Politecnico di Torino

Machine Learning for Networking

Adversarial Attacks on Tabular Data Classifiers

Group 34

Lorenzo Filomena	Fabio Marmello	Cristian Racca	Andrea Raineri
s332995	s331410	s328892	s323351

Index

Introduction

In this project, we work on the German Credit Dataset, containing information about credit applicants and their associated credit risk. Initially, we focus on the analysis and preprocessing of the data, including categorical variables encoding and normalizing the data to ensure it is well-suited for machine learning tasks. We then apply both unsupervised and supervised techniques to the data. Final goal of our work is to find which among a list of well-known classifiers best suits our problem and later we explore some adversarial attack techniques to test the robustness of our model. The main tools we use in the project are the Scikit-learn[1] library, the K-Modes library[2], the Keras library[3] and the Adversarial Robustness Toolbox[4].

1 Data exploration and preprocessing

1.1 Dataset Acquisition

The dataset contains 1000 samples, each representing a unique loan applicant described by 20 distinct features having mixed data types, 13 categorical and 7 numerical. Some examples of the attributes of the dataset are `checking account balance`, `credit history`, `loan purpose`, `employment status`, and `property ownership`. Each sample has a label declaring the acceptance or refusal of the loan request to the bank, which we will use as ground truth for to train and evaluate the performance of our models.

1.2 Data Preprocessing

We preprocess the raw data to make it suitable for our machine learning tasks. Since the algorithms we are going to apply to the dataset expect a numerical input data type, we perform one-hot encoding on the categorical features. Despite significantly increasing the total number of features of the dataset when encoding variables with many different possible classes, this type of encoding offers some good properties for our task compared to other solutions (e.g. label encoding):

- As models interpret numerical values "as given", one-hot encoding helps us not introducing ordinal relationships between categories on which the model could possibly learn and be misleaded;
- For classification algorithms which assign weights to each input dimension (e.g. linear maps, multi-layer perceptron neural networks), each category can be differently weighted in the training phase, thus allowing flexibility to the models.

After applying one-hot encoding to the dataset we end up having 63 features. Being this generically a more than acceptable number of features, the limited quantity of records in the dataset could lead to overfitting issues in our models and reduced their generalization capabilities. In the next sections we explore some common dimensionality reduction techniques to possibly reduce the number of features.
After feature encoding we apply in parallel two different scale transformations on the data:

1. Min-Max, to bring each feature in the range $[0, 1]$. This helps removing the impact of different scales across the data dimensions and can lead to faster training and better overall performance of the models

2. Standardization, in order to level the variance of all the features for the Principal Component Analysis (PCA) step[1].

We finally split the dataset into training and test sets, with an 80:20 ratio.

1.3 Exploratory Data Analysis

In Figure 1 and 2 we plot boxplots and histograms of all numerical features, showing their empirical distributions, average trends and possible outliers.

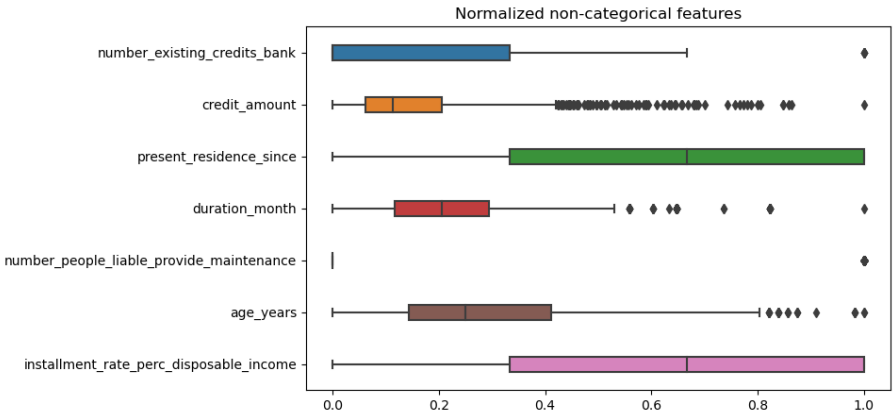


Figure 1: Boxplot of numerical features

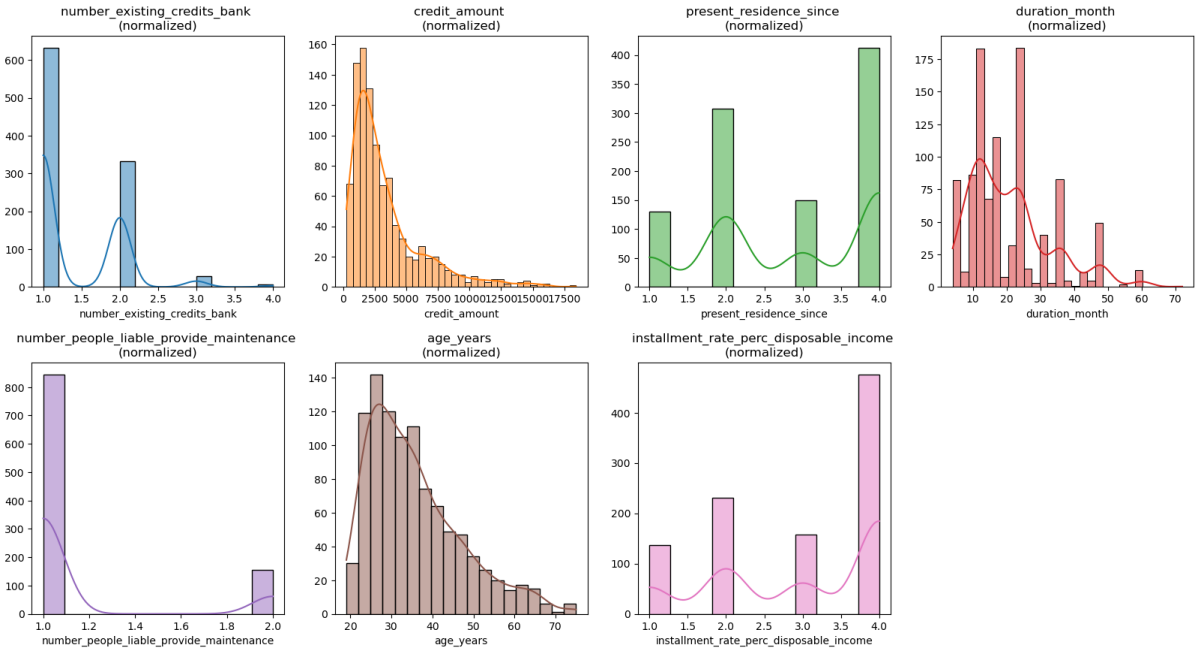


Figure 2: Empirical distribution of numerical features

As categorical features are more difficult to efficiently visualize, we try to assess which features could have been possible discriminants in loan attribution by computing the rejection rate for each class and testing for significant differences. Figure 3 shows four features

[1]If one feature "varies" more than the others mainly due to their respective scales, PCA would select such feature as dominant over the others in its principal components; therefore standardization PCA shows better results, with principal components having the same order of magnitude [5]

thata emerged from this analysis: `foreign_worker`, `status_existing_checking_account`, `saving_account_bonds` and `credit_history`.

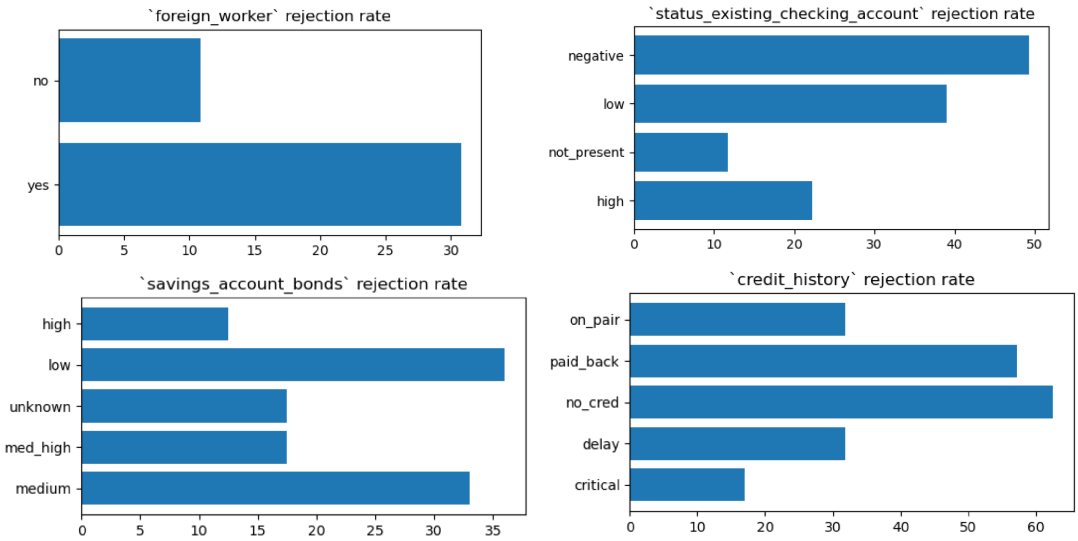


Figure 3: Rejection rates for different classes of categorical features

The correlation matrix of numerical features shows some correlation between the credit amount and the duration of the request loan, which we further explore also accounting for the ground truth. We clearly see a linear dependency between the two features, but no significant association to the loan attribution is detected (Figure 4).

