CREDIT RISK

Presentors:

Elad Sapir

Tomer Raitsis

Solal Ohana

Lecturer: Dr. Tammar Shrot

Table of contents

- **Abstract 3**
- **Introduction.....4**
- **Review of the selected methods.....5**
- **A Review of the Development 8**
  - **i. Graphical representation 12**
  - **ii. The AUC&ROC graphical representation 13**
- **Our dilemmas 14**
  - **i. KNN 14**
  - **ii. SVC 16**
- **Operating instructions.....17**
- **Comparison between the algorithms 18**
- **Summary of our studies 20**
- **Bibliography 22**

Abstract

In this work, we investigate the use of K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) for credit risk assessment using a dataset of about 2000 individuals from a German bank. The dataset includes information such as income, age, job, and risk.

Our results show that both KNN and SVM are effective in predicting credit risk, with KNN achieving an accuracy of 90.1857% and SVM achieving an accuracy 92.5729%.

Overall, our work highlights the potential of KNN and SVM for credit risk assessment and demonstrates the importance of considering multiple machine learning algorithms in this task.

The results of this work may have the potential to enhance the current loan application process performed by human resources and also decrease the mistakes that may happen due to factors such as lack of knowledge, fatigue and others.

Introduction

Financial credit risk assessment is a crucial aspect of accounting and finance that involves predicting the likelihood of business failure and assessing the potential impacts on the economy and society.
Accurate risk assessment is essential for making informed financial decisions and mitigating risk.
Despite the numerous efforts that have been made in this field, it remains a challenging task due to the complexity of the financial system and the potential for unforeseen events to impact business performance.

Using AI to solve credit risk problems can improve the accuracy and efficiency of credit risk assessments, enhance risk management capabilities, and increase accessibility to credit. Machine learning algorithms can more accurately predict the likelihood of default for a given borrower, and automating the credit risk assessment process with AI can significantly reduce the time and resources required to evaluate applications. By continuously learning from past data, an AI model can help identify patterns and trends that may not be apparent using traditional methods, allowing for more informed risk management decisions.

Why We Chose These Approaches for Solving the Problem: A Review of the Selected Methods

One of the approaches for this problem (Credit Risk) is Classification. Classification methods are a type of machine learning algorithm that can be used to predict the class or category of a given data point based on its features. In the context of financial credit risk assessment, classification algorithms can be used to predict the likelihood of business failure based on historical data.

To use classification methods for this task, a dataset of historical data is first collected and labeled with the outcome of interest.
This dataset is then used to train the classification algorithm. During training, the algorithm learns to identify patterns and relationships in the data that are indicative of risk. Once trained, the classification algorithm can be used to predict the likelihood of failure for a new business based on its features.

There are many different classification algorithms that can be used for financial credit risk assessment, including K-Nearest Neighbors (KNN), Support Vector Machines (SVMs), Decision Trees, and Logistic Regression.

In general, classification methods are a useful tool for financial credit risk assessment because they can learn to identify patterns and relationships in the data that are indicative of risk. This can help to improve the accuracy of risk prediction and enable more informed decision-making. Some of the benefits of using classification algorithms for financial credit risk assessment include:

Automation: Classification algorithms can be used to automate the risk assessment process, making it faster and more efficient.

Scalability: Classification algorithms can handle large datasets, making them well-suited for tasks involving a large number of businesses.

Accuracy: With the right training data and parameters, classification algorithms can achieve high levels of accuracy in predicting business failure.

Flexibility: Classification algorithms can be used to predict risk for a wide range of businesses in different industries and economic conditions.

Overall, classification methods are a powerful tool for financial credit risk assessment and can help to improve prediction accuracy and enable more informed decision-making.

That being said, K-Nearest Neighbors (KNN) and Support Vector Classification (SVC) are two algorithms that are often used for this purpose and have been shown to be effective in many applications.

KNN is a non-parametric method that works by identifying the K data points in a dataset that are most similar to a given data point and using those points to make a prediction. One of the main advantages of KNN is that it is simple to implement and can be easily adjusted to handle new data. It also has the ability to handle multi-dimensional data, which is often the case in financial credit risk assessment.

SVC stands for Support Vector Classification, which is a type of SVM (Support Vector Machine). SVM is a supervised machine learning algorithm that can be used for classification, regression, and other tasks.

