IML
THA1 - Submission

Umut Ucak, Lotfi El Ouazizi, Iago Charles Lacroix
Group 8

October 30, 2023

1 A. Supervised Learning (SL) - Regression

1.1 A.I. Linear Regression

1.1.1 Data acquisition

X	Y
23.77	445.72
14.43	466.58
28.25	436.99
12.26	470.87
14.55	457.77
24.57	448.15
10.25	476.03
21.96	442.87
9.82	477.93
18.58	459.06
20.57	456.18
8.61	473.62
9.99	473.64
13.97	463.64
16.76	462.44
25.58	454.39
17.34	452.06
19.13	453.31
12.75	466.93
29.46	436.37

1.1.2 Data transformation

We used **Min-Max Normalization**. We chose this over other methods because of implementation simplicity.

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{1}$$

1.1.3 Least Squares

Model parameters: -0.9329765967793675 0.9415758227757707

X	Y
0.7270983213429256	0.22497593840231045
0.27913669064748203	0.7269008662175163
0.9419664268585132	0.014918190567853814
0.1750599520383693	0.8301251203079884
0.28489208633093527	0.5149181905678532
0.7654676258992805	0.28344562078921975
0.0786570743405276	0.9542829643888346
0.6402877697841727	0.15640038498556302
0.05803357314148685	1.0
0.47817745803357303	0.5459576515880654
0.573621103117506	0.476660250240616
0.0	0.8962945139557266
0.06618705035971226	0.8967757459095279
0.257074340527578	0.6561597690086617
0.3908872901678658	0.6272858517805581
0.8139088729016786	0.43358999037536045
0.418705035971223	0.377526467757459
0.5045563549160671	0.4076034648700673
0.19856115107913672	0.7353224254090471
1.0	0.0

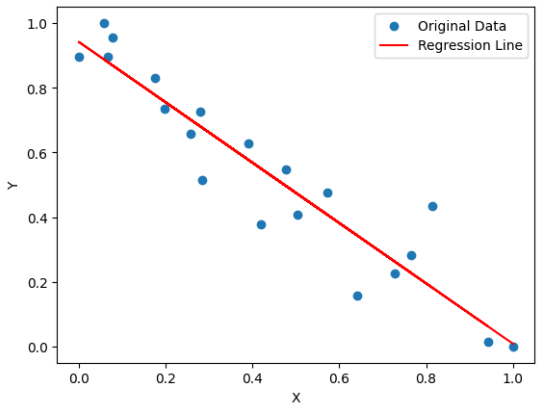

Figure 1: least squares regression line and data points

1.1.4 LR with GD - cost function

We chose the **Mean Squared Error (MSE)**. We used this because of simplicity to implement.

- Model parameters: 0.77645795, 0.79261957
- Cost: 0.06155010691609599
- Gradient: 0.58139903 0.40516739
- Updated model parameters: 0.0581399, 0.04051674

1.1.5 LR with GD - first iteration

- initial model parameters [0.11065559 0.3924683]
- gradients: [-0.25751434 0.00661256]
- updated model parameters: [0.13640702 0.39180704]
- learning rate: 0.1