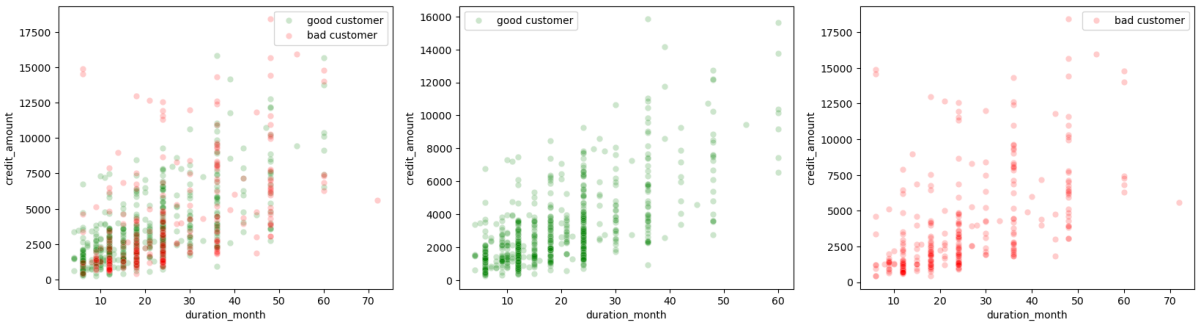
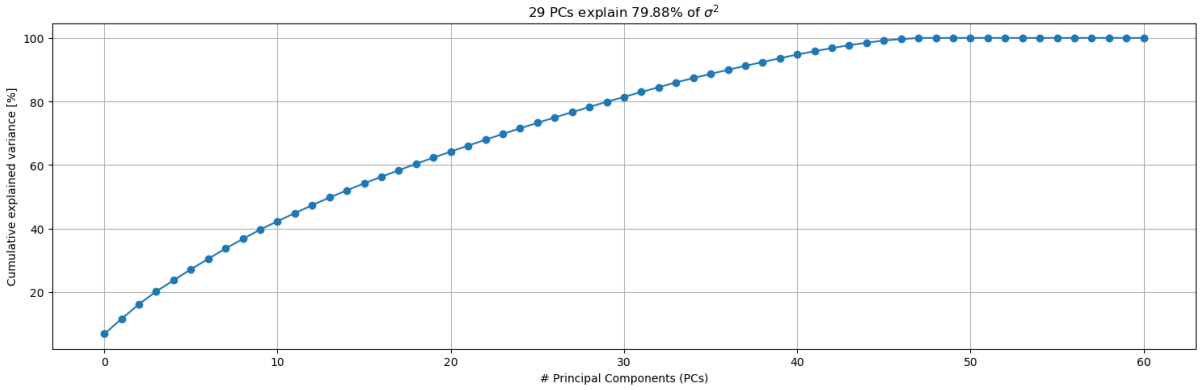


Figure 4: Scatter plot of `credit_amount` over `duration_month`, with loan attribution label

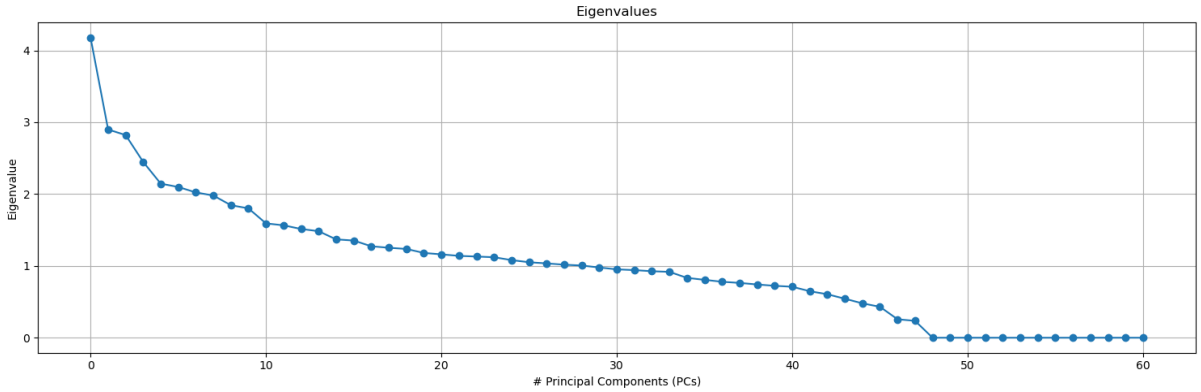
2 Unsupervised Data Analysis

2.1 Dimensionality reduction

We apply Principal Component Analysis (PCA) over the standardized data to select a reduced number of features without much loss of information. The number of principal components (PCs) to take can be chosen with different strategies. We test for the "Elbow method" and the "Kaiser criterion". As there is not an obvious inflection point in the cumulative explained variance plot (Figure 5a) required by the former method, we rely on the latter to take components with eigenvalues higher than 1 (Figure 5b). We end up selecting 29 PCs, with a cumulative explained variance of 78.28%.



(a) Cumulative explained variance over number of PCs



(b) $|\lambda|$ of PCs

Figure 5

We also apply the t-Distributed Stochastic Neighbor Embedding (t-SNE) transformation to visualize the data in two dimensions. We can see that the t-SNE was able to group some data points (Figure 6), possibly showing the presence of non linear relations between samples in the dataset which could be helpful in the classification task. In the next section we try to use common clustering techniques to possibly identify inherent patterns or groupings within data.

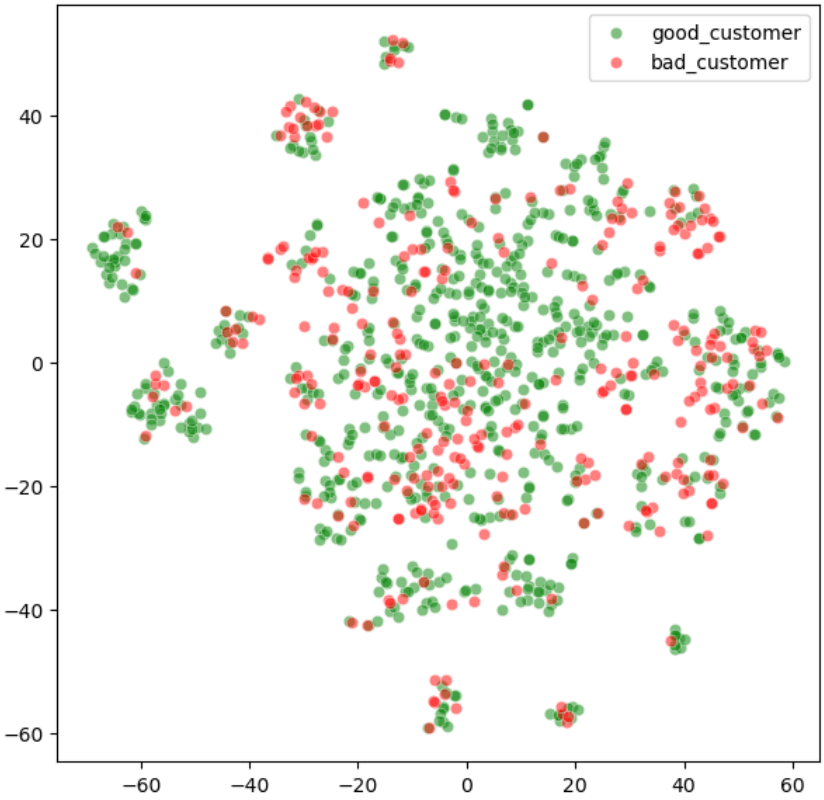


Figure 6: t-SNE applied to standardized data

2.2 Clustering

We apply two main families of unsupervised clustering algorithms to our dataset: DBSCAN and Partitive (k-Means family) algorithms. Given the mixed nature of the dataset, we first apply DBSCAN and k-Means directly on the PCA-transformed data.

k-Means clustering has two main tunable hyperparameters: the initialization technique, which can either be *random* or *kmeans++*, and the number of clusters k, of which we explored the range $[2, 10]$. Plotting the clustering error and the silhouette score (Figure 7), we select $k = 6$, as a sweet spot for low clustering error in both *random* and *k-means++* initializations and relatively high silhouette score. We assess the purity of the clusters compared to the ground truth and compare the most recurring features within the clusters: for example the Cluster "1" has purity = 0.89 and the loan purpose is not for education or retraining, the applicant is not a foreign worker and he owns or rents a house.