In addition, SVMs have a number of unique properties that make them well-suited for this task, such as their ability to handle large datasets efficiently and their robustness to noise and uncertainty.
SVC is a specific variant of SVM that is used for classification tasks, where the goal is to predict the class (e.g., creditworthy or not creditworthy) of a given data point based on its features.

In SVC, the goal is to find the hyperplane in a high-dimensional space that maximally separates different classes of data. The data points closest to the hyperplane are called support vectors and have the

greatest impact on the position of the hyperplane. SVC uses these support vectors to make predictions about the class of new data points.

Overall, KNN and SVCs are two algorithms that are often used for financial credit risk assessment and have been shown to be effective in many applications.

Presenting the Code:

A Review of the Development Process and Challenges Encountered

Our work began with the task of preprocessing the data:

```
df = pd.read_csv("Database.csv", index_col=0)

for col in ['Saving accounts', 'Checking account']:
    df[col].fillna('none', inplace=True)
j = {0: 'unskilled and non-res', 1: 'unskilled and res', 2: 'skilled', 3: 'highly skilled'}
df['Job'] = df['Job'].map(j)

# encoding risk as binary
r = {"good": 0, "bad": 1}
df['Risk'] = df['Risk'].map(r)

# getting dummies for all the categorical variables
dummies_columns = ['Job', 'Purpose', 'Sex', 'Housing', 'Saving accounts', 'Checking account']
for col in dummies_columns:
    df = df.merge(pd.get_dummies(df[col], drop_first=True, prefix=str(col)), left_index=True, right_index=True)

# drop redundant variables
columns_to_drop = ['Job', 'Purpose', 'Sex', 'Housing', 'Saving accounts', 'Checking account']
df.drop(columns_to_drop, axis=1, inplace=True)
df['Log_CA'] = np.log(df['Credit amount'])

X = df.drop(['Risk', 'Credit amount'], axis=1).values
y = df['Risk'].values
```

- We started with transferring the literal data into numeric data, we transferred 'Risk' into binary values (0,1), for the rest of the literal data we used a technique called "one-hot encoding".
One-hot encoding is a way of representing categorical data numerically so that it can be used in machine learning algorithms.
It works by creating a new column for each unique category in the categorical column and then encoding the values in the original column as 1s and 0s in the new columns (dummies columns).

- After preprocessing the data we separated it into X and y, X contains all the parameters that we are trying to train the machine with, and y is the parameter that we try to teach the machine to find by itself.

The next step was dividing it into a training set and a testing set:

```
# # train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

"random state = 42" is a specific value that is used as a seed for the pseudorandom number generator in certain computer programs.
By setting the random state to a specific value, the program will always produce the same sequence of random numbers, which can be useful for reproducibility in scientific experiments or debugging.

Fun fact :

The number 42 is often used as a default value because it is a nod to the science fiction book "The Hitchhiker's Guide to the Galaxy" where the number 42 is "The Answer to the Ultimate Question of Life, the Universe, and Everything" according to a supercomputer.

Then we ran our KNN algorithm on the data and assessed its accuracy. To conclude our analysis, also printed the confusion matrix to visualize the performance of a classification model.

It allows you to see the number of correct and incorrect predictions made by the model.

```python
# finding the best k value for the KNN algorithm
print('Looking For the Best K...')
max_score = 0
max_k = 0
knn_max = 1
for k in range(1, 100):
    knn = KNeighborsClassifier(n_neighbors=k, metric='manhattan')
    knn.fit(X_train, y_train)
    score = accuracy_score(knn.predict(X_test), y_test)
    if score > max_score:
        max_k = k
        max_score = score
        knn_max = knn
print('The best K is ' + str(max_k) + '\n')

print('Predicting and Calculating the Precision (Using KNN)...')
knn = knn_max
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print('KNN Precision: ' + str(round(accuracy_score(y_pred_knn, y_test) * 100, 4)) + ' %\n')

# confusion matrix calculation
confusion_mat_KNN = confusion_matrix(y_test, y_pred_knn)
print('KNN Confusion Matrix:\n')
# Convert confusion matrix into a Pandas dataframe
df_cm = pd.DataFrame(confusion_mat_KNN, index=['True:', 'False:'], columns=['Predicted:', 'Not Predicted:'])
# Display the confusion matrix as a table
display(df_cm)
print('\n')
```

```
KNN Precision: 90.1857 %

KNN Confusion Matrix:

       Predicted:  Not Predicted:
True:         245              18
False:         19              95
```

Then we also ran our SVC algorithm on the data and assessed its accuracy and printed its confusion matrix.