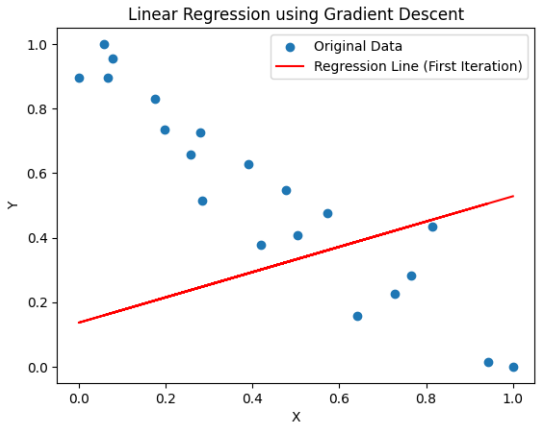

Figure 2: GD 1st iteration

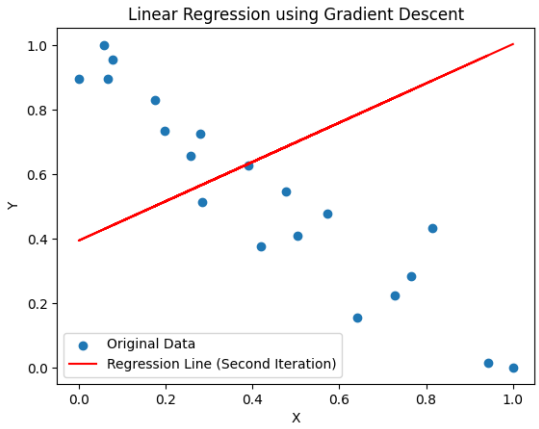

Figure 3: GD 2nd iteration

1.1.6 LR with GD - second iteration

- gradients: [0.14260905 0.20083417]
- updated model parameters: [0.39388986 0.60961307]

1.1.7 LR with GD - third iteration

- gradients: [0.11965978 0.18911778]
- updated model parameters: [0.38192388 0.59070129]

1.1.8 LR with GD - final iteration

- iterations: 1000
- gradients: [-3.01709946e-05 6.45755483e-05]
- updated model parameters: [0.9411715 -0.93211122]