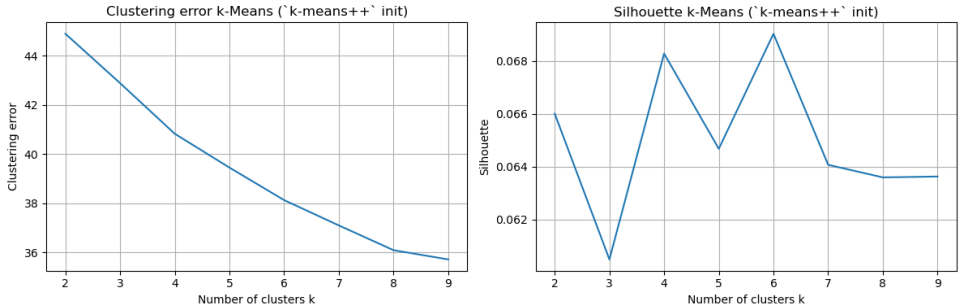


Figure 7: Clustering error and silhouette score for k-Means over number of cluster k (on PCA-transformed data)

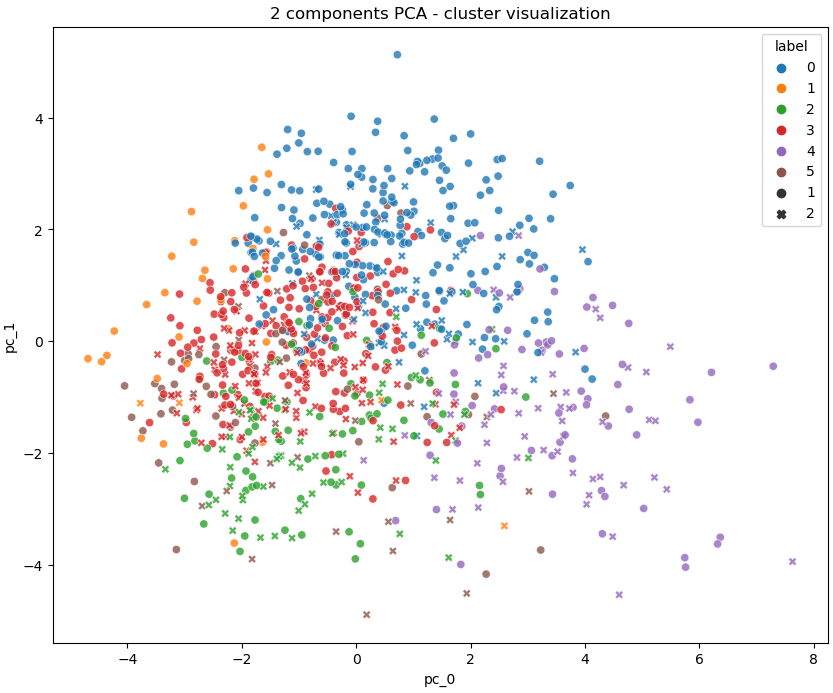


Figure 8: k-Means cluster visualization in 2 components (on PCA-transformed data)

DBSCAN has two main tunable parameters: the minimum number of points around core points and the coefficient ϵ which is the radius of the neighborhood around a point. We compute the silhouette of DBSCAN clustering both including and excluding the outliers (Figure 9), since the high number of unclassified outliers can significantly disturb the silhouette score, being outliers singularities by definition. It should be noted that the silhouette score could not be an efficient metric to evaluate clustering performance in connectivity-based algorithm like DBSCAN as very distant points could end up being grouped in the same cluster.

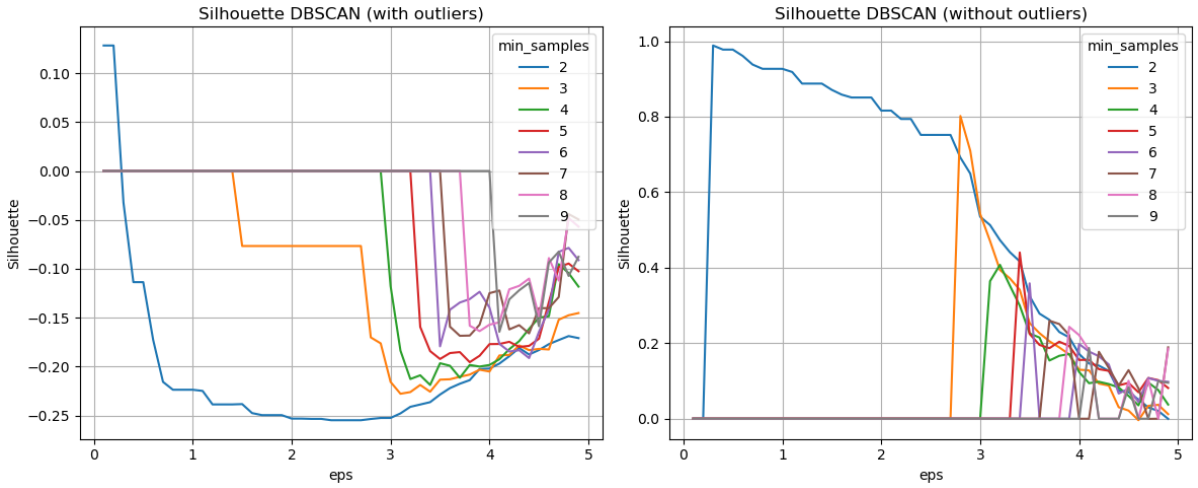


Figure 9: Clustering error and silhouette score for DBSCAN (on PCA-transformed data)

DBSCAN gave us very poor results, identifying with the best set of parameters only two small clusters of 8 and 6 elements respectively and a bigger cluster of 284 elements, with purity of 0.79 (which roughly matches the dataset imbalance ratio). The majority of the samples were classified as outliers.

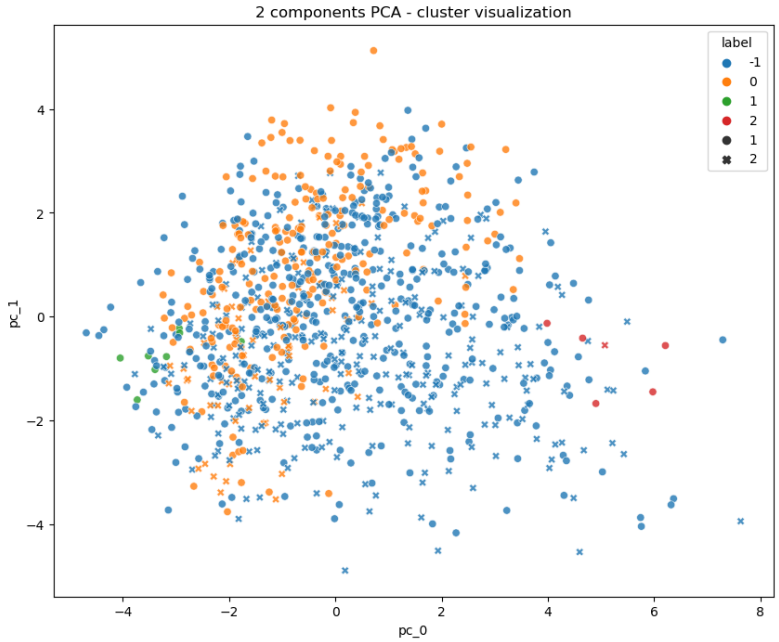


Figure 10: DBSCAN cluster visualization in 2 components (on PCA-transformed data)

Another approach we follow is to perform clustering using non PCA-transformed normalized mixed data with two variants of the previous algorithms:

- k-Prototypes
- DBSCAN with a custom distance metric

K-Prototypes[6] is a variant of the k-Means algorithm that can operate on mixed data, combining the standard k-Means for numerical features and the k-Modes variant for categorical ones. This approach is better suited in this situation as categorical features do not have ordinal value significance, making standard distance measures like Euclidean distance not meaningful. Furthermore, the k-Modes clustering algorithm gives cluster centroids discrete values for categorical values, allowing them to be traced back to the original application domain.