```
print('Searching for the Best Parameter Values for SVC...')
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
svc = SVC()
grid_search = GridSearchCV(svc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters found by the grid search
print('The best values for "C" and "gamma" are: ' + str(grid_search.best_params_) + '\n')

bestVals = list((grid_search.best_params_).values())

print('Predicting and Calculating the Precision (Using SVC)...')
svc = SVC(C= bestVals[0], gamma= bestVals[1], kernel='rbf') ## the default kernel is 'rbf'
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)
ans1 = accuracy_score(y_pred_svc, y_test)

print('SVC Precision: ' + str(round(accuracy_score(y_pred_svc, y_test) * 100, 4)) + ' %\n')

# confusion matrix calculation
confusion_mat_SVC = confusion_matrix(y_test, y_pred_svc)
print('SVC Confusion Matrix:\n')
# Convert confusion matrix into a Pandas dataframe
df_cm = pd.DataFrame(confusion_mat_SVC, index=['True:', 'False:'], columns=['Predicted:', 'Not Predicted:'])
# Display the confusion matrix as a table
display(df_cm)
print('\n')
```

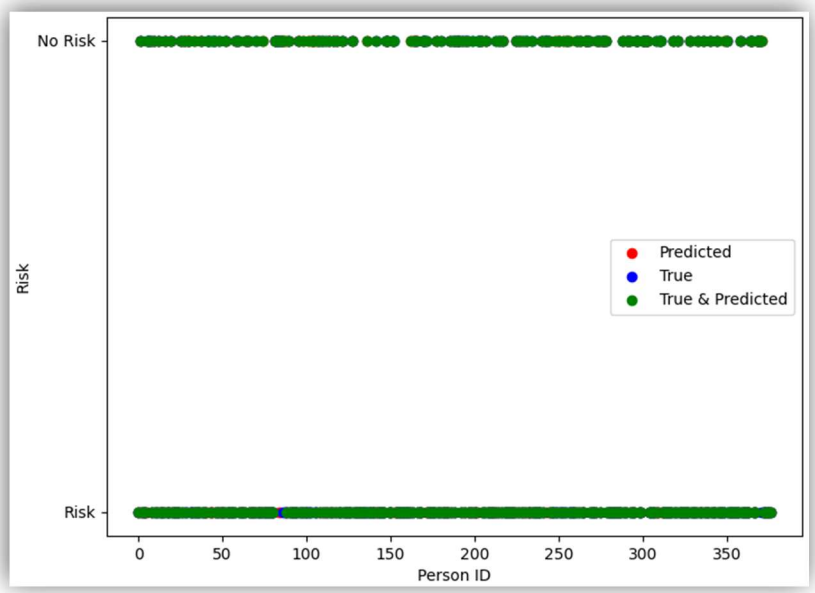
```
SVC Precision: 92.5729 %

SVC Confusion Matrix:

        Predicted:  Not Predicted:
True:          261               2
False:          26              88
```

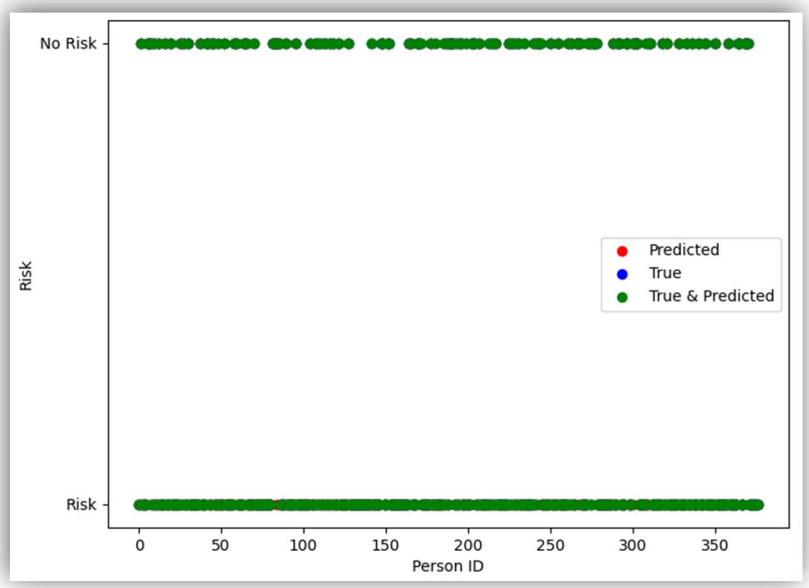
We presented the results through the use of graphical representations.

The purpose of graphic representation is to make it easier to understand and interpret data (will be detailed below).

- It's highly recommended to try it yourself and zoom into the graphs to understand them and see more details.

KNN & SVC graphical representation:





X-axis: Represents Person ID.

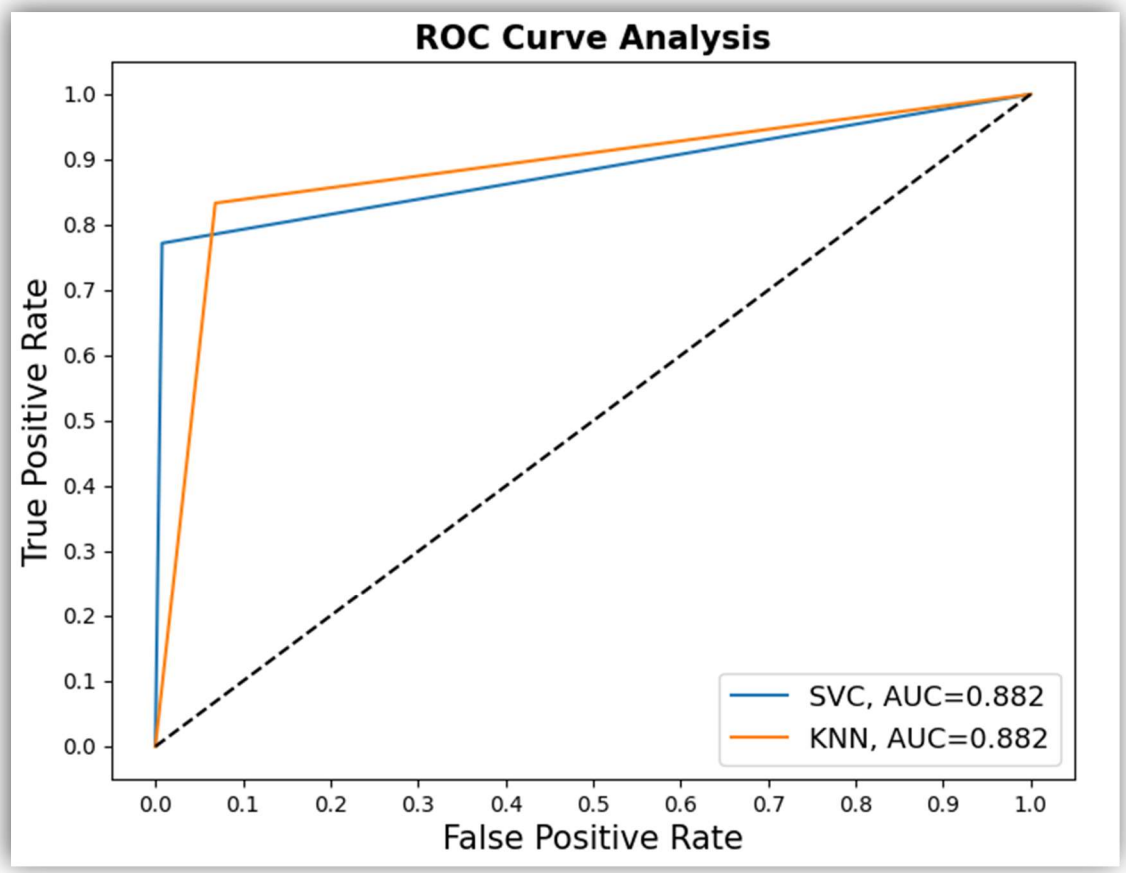
Y-axis: Represents Risk as a binary value (Risk / No Risk).

- Red – False prediction.
- Blue – True (actual value).
- Green – True prediction.

The AUC & ROC graph for both algorithms:

A Receiver Operating Characteristic (ROC) curve is a plot of the true positive rate (sensitivity) against the false positive rate (1 - specificity) for a binary classification problem.

The area under the curve (AUC) is a measure of the ability of the model to correctly discriminate between positive and negative classes. An AUC of 1 indicates perfect discrimination, while an AUC of 0.5 indicates a model that is no better than random guessing.



- We will discuss the conclusion of the system and specific on the graphs in the next chapters.

Our Dilemmas

In our search for a suitable database, we prioritized finding one that was comprehensive and contained all the data needed for running algorithms. However, a limitation we encountered was that many of the databases we reviewed were insufficient in terms of the results of loan requests.

In order to effectively solve our problem, we needed to identify algorithms that would yield the highest success rates. After careful consideration, we determined that classification methods would be the most suitable due to the binary nature of our results, we also based our decision on previous research on this subject.

One of the significant challenges we faced was selecting the most suitable parameters for our functions, with the aim of achieving the highest success rates:

KNN:

The final accuracy of the algorithm is 90.1857%.

Our steps were:

- Running the algorithm as it is (with the default parameters) is 68.7003%.
- Find the best k (number of neighbors that will bring the best accuracy between 1 - 100).
  we found that most of the results are around 70% and the best one was k=1 and its accuracy is 89.125%.
  We decided to keep the loop that finds the best k and to use it in each run of the algorithm in case there will be a better k using other data sets.
- Find the best 'metric' parameter:
  Manhattan distance is the sum of the absolute differences of coordinates in the n-dimensional space, yielding maximum accuracy of 90.18%.

- Chebyshev distance is the maximum absolute difference of coordinates, yielding maximum accuracy of 79.31%.
- Euclidean distance is the square root of the sum of squares of differences of coordinates in the n-dimensional space, yielding maximum accuracy of 89.12%.
- The Minkowski distance is a generalization of the Euclidean distance and Manhattan distance. It is defined as the "p-norm" distance between two points in a Euclidean space, where p is a positive integer.
  Minkowski distance with p=1 is Manhattan distance.
  Minkowski distance with p=2 is Euclidean distance
  Minkowski distance with p>2 generalizations of Euclidean distance (running on p=4,5,6,7,8 – yielded around 89%).
  We decided to use the Manhattan distance as our 'metric' parameter because it yielded maximum accuracy of 90.18%.

- Find the best 'algorithm' parameter:
  the "algorithm" parameter is used to specify the algorithm used to compute the nearest neighbors, the options are k-d tree, ball tree and brute.
  A k-d tree is a space-partitioning data structure that recursively divides the space to efficiently find the nearest neighbors in a k-dimensional space.
  A ball tree is a binary tree structure where each internal node represents a hyper-spherical region in the feature space and is used to efficiently find nearest neighbors in large datasets with high dimensional feature space.