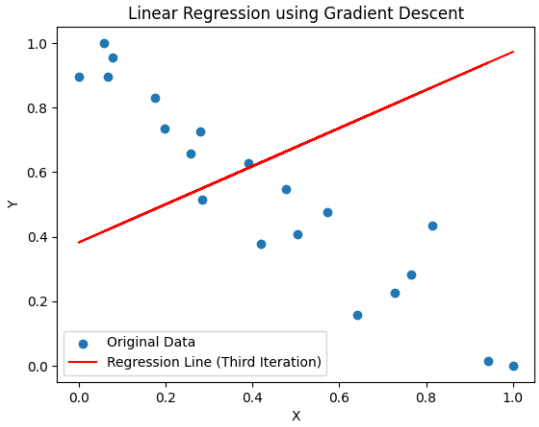

Figure 4: GD 3rd iteration

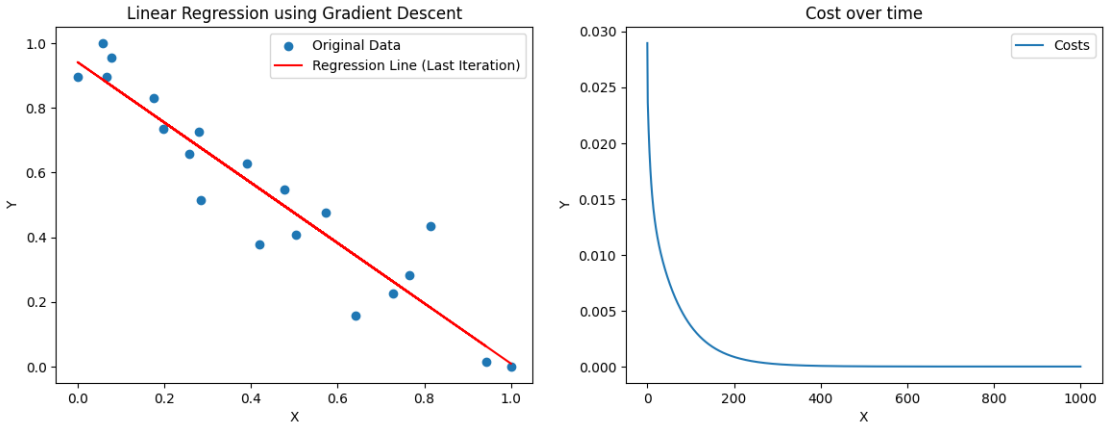

Figure 5: GD final iteration & cost over iterations

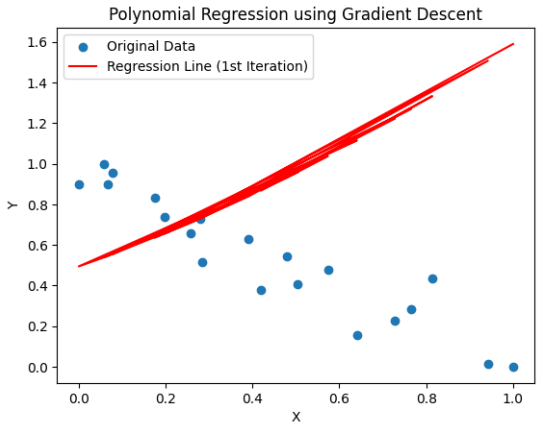

Figure 6: Polynomial GD 1st iteration

1.1.9 Discussion

Least Squares (LS) is a method that calculates the optimal coefficients for a linear model in a single step, relying on mathematical formulas. On the other hand, Linear Regression with Gradient Descent (GD) is an iterative optimization technique that gradually optimizes the coefficients through multiple steps, adjusting them to minimize the error. While LS provides a solution in one step, it can be computationally expensive when dealing with large or high-dimensional datasets due to matrix operations. In contrast, GD is more efficient when working with larger and high-dimensional datasets, as it processes data point by point or in mini-batches, making it scalable and memory-efficient.

LS is known for its simplicity, as it does not require fine-tuning hyper-parameters. It derives a direct solution from the data. In contrast, GD requires parameter tuning, such as including the selection of the learning rate and the number of iterations. These choices can significantly impact the algorithm's performance.

1.2 A.II. Polynomial Regression

1.2.1 Partial Derivatives

- initial model parameters: [0.46227201 0.63336415 0.75533532]
- Cost: 0.32583882011526083
- Gradient: [0.82382048 0.74832303 0.68218229]

1.2.2 Updating Rules

Updated model parameters: [0.08238205, 0.0748323 , 0.06821823]

1.2.3 GD - first iteration

- initial model parameters: [0.58660761 0.82525636 0.42217028]
- Gradient: [0.92454987 0.80409351 0.71734257]
- updated model parameters: [0.49415263 0.74484701 0.35043602]

1.2.4 GD - second iteration

- Gradient: [0.55744704 0.50018841 0.4511467]
- updated model parameters: [0.43840792 0.69482817 0.30532135]

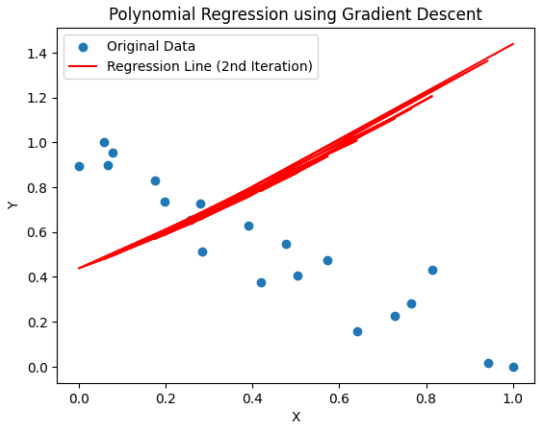

Figure 7: Polynomial GD 2nd iteration

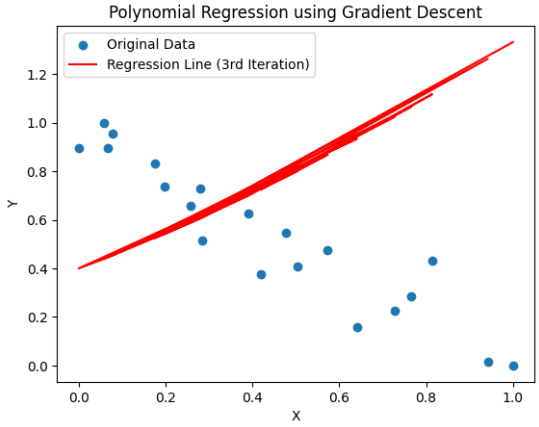

Figure 8: Polynomial GD 3rd iteration

1.2.5 GD - third iteration

- Gradient: [0.38546491 0.35876326 0.32636611]
- updated model parameters: [0.39986143 0.65895184 0.27268474]

1.2.6 GD - final iteration

- Gradient: [-1.38055585e-04 2.73844137e-04 7.71039098e-05]
- updated model parameters at final step 3615: [0.79083609 -0.24885059 -0.56452113]
- max iterations: 10000
- learning rate: 0.1

1.2.7 Discussion

Linear regression is the preferred choice for modeling straightforward, linear relationships in your data, making it suitable for scenarios where the data points follow a simple, straight-line trend; such as predicting temperature based net hourly energy output like in our example. In contrast, polynomial regression is employed when dealing with more complex, non-linear relationships, allowing for the modeling of curves and bends in the data.