To be able to compute the silhouette score and, later, apply the DBSCAN clustering, we implement the same custom distance metric used internally by the k-Modes library, combining the Euclidean distance (L2-norm) and the Jaccard distance (J) as shown in Equation 1.

$$f(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x}_{\text{num}} - \boldsymbol{y}_{\text{num}}\|_2 + \gamma J(\boldsymbol{x}_{\text{cat}} - \boldsymbol{y}_{\text{cat}})$$
$$\gamma = \frac{1}{c}\sum_{i=1}^{c}\sigma_i; \quad c = |\text{numerical features}| \tag{1}$$

σbeing the standard deviation of each numerical feature

We try to visualize the obtained clusters in 2 dimensions selecting the first two PCA components (Figure 12).

With the silhouette score, we selected 6 clusters but the results showed a very low purity, around 70% of elements with label "1". Since the 70% of the dataset has the label "1" and the remaining 30% has the label "2", it does not identify any meaningful grouping of the data, it only reflects the general distribution of the classes.

We also apply the DBSCAN algorithm to the same mixed min-max scaled data, using as the defined custom distance metric (Equation 1). Clustering performances show very poor also in this setup: two big clusters are identified, with purity of 0.7; 0 outliers were found, probably due to the small value of γ in the distance metric, which led to smaller distances between samples.

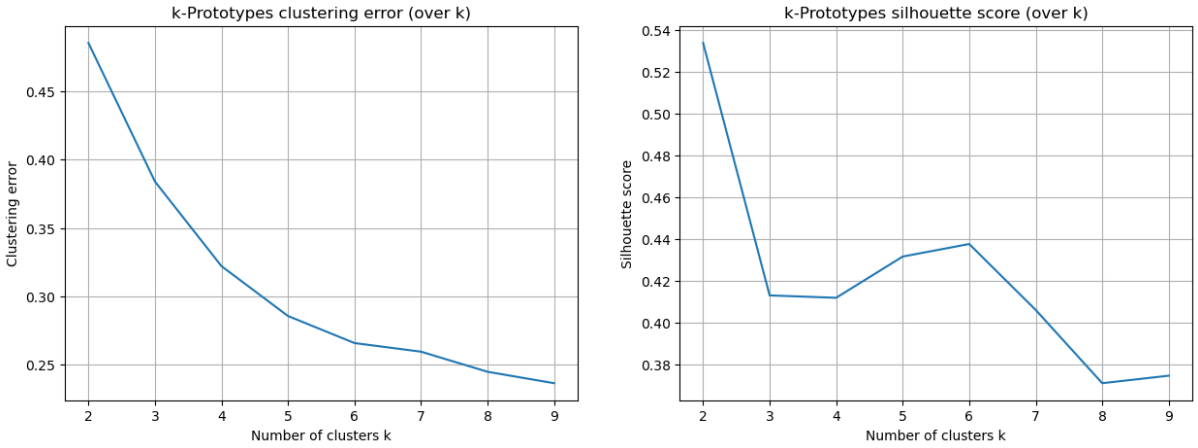


Figure 11: Clustering error and silhouette score for k-Prototypes

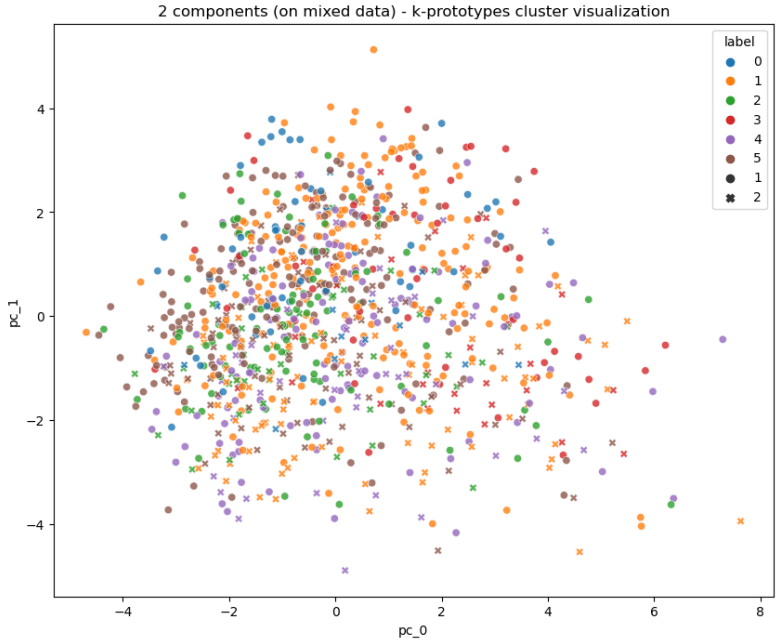


Figure 12: k-Prototypes cluster visualization (in 2 PCA components)

3 Supervised Data Analysis

3.1 Classifier Selection

We experiment with different well-known classifiers to find out which could give better performance in terms of accuracy with our dataset. We try the following to get a first overview of the most promising models:

- Support Vector Machine [SVM] (linear model with regularized Hinge loss)
- Logistic Regression [LR] (linear model with logistic loss)
- Decision Tree [DT] / Random Forest [RF] (non-linear models)
- Nearest Neighbours [kNN]
- Neural Network built with Keras library[NN]

As recommended by the German Credit dataset documentation, we introduce custom class weights in all models that support them in the loss function to both account for dataset imbalance ratio and to focus the attention of the models in reducing the number of bad customers classified as good ones. The following values are used:

Label 1: weight 0.17
Label 2: weight 0.83

We further split the training set previously obtained (80% of the full dataset) into train and validation sets, to be able to assess the model performance and tendency to over-fit or under-fit. We build our neural network with the Keras library as it allows greater control over the model structure compared to the `MLPClassifier` implemented in the Sci-kit Learn library.

We train the models on both the one-hot encoded dataset and the PCA-processed dataset and we verify that there are no significant differences in terms of classification performance. For each model to test we compute the confusion matrix (Figure 13), balanced and unbalanced accuracy scores on the validation set.
We select the following as most promising models to be further investigated and tuned

- Decision Tree
- Support Vector Machine (with linear kernel)
- Neural Network

with respective balanced accuracy of 0.66, 0.74, and 0.77.

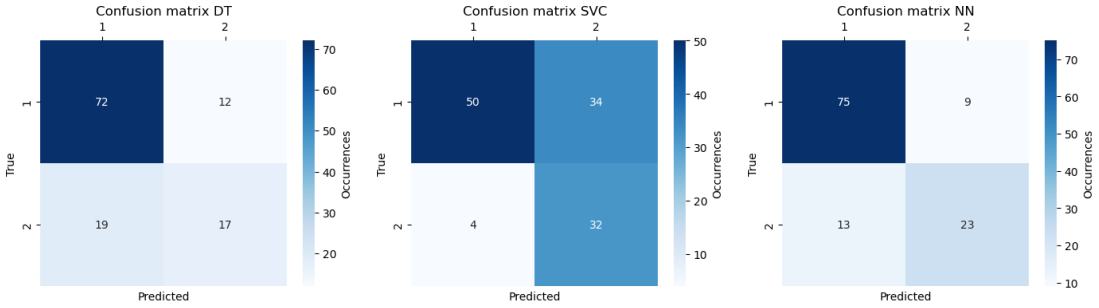


Figure 13: Confusion matrices on the validations set of the most promising models

3.2 Cross Validation and Hyper-parameters Tuning

We setup hyper-parameter tuning and cross-validation on the previously selected models. Cross-validation is performed with a Stratified Shuffle Split strategy, with $K = 5$ for the Neural Network and $K = 10$ for the other models, to respect the class ratio among the splits and keep a reasonable computational time. Being the space of the hyper-parameters relatively small, we preferred a Grid-Search approach for the tuning.

