second property regarding the one-hot-encoded features the HSJ attack would have made it through the Input Sanitization **first-line of defense**.

installment_rate	resident_since	age	credit_number	people_liability	credit_history_A30	credit_history_A31
0.938321	0.002319	0.357795	0.000000	0.006728	0.938686	0.000000
0.278431	0.686332	0.258954	0.000000	0.024188	0.000000	0.000077
0.554319	0.546406	0.411751	0.139094	0.987136	0.309462	0.200460
0.287784	0.995735	0.298304	0.000000	0.030119	0.000000	0.000000
0.211024	0.347826	0.655598	0.000000	0.023432	0.797241	0.000000
...	...	...	...	...	...	...
1.000000	0.324287	0.185429	0.078993	0.002554	0.080446	0.042270
0.027763	0.330802	0.000000	0.045973	0.000000	0.035362	0.009925
1.000000	0.317816	0.223137	0.556624	0.000000	0.067902	0.063577
0.898949	0.981999	0.513213	0.402194	0.000196	0.000000	0.000000
0.979623	0.997608	0.472195	0.485548	0.008022	0.000000	0.000000

Figure 4.6: Adversarial X_test obtained from HSJ attack