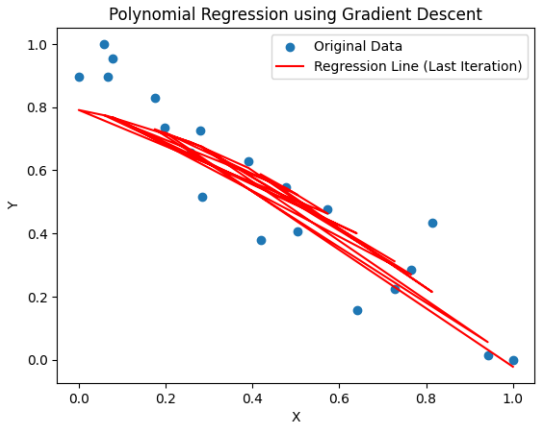

Figure 9: Polynomial GD final iteration

While linear regression involves a simple model with just two parameters: 'a' and 'b', making it easy to work with and interpret, polynomial regression can become more complex, particularly as you introduce higher-degree polynomial terms, which complicates the model by introducing more parameters to estimate.

It's important to note that linear regression is less susceptible to overfitting, while polynomial regression can be more prone to overfitting, requiring careful model selection and potential regularization techniques.

The choice between these two methods hinges on the nature of your data and the specific relationships you aim to model. Linear regression excels in simplicity and interpretability for linear relationships, while polynomial regression offers flexibility to capture more complex, non-linear data patterns.

2 B. Supervised Learning - Classification

2.1 B.I. Classification Models Comparison

2.1.1 B.I.1 Dataset Creation

The approach used involves the use of the make_classification function from the scikit-learn library, which enables the generation of synthetic datasets for classification tasks. In this context, the function is defined to create a dataset representing a binary classification problem with 80 data points in total, equally distributed between two distinct classes. The dataset is designed to have two continuous real-number attributes without any redundant or associated features. Also, the random_state parameter ensures reproducibility and allows production of the same dataset for analysis and experimentation. The output provides the data points for Dataset 1, displaying the values of the two features along with their corresponding class labels, demonstrating the successful creation of the dataset based on the specified approach.

Data points for Dataset 1 :	
[-1.20865506 1.39853577]	1
[1.95588774 0.51437479]	0
[1.53926109 0.96887226]	0
[-0.27137152 1.72297949]	0
[-1.53339249 0.29640149]	1
[0.68567276 1.22642031]	0
[-0.61642745 1.12278866]	1
[1.61764757 0.71966389]	0
[-1.89652507 0.22925863]	1
[-1.19990657 0.89553503]	1
[-1.20201373 1.15031085]	1
[-1.48560338 1.17781406]	1
[0.95578557 0.54747838]	0
[-0.49017759 1.07601291]	0
[-0.35470274 1.16363465]	1
[1.32634016 1.45679663]	0
[-0.52618558 1.31331921]	1
[1.813158 0.59356382]	0
[0.12056603 1.39985158]	0
[2.02724679 1.0258552]	0
[-0.48719718 1.18178803]	1
[2.96684913 0.20691788]	0
[1.5391322 0.78786631]	0
[-0.75326507 1.11457954]	1
[-1.78483513 0.99206241]	1
[-1.1752608 0.97881933]	1
[0.5147419 1.75178976]	0
[1.63583502 1.16770143]	0
[-2.20015787 0.22776286]	1
[1.38705287 0.9720031]	0
[-1.22517788 0.97365692]	1
[0.73364143 0.5048166]	0
[-1.05577689 0.54618484]	1
[0.22264648 1.63365256]	0
[-1.80221269 0.49805788]	1
[-0.33791796 2.22241522]	0
[-0.90486794 0.70013777]	1
[-1.22385118 0.89924665]	1
[1.12361352 -0.17931905]	0
[1.30100908 1.03301356]	0