The search space of the Decision Tree includes the two "criterion" available for the sci-kit learn class {"gini", "entropy"}, the depth of the tree and the minimum number of samples needed to split a node; we limit the number of splits to 10 since increasing it would lead the classifier not to learn from the data[7].

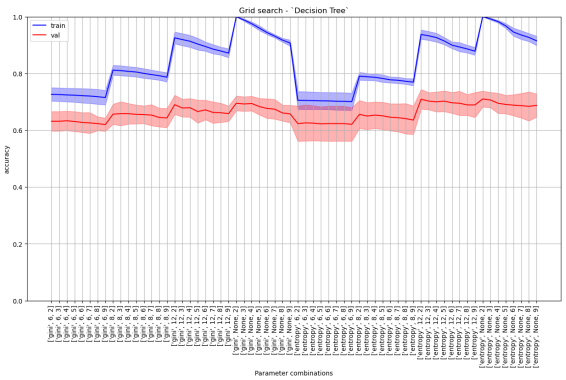


Figure 14: (a) DT Tuning

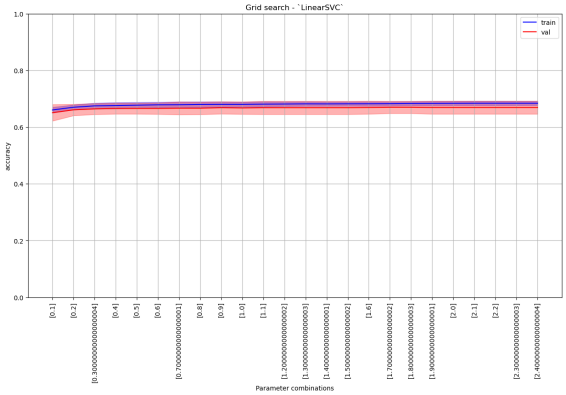


Figure 15: (b) SVC Tuning

The search space of the Support Vector Classifier only includes other regularization parameter C: higher values make the decision boundary more able to vary[8].

To automate the hyper-parameter tuning of our Keras neural network we use the Keras Tuner library, re-implementing the `GridSearch` class since it does not support cross-validation. The search space includes the number of neurons for the two hidden layers. A high value of nodes per layer could lead our model to overfit due to its unnecessary complexity, providing no real improvements on the validation accuracy. We test different activation functions of the hidden layers, such as Rectified Linear Unit (ReLU) and Hyperbolic Tangent (tanh). No meaningful difference in terms of validation accuracy or complexity of the model can be seen, so we finally choose ReLU for the final tuning process given its widespread usage in neural network applications and its advantage of not suffering from the vanishing gradient problem[9].

3.3 Classifier Evaluation

After tuning the hyperparameters we confirm the Neural Network as the best model with the highest balanced accuracy on the validation set. As the neural network outputs a probability, we find the best threshold for the classification leveraging the outputs of the `roc_curve` function to maximize the difference between true positive rate and false positive rate on the training data. Due to the high number of samples with label "Good" our model is better at identifying good loans (85% of the "Good" predictions are actually "Good") rather than at identifying bad loans (58% of the "Bad" predictions are actually "Bad"), also the majority of the "Good" samples were predicted right (recall is equal to 0.79). The recall of the "Bad" samples is 0.67, probably due to the unbalanced dataset. We mitigated this effect by giving a higher weight to the samples of this class over the others, leading to a lower number of false positives. The overall balanced accuracy of our model on the testing dataset was 0.73.

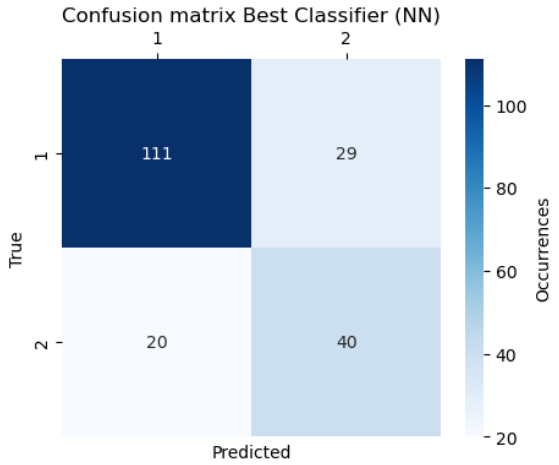


Figure 16: Confusion matrix on testing data

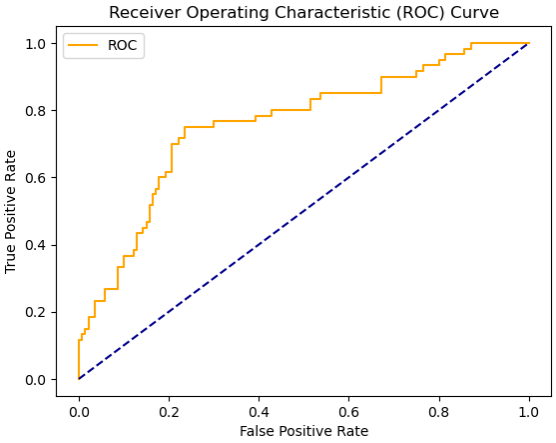


Figure 17: ROC function on testing data

Class	Good	Bad
Precision	0.85	0.58
Recall	0.79	0.67
F1-Score	0.82	0.62
Balanced accuracy	0.73	

Table 1: Final model metrics

4 Adversarial Attacks

As artificial intelligence continues to evolve and its usage keeps spreading in various domains, robustness and security of machine learning models have emerged as critical areas of investigation and focal points for researchers. Model misbehaviour could indeed lead to disastrous outcomes in real world safety-critical applications, such as industrial control systems, healthcare diagnostics and road-sign recognition in autonomous vehicles.
With **adversarial attacks** we refer to a class of techniques aimed at deceiving the trained models in their classification task by using as input noisy or specially crafted data points, defined **adversarial examples**.
This kind of attacks can be mainly classified in two categories based on the knowledge of the target model inner architecture:

- **black-box** attacks
- **white-box** attacks

While black-box attacks only rely on model outputs to generate adversarial data points, white-box attacks leverage the access to the model structure and parameters to produce carefully crafted perturbations in the data and precisely control the model behaviour.

In the following sections, we summarize our experiments with different adversarial techniques applied to the best neural network classifier we obtained previously. We mainly take into account the balanced accuracy score[10] to evaluate model robustness against the attacks and we compare the classification outcomes with the ones presented in section (3.3).

4.1 Random Noise

One fundamental approach in the exploration of neural networks' vulnerability to adversarial attacks involves the introduction of random noise to the initial input data. In this method, small perturbations are added to the input features, aiming to deceive the neural network into making incorrect predictions. This technique is a valuable starting point for evaluating the resilience of machine learning models. However, it is crucial to note a significant limitation associated with this approach, particularly when dealing with datasets containing categorical features. As we are applying continuously distributed noise to each feature, categorical features may assume values that cannot be traced back to the original business domain[2].

Given $\boldsymbol{x}$, we want to generate $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\eta}$, where $\boldsymbol{\eta}$ is a random noise N-vector. To evaluate the model resistance to different noise sizes, we decide to generate the noise vectors with fixed Euclidean norm ϵ.
Considering the noise vector as a N-vector of uncorrelated Gaussian-distributed random variables,

$$\boldsymbol{\eta} = \{\eta_i\}^N, \quad \eta_i \sim N(0, \sigma^2) \tag{2}$$

$$\mathbb{E}\left[\|\boldsymbol{\eta}\|_2^2\right] = \mathbb{E}\left[\sum_i^N \eta_i^2\right] = \epsilon^2 \tag{3}$$

As the variables are uncorrelated, we can write

$$\mathbb{E}\left[\|\boldsymbol{\eta}\|_2^2\right] = \sum_i^N \mathbb{E}\left[\eta_i^2\right] = \sum_i^N \sigma^2 = N\sigma^2 = \epsilon^2 \tag{4}$$

[2]Other solutions could be taken into account to overcome this limitation, such as randomly altering the category value of the features

We can thus generate for each data point a N-vector of samples from a Gaussian distribution[3]

$$N\left(0, \frac{\epsilon^2}{N}\right)$$

The adversarial data generated show the model's high sensitivity to noise, with a remarkable drop of the balanced accuracy as the noise L2-norm ϵ is increased.