  "brute" computes the distance between all pairs of points in the dataset, which can be computationally expensive for large datasets.
  None of these parameters improved our accuracy so we let the algorithm choose itself with the option of "auto" (default).

SVC:

The final accuracy of the algorithm is 92.5729%.

Our steps were:

- Running the algorithm as it is (with the default parameters) is 69.7612%.
- The algorithm can run with 3 different functions ('rbf', 'linear', 'sigmoid') in the parameter 'kernel', for each function we searched for the best value of the parameter 'C' that will bring the highest accuracy overall ("C" controls the trade-off between maximizing the margin and minimizing the classification error), we found out that the highest accuracy yielded with 'rbf' and 'C'=1000 (yielded 79.31% accuracy).

  We let the greedy search run for every execution of the program to find the value of 'C' that will bring the highest accuracy.
- Since we found out that 'rbf' yields the highest accuracy we searched for the best 'gamma' parameter ("gamma" controls the width of the RBF kernel used in the SVC algorithm. A larger gamma value results in a narrower and more complex decision boundary, while a smaller gamma value results in a wider and simpler decision boundary):

  We used a greedy algorithm that chooses the value of 'gamma' (in addition to finding the best 'C') that yielded the highest accuracy. (yielded 92.5729% with 'C' =10 and 'gamma'=1).

Operating Instructions of the system

- The first step is to make sure that you have the database file (.csv) in the same folder of the project.
- All that is left to do is to run the code, you can see the actual results in the terminal (for each algorithm the program prints the accuracy percentages and the confusion matrix), and after it finishes calculating it represents the graphs as shown above.
- In each of the graph windows that opened, there are functionalities such as zooming in to see a better picture, after both of the algorithms graphs, the window of the AUC/ROC graph will be presented as shown above.

  *every time that you want to see the next graph you need to close the previous one.

Comparison between the algorithms

A good way to start comparing the two algorithms is by looking at the ROC curve.

It is clear that the KNN algorithm performed better than the SVC algorithm, this is due to the fact that KNN provides higher true positive rates at lower false positive rates than the SVC algorithm.
The fact that the AUC values are so close, 0.882446 for KNN and 0.882162 for SVC, suggests that both algorithms have very similar overall performance, the AUC is a measure of how well the model is doing overall.
In conclusion, the KNN algorithm is performing better than the SVC algorithm in terms of correct classification with a lower rate of false positives, but both have a very good overall performance according to AUC.

To compare the algorithms more specifically, we will discuss on some important parameters:

Number of parameters: The KNN algorithm has a relatively low number of parameters to adjust, with only the number of nearest neighbors (k) being the main parameter. SVC has more parameters to adjust, such as the kernel type, C and gamma.
Training time: The KNN algorithm is relatively faster to train, especially when working with large datasets, since it only needs to store the training data and doesn't require any complex computations. SVC can be slower to train, especially when working with large datasets and/or complex kernel types, as it needs to find the optimal hyperplane that separates the different classes which can be computationally intensive.

Memory usage: The KNN algorithm is relatively memory-efficient, as it only needs to store the training data. SVC requires more memory, because it also needs to store the model parameters.