Data points for Dataset 1:	
[-0.27044505 1.50873366]	1
[0.47075595 1.16030771]	0
[-1.64714051 0.4615123]	1
[0.87574425 0.88785539]	0
[-3.20909336 -0.72961284]	1
[1.87027696 0.60981779]	0
[-0.34596117 1.7709182]	1
[-1.69224258 0.93718732]	1
[0.05849886 1.32431612]	1
[0.00806779 1.79872616]	1
[1.55997485 2.56518756]	1
[1.23586429 0.66293756]	0
[0.26674596 1.65990745]	1
[0.46237936 1.35720506]	0
[1.05510824 0.71402034]	0
[2.68054502 -0.455369]	0
[0.50726702 1.50002943]	1
[-0.74679839 1.00334237]	1
[-0.58459962 1.39500688]	0
[-0.16930976 1.7695967]	0
[-1.40546311 1.54753492]	1
[3.03822303 -0.30709332]	0
[-1.62672636 0.55393945]	1
[1.4099803 -0.05130703]	0
[1.56024947 2.67825113]	1
[0.18118807 1.46586851]	1
[0.97797973 0.84198329]	0
[-0.21302457 1.475438]	1
[2.3388851 0.09169289]	0
[0.09165091 0.5606838]	0
[0.57336357 1.49931658]	0
[-0.01469419 1.95166588]	0
[0.10025498 1.38127515]	0
[0.09995307 1.41720834]	0
[-0.90379865 1.37957365]	1
[-0.43966452 1.43911883]	1
[2.00037091 0.17858245]	0
[-1.79838317 0.15272089]	1
[-2.3393616 -0.12037193]	1
[-1.21215286 0.46392536]	1

2.1.2 B.I.2 Dataset 2 creation (Dataset 1 + outliers

The approach involves the introduction of outliers to Dataset 1 by generating 4 outliers for each class, ensuring that they fall within a predefined range of values. Then, they are combined with the original dataset, creating the Dataset 2, containing the original data points along with the new added outliers, expanding the dataset to include the additional data points for each class.

Data points for Dataset 2 (including outliers):	
[-1.20865506 1.39853577]	1.0
[1.95588774 0.51437479]	0.0
[1.53926109 0.96887226]	0.0
[-0.27137152 1.72297949]	0.0
[-1.53339249 0.29640149]	1.0
[0.68567276 1.22642031]	0.0
[-0.61642745 1.12278866]	1.0
[1.61764757 0.71966389]	0.0
[-1.89652507 0.22925863]	1.0
[-1.19990657 0.89553503]	1.0
[-1.20201373 1.15031085]	1.0
[-1.48560338 1.17781406]	1.0
[0.95578557 0.54747838]	0.0
[-0.49017759 1.07601291]	0.0
[-0.35470274 1.16363465]	1.0
[1.32634016 1.45679663]	0.0
[-0.52618558 1.31331921]	1.0
[1.813158 0.59356382]	0.0
[0.12056603 1.39985158]	0.0
[2.02724679 1.0258552]	0.0
[-0.48719718 1.18178803]	1.0
[2.96684913 0.20691788]	0.0
[1.5391322 0.78786631]	0.0
[-0.75326507 1.11457954]	1.0
[-1.78483513 0.99206241]	1.0
[-1.1752608 0.97881933]	1.0
[0.5147419 1.75178976]	0.0
[1.63583502 1.16770143]	0.0
[-2.20015787 0.22776286]	1.0
[1.38705287 0.9720031]	0.0
[-1.22517788 0.97365692]	1.0
[0.73364143 0.5048166]	0.0
[-1.05577689 0.54618484]	1.0
[0.22264648 1.63365256]	0.0
[-1.80221269 0.49805788]	1.0
[-0.33791796 2.22241522]	0.0
[-0.90486794 0.70013777]	1.0
[-1.22385118 0.89924665]	1.0
[1.12361352 -0.17931905]	0.0
[1.30100908 1.03301356]	0.0
[-0.27044505 1.50873366]	1.0
[0.47075595 1.16030771]	0.0
[-1.64714051 0.4615123]	1.0
[0.87574425 0.88785539]	0.0