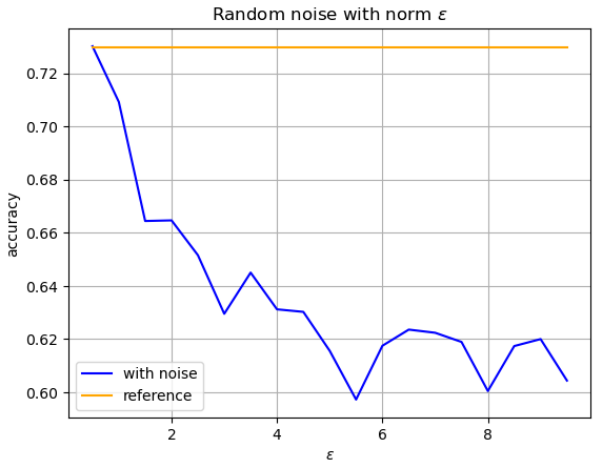


Figure 18: Balanced accuracy of the model after injection of random noise over all feature dimensions

4.2 Feature specific noise

Rather than uniformly applying noise to all features simultaneously, an alternative approach involves applying noise to individual features. This solution allows us to assess the sensitivity of the model to some of the input features.
For a given feature x we generate $x' = x + \eta$ where $\eta \sim N(\epsilon, .001)$. We limit $\epsilon \in [0, .50]$, with increment steps of .05, to avoid noise domination in the final altered feature value.

Plots in figure 19 present the results on some of the features the model showed to be more sensitive on. The high decrease of performance for very low values of noise suggests that the margin of decision boundary of the model across these dimensions is relatively small, thus making these features good targets for an attacker willing to trigger model misbehaviours without heavily affecting the original input data.

4.3 Fast Gradient Sign Method

Described for the first time by Goodfellow *et al.* in 2015 [11], the Fast Gradient Sign Method (FGSM) is a white-box adversarial attack technique that capitalizes on the intimate knowledge of the underlying neural network architecture. Unlike random noise-based methods, FGSM is a targeted approach that strategically manipulates input data to deceive the model into producing incorrect predictions. This method operates on the principle of exploiting the gradient information of the neural network with respect to the input features.

In FGSM, adversaries compute the gradient of the loss function with respect to the input data and use its direction to generate a perturbation vector, thus moving the data into the direction of loss function maximization [formula (5)]. This approach is able to efficiently generate very effective adversarial examples even with small distance to the original input.

[3]We also clip the noise sampled from the Gaussian distribution to be at most limited in the range $[-.5, .5]$ to keep the noise value in an acceptable range accounting the normalization of each feature

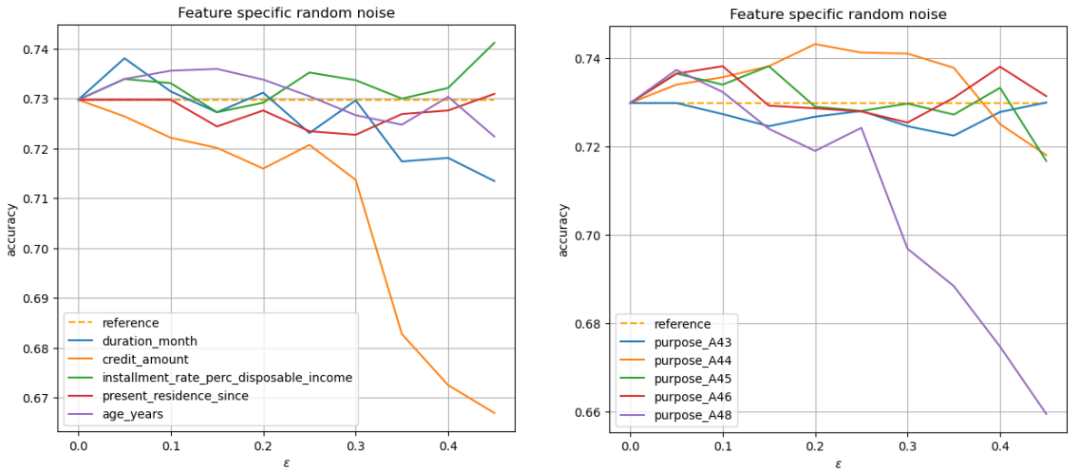
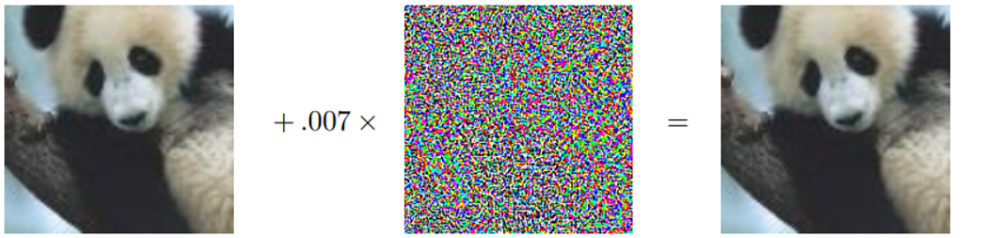


Figure 19: Balanced accuracy of the model after injection of random noise over single feature dimensions

$$\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{\eta}; \qquad \boldsymbol{\eta} = \epsilon \operatorname{sign}\left(\nabla_{\boldsymbol{x}} L\left((\boldsymbol{x}, y), h\right)\right) \tag{5}$$



$\boldsymbol{x}$ $\operatorname{sign}\left(\nabla_{\boldsymbol{x}} L\left((\boldsymbol{x}, y), h\right)\right)$ $\boldsymbol{x} + \epsilon \operatorname{sign}\left(\nabla_{\boldsymbol{x}} L\left((\boldsymbol{x}, y), h\right)\right)$

Figure 20: FGSM applied to image recognition

To mount the attack against our neural network classifier we used the Adversarial Robustness Toolbox (ART)[4] Python library, implementing various optimized attack techniques aimed at evaluating machine learning models security.

We evaluated the effect of the attack on a range of different distance values ϵ [figure 21]. To be able to compare the results to the random-noise based attack presented in section (4.1), we computed the L2-norm of the perturbation vectors applied on each data point generated by the adversary. As the noise introduced over each feature is in the set $\{-\epsilon, +\epsilon\}$, we compute the norm $\|\boldsymbol{\eta}\|_2 = N\epsilon^2$. We observe that the distance between the adversarial examples and the normal inputs required to heavily mislead the model is significantly smaller. As shown in (18) the model accuracy settles down around 0.60 with $\|\boldsymbol{\eta}\|_2 \geq 5.5$ in the random noise setup, while FGSM quickly drops the performance to 0.55 with $\|\boldsymbol{\eta}\|_2 \geq 0.5$.
As smaller but carefully crafted perturbations can be applied on the input data, FGSM-based attacks become much harder to detect in live environments.

In figure 22 we show the classification outcomes of a set of adversarial examples generated from the original test set with $\epsilon = 0.1$. We clearly see that the model is completely biased in predicting all the data points as class 1, with an f1-score of class 2 of 0.10.

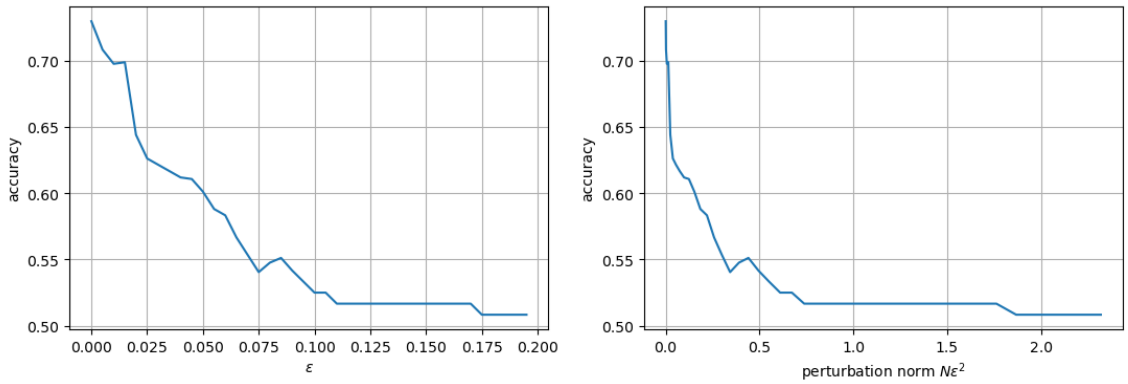


Figure 21: Balanced accuracy of the model with different ϵ distance values on FGSM