Accuracy: The KNN algorithm can be less accurate than SVC, especially when working with high-dimensional data, since it may be difficult to find k-nearest points that are truly representative of the input in high-dimensional space. SVC can be more accurate, especially when working with non-linear data, as it can find a hyperplane that separates the different classes.
From the confusion matrices, it appears that the KNN model has a higher number of false positives and a lower number of true compared to the SVC model.
On the other hand, the SVC model has a higher number of true positives and a lower number of false positives compared to the KNN algorithm.

Type of problem: The type of problem in this code is classification, as both KNN and SVC are being used to predict labels for given inputs.

Overall it seems that KNN is faster, memory efficient and has less parameters to adjust means that it will be less complicated to work with.
In the matter of accuracy, SVC is more accurate than KNN and will provide better results.
These differences can be crucial (for example a big dataset of 1million loan requests) the difference between the accuracy (2.38%) between the two algorithms can cause false predictions of 23800 loan requests.
Additionally, it's worth considering the runtime of SVC as another factor to weigh when making a decision on which algorithm to use.

Summary of Our Studies

We have gained a comprehensive understanding of the various algorithms through our studies.

One key insight we have acquired is the importance of preprocessing data in order for algorithms to effectively utilize all available information.

This includes tasks such as cleaning, normalizing, and formatting data in a way that is compatible with the specific algorithm being used.

By preprocessing data, we can ensure that the algorithm is able to give relevant results using all the existing data, rather than being limited by inconsistencies or inaccuracies in the input.

Another crucial aspect we have learned is the significance of algorithm parameters and their impact on results and accuracy. These parameters are often adjustable settings within the algorithm that can be adjusted to optimize performance.

Through our studies, we have come to understand that the choice of these parameters can have a significant influence on the results and accuracy of the algorithm.

Therefore, it is crucial to have a good understanding of the parameters and their effects in order to select the best values for a given problem.

Furthermore, we have learned various methods to find the optimal values for the parameters of functions in a greedy way and we have become more knowledgeable in utilizing different tools such as libraries such as 'pandas' to manipulate and process data, 'matplotlib' to visualize data and 'sklearn' for the usage of AI algorithms.

We have learned about the use of a confusion matrix to define the performance of a classification algorithm and the use of ROC curve to represent the performance of a binary classifier in a graphical way.

Through our studies, we have also come to understand that a very high accuracy may not always be indicative of an optimal solution and that a "too good" performance may not be as good as it seems (over fitting).

AI systems are typically trained using large amounts of data and advanced algorithms that allow them to identify patterns and make predictions.

The aim of AI research is to create systems that can learn, solve problems, and make decisions in a way that is similar to human intelligence. Through this study, I have come to understand the significance of preparing data before using it, the ways to choose the algorithm that will yield the best results and fit the data, and the methods to identify and extract important information from data sets.

Bibliography

- Ning Chen, Bernardete Ribeiro3, An Chen. "Financial credit risk assessment: a recent review" Published online: 27 October 2015
- Jorma Laaksonen and Erkki Oja. "Classification with Learning k-Nearest Neighbors"
- Aida Krichene Abdelmoula1. "Bank credit risk analysis with k-nearestneighbor classifier: Case of Tunisian banks"

https://www.kaggle.com/ - (link)

Our repository on Github – code with full report on it: link