Data points for Dataset 2 (including outliers):	
[-3.20909336 -0.72961284]	1.0
[1.87027696 0.60981779]	0.0
[-0.34596117 1.7709182]	1.0
[-1.69224258 0.93718732]	1.0
[0.05849886 1.32431612]	1.0
[0.00806779 1.79872616]	1.0
[1.55997485 2.56518756]	1.0
[1.23586429 0.66293756]	0.0
[0.26674596 1.65990745]	1.0
[0.46237936 1.35720506]	0.0
[1.05510824 0.71402034]	0.0
[2.68054502 -0.455369]	0.0
[0.50726702 1.50002943]	1.0
[-0.74679839 1.00334237]	1.0
[-0.58459962 1.39500688]	0.0
[-0.16930976 1.7695967]	0.0
[-1.40546311 1.54753492]	1.0
[3.03822303 -0.30709332]	0.0
[-1.62672636 0.55393945]	1.0
[1.4099803 -0.05130703]	0.0
[1.56024947 2.67825113]	1.0
[0.18118807 1.46586851]	1.0
[0.97797973 0.84198329]	0.0
[-0.21302457 1.475438]	1.0
[2.3388851 0.09169289]	0.0
[0.09165091 0.5606838]	0.0
[0.57336357 1.49931658]	0.0
[-0.01469419 1.95166588]	0.0
[0.10025498 1.38127515]	0.0
[0.09995307 1.41720834]	0.0
[-0.90379865 1.37957365]	1.0
[-0.43966452 1.43911883]	1.0
[2.00037091 0.17858245]	0.0
[-1.79838317 0.15272089]	1.0
[-2.3393616 -0.12037193]	1.0
[-1.21215286 0.46392536]	1.0
[-0.38403059 -2.84444261]	0.0
[0.29797487 -0.38806564]	0.0
[-0.47779319 -1.01799107]	0.0
[-1.7721082 0.7156258]	0.0
[-1.20207196 -1.39903635]	1.0
[0.726803 0.17485257]	1.0
[-2.19252033 0.08146873]	1.0
[-1.89336081 1.71201089]	1.0

2.1.3 B.I.3 Split Dataset 1 and Dataset 2 in training and testing data

The data was split into training and testing sets using an 80-20 split ratio, with 80% of the data allocated for training and 20% for testing. It ensures that an adequate amount of data is available for training the models while also providing sufficient data for testing their performance. The split data outlines the shapes of training and testing sets for both datasets, and provides a clear overview of the dataset distributions for further reference and analysis.

```
Split data for Dataset 1:
X1_train shape: (64, 2), y1_train shape: (64,)
X1_test shape: (16, 2), y1_test shape: (16,)

Split data for Dataset 2:
X2_train shape: (70, 2), y2_train shape: (70,)
X2_test shape: (18, 2), y2_test shape: (18,)
```

2.1.4 B.I.4 k-NN, Naive Bayes, Decision Trees, and Random Forests

For Dataset 1, the classification models achieved an accuracy of 0.875, indicating that 87.5% of the predictions were correct. The confusion matrix revealed 6 true positives, 8 true negatives, 2 false positives, and 0 false negatives. In Dataset 2, the models achieved an accuracy of 0.8333, indicating that 83.33% of the predictions were correct. The confusion matrix showed 11 true positives, 4 true negatives, 1 false positive, and 2 false negatives. The results demonstrate the models ability to classify data accurately, with slightly higher accuracy achieved for Dataset 1 compared to Dataset 2, suggesting that the outliers in Dataset 2 have maybe affected the overall accuracy.

```
Results for Dataset 1:
Accuracy: 0.875
Confusion Matrix:
[[6 2]
 [0 8]]


Results for Dataset 2:
Accuracy: 0.8333333333333334
Confusion Matrix:
[[11  1]
 [ 2  4]]
```

2.1.5 B.I.6 Discussion

The visualized decision boundaries offer insights into the similarities and dissimilarities between the various classification models. Across these 4, some similarities can be observed : they can capture linear decision boundaries effectively when the data is well separated, as seen in the case of the linearly separable datasets. But there are also some dessimilarities with more complex datasets, where the models behavior diverges : K-NN tends to produce boundaries based on local data points, while Naive Bayes assumes feature independence, resulting in simpler decision boundaries. Decision Trees and Random Forests can handle complex decision boundaries effectively, with Random Forests showing improved robustness and reduced overfitting. These observations highlight the need to select models based on the properties of the dataset and the intended trade-offs, as they highlight the trade-offs between model complexity, interpretability, and generalization performance.