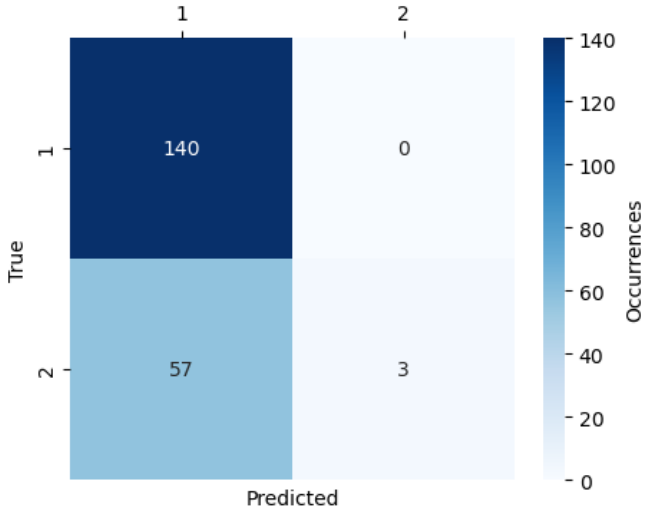


Figure 22: Confusion matrix on FGSM adversarial test, $\epsilon = 0.1$

4.4 Boundary Attack

Presented at ICLR 2018 by Brendel *et al.*[12], the Boundary Attack is a decision-based black-box adversarial attack. As the attack requires no knowledge outside of the model final output labels, it becomes actually applicable to real-world safety-critical environments where we do not usually get access to the model structure. The attack comes in two variants,

- *untargeted*, where the adversarial examples are generated to generically decrease the classification performance of the model
- *targeted*, where the computed perturbations are crafted by aiming at maximising the model probability of classifying the input with a target output class

The generation of adversarial examples follows an iterative approach (Figure 23) to move the generated samples towards the region of the input space which allows the smallest amount of perturbation to trigger a misclassification. At each iteration the generated sample randomly moves

1. over a projected sphere around the original input
2. towards the original input

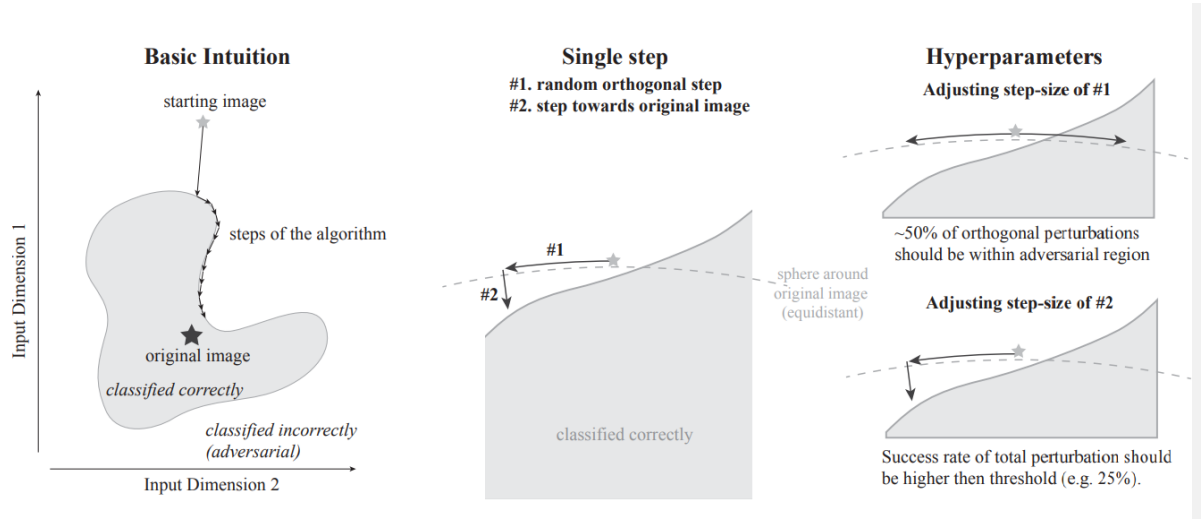
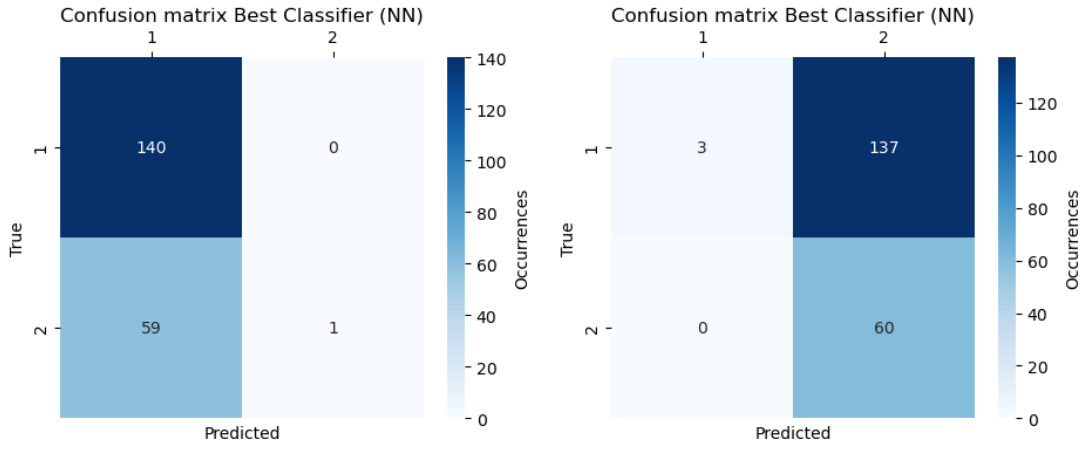


Figure 23: Boundary Attack local decision boundary approximation step

Evaluating the classification rate of the adversarial sample w.r.t. the original input classification outcome a local approximation of the decision boundary is reconstructed and at the next iteration the generated samples will move in the optimal direction.

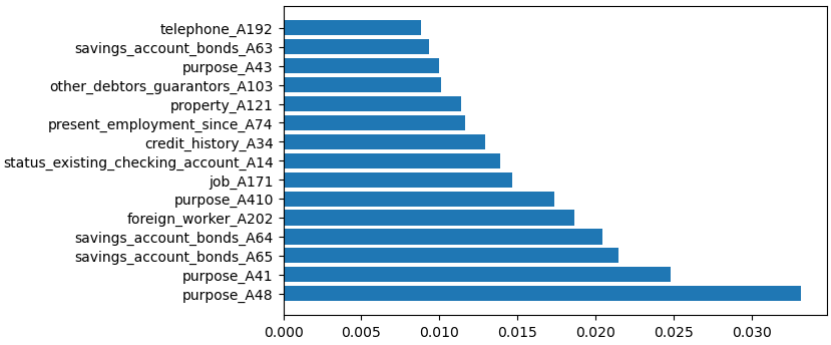
Being FGSM already an untargeted type of adversarial attack, we decided to test the targeted variant of the Boundary Attack and check the amount of noise applied to input data to specifically target each of our output classes. We display in Figure 24 the confusion matrices of the model predictions on the adversarial examples generated targeting each of the class. In Figure 25 we show the top-15 features per amount of average noise applied in the generated adversarial examples. We want to highlight that the most influenced features match the ones already detected previously applying random noise on individual features (Figure 19).



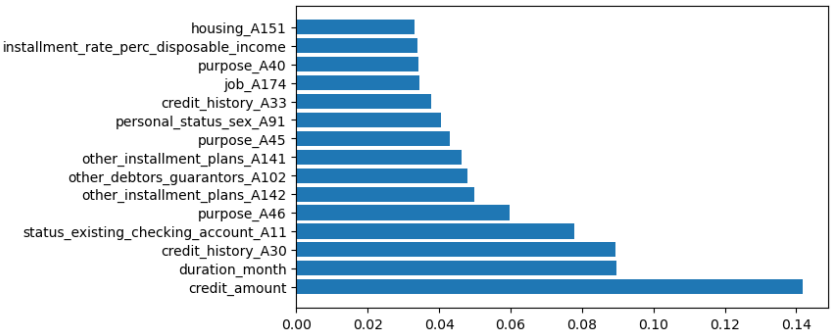
(a) Target class 1 (b) Target class 2

Figure 24: Confusion matrices on Boundary Attack generated adversarial examples

The Boundary Attack stands out as a noteworthy technique in the evaluation of machine learning model robustness, particularly due to its distinctive attributes that contribute to its effectiveness. One key advantage lies in its independence from access to the internal decision process of the model. It's decision-based nature makes it easily applicable to scenarios where the inner workings of the model are proprietary or not fully accessible. Another key advantage is its simplicity and efficacy in generating adversarial examples without resorting to intricate



(a) Target class 1



(b) Target class 2

Figure 25: Top-15 features per amount of average noise applied with Boundary Attack

or computationally demanding procedures. This makes it a pragmatic choice for assessing model resilience in real-world applications, where the simplicity of attack methods can often mimic the strategies employed by potential adversaries with limited resources or expertise. Moreover, the Boundary Attack's capability to target specific output classes introduces a notable security concern, particularly in environments where the various classes hold different levels of importance within the business domain. This targeted adversarial capability accentuates the potential impact of an attack, as adversaries can strategically manipulate the model to misclassify instances into predefined high-impact output classes.

4.5 Countermeasure exploration

One of the most explored countermeasures to adversarial attacks in literature is **adversarial training**. Adversarial training represents a proactive strategy that involves augmenting the training dataset with adversarial examples. By exposing the model to deliberately crafted perturbations during the training process, the model should learn to become more robust and resilient to adversarial attacks, possibly enhancing its ability to generalize and accurately classify both benign and adversarial inputs.

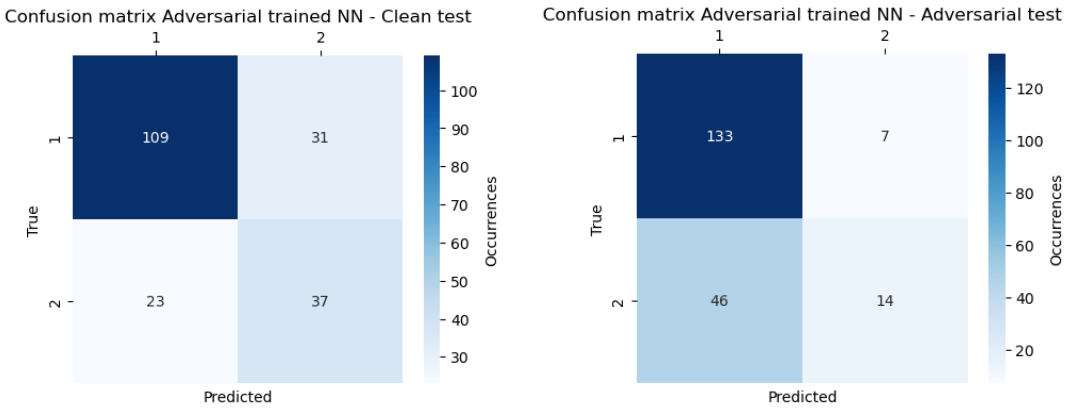
A basic adversarial training setup requires a two-step process:

1. A first model (*Model A*) is trained on the clean training set and tested on the original test set. *Model A* is then used to generate two sets of adversarial examples (*adv_A_S1*, *adv_A_S2*)

2. A second model (*Model B*) is trained on the original clean training set augmented with the adversarial set *adv_A_S1* and its performances are evaluated on both the clean test and the previously generated *adv_A_S2* adversarial set

As shown in 2019 by Qin *et al.*[13], this simple setup does not guarantee that the adversarially trained model at the second step will be robust against adversarial examples further generated by itself, thus not actually preventing possible attacks by adversaries that have access to the final model ("*Round Gap Of Adversarial Training*", RGOAT). A reasonable approach could be to iteratively apply the same schema, i.e. train a model on clean data augmented with previously generated adversarial examples and use it to generate new adversarial train and test sets for the next rounds, possibly randomly rotating the adversarial techniques used to generate the "malicious" examples (e.g. Fast Gradient Sign Method, Jacobian-based Saliency Maps Attacks[14], Basic Iterative Method[15], Carlini&WagnerL2[16]). Anyhow, as shown by Qin *et al.*, increasing the number of rounds of adversarial training does not solve the RGOAT problem, thus showing the limitations of this type of countermeasure.

We use the ART Python library[4] to setup an adversarial training pipeline against our neural network model (only against the FGSM technique showed in Section 4.3). Our best-trained model (Section 3.3) is used to generate adversarial examples to augment the training dataset and later test the adversarially trained model. We observe that the new trained model (*Model B*) is slightly sacrificing performance in classifying the original test data (0.70 balanced accuracy against 0.73) while not much increasing its robustness against the adversarial examples (0.59 balanced accuracy against 0.53) (Figure 26), probably due to the small capacity of the selected network preventing it from learning from the adversarial examples while showing good performances in the clean-training stage[17]. The process could be iterated to assess for possible enhancements, keeping in mind that the final trained model will eventually still be vulnerable to self generated adversarial examples (RGOAT problem[13]), as shown in Figure 27.



(a) clean data (b) *Model A* adversarial data

Figure 26: Classification outcomes of the first-round adversarially trained model

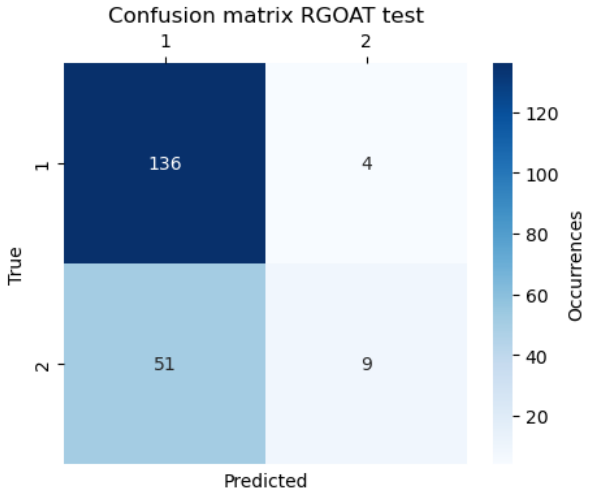


Figure 27: Classification outcomes of *Model B* over (self)generated adversarial examples

References

[1] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[2] Nelis J. de Vos. *kmodes categorical clustering library*. `https://github.com/nicodv/kmodes`. 2015.

[3] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[4] Maria-Irina Nicolae et al. *Adversarial Robustness Toolbox v1.0.0*. 2019. arXiv: `1807.01069 [cs.LG]`.

[5] *Importance of Feature Scaling*. URL: `https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html`.

[6] Z. Huang. "Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values 283–304 (1998). https://doi.org/10.1023/A:1009769707641". In: *Data Mining and Knowledge Discovery 2* (1998), pp. 283–304. DOI: `10.1023/A:1009769707641`. URL: `https://doi.org/10.1023/A:1009769707641`.

[7] *Decision Trees*. URL: `https://scikit-learn.org/1.0/modules/tree.html`.

[8] *Support Vector Machine Classifier*. URL: `https://scikit-learn.org/1.0/modules/svm.html`.

[9] Luca Vassio, Tailai Song, and Gabriele Ciravegna. "Machine Learning for Networkings". L11-NeuralNetworks.pdf presentation, slide 39.

[10] *Balanced accuracy score*. URL: `https://scikit-learn.org/1.0/modules/generated/sklearn.metrics.balanced_accuracy_score.html`.

[11] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: `1412.6572 [stat.ML]`.

[12] Wieland Brendel, Jonas Rauber, and Matthias Bethge. *Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models*. 2018. arXiv: `1712.04248 [stat.ML]`.

[13] Yi Qin, Ryan Hunt, and Chuan Yue. "On Improving the Effectiveness of Adversarial Training". In: *Proceedings of the ACM International Workshop on Security and Privacy Analytics*. IWSPA '19. Richardson, Texas, USA: Association for Computing Machinery, 2019, pp. 5–13. ISBN: 9781450361781. DOI: `10.1145/3309182.3309190`. URL: `https://doi.org/10.1145/3309182.3309190`.

[14] Nicolas Papernot et al. *The Limitations of Deep Learning in Adversarial Settings*. 2015. arXiv: `1511.07528 [cs.CR]`.

[15] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. *Adversarial examples in the physical world*. 2017. arXiv: `1607.02533 [cs.CV]`.

[16] Nicholas Carlini and David Wagner. *Towards Evaluating the Robustness of Neural Networks*. 2017. arXiv: `1608.04644 [cs.CR]`.

[17] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: `1706.06083 [stat.ML]`.