3 C. Unsupervised Learning

3.1 C.I. Iris Clustering with K-means

3.1.1 C.I.1. K-means description

The K-means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping clusters where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as "far" as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at

the minimum. The less variation we have within clusters, the more homogeneous the data points are within the same cluster.

3.1.2 C.I.2. Running the K-means algorithm for iris.csv

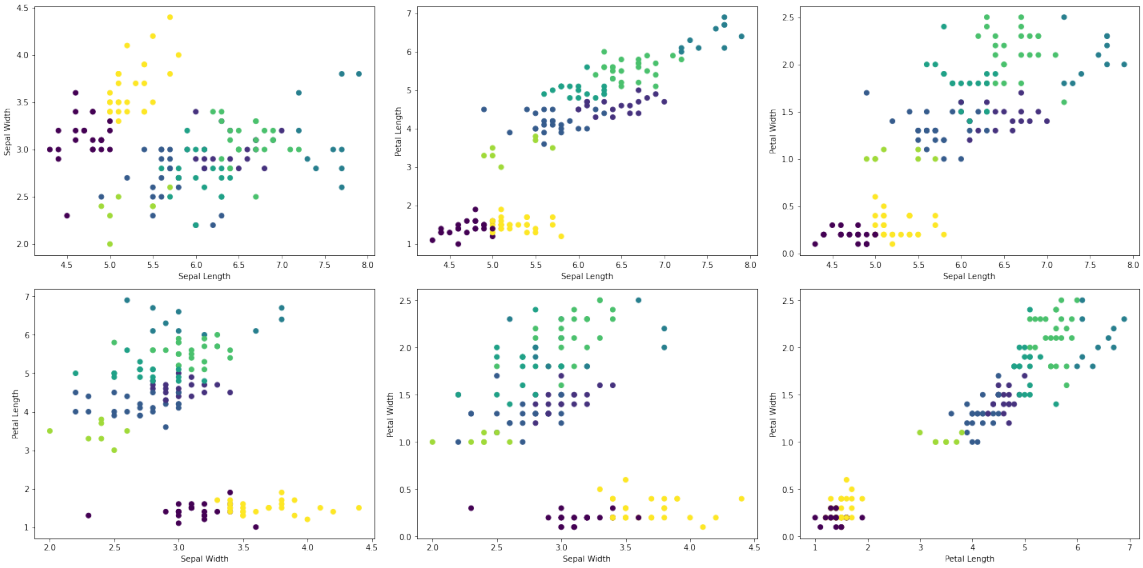

Figure 10: Plots made by the k-means algorithm.

3.1.3 C.I.3. Seeing if the model is viable for our problem

Once done with the algorithm that is provided in the task_C.ipynb file, we get :

Accuracy: 0.24

Confusion matrix: $\begin{pmatrix} 0. & 47. & 14. \\ 50. & 0. & 0. \\ 0. & 3. & 36. \end{pmatrix}$

We see that the model is not suitable because the accuracy is nearly at 0.25, however we need a result near 1 to say that it is suitable. The confusion matrix shows that the model is not able to predict the right cluster for the data points.

3.1.4 C.I.4. Silhouette score

Upon launching the code provided in the file from this part, we have a silhouette score of 0.55 which means that the clusters are not well separated : the silhouette score is there as a measure of how close each point in one cluster is to points in the neighboring clusters.

3.1.5 C.I.5. Applying the K-means to unknownspecies.csv

When having the result, we can see that the silhouette score is 0.44, which means that the clusters from the unknown species are even more seperated than the clusters from the iris dataset.

3.1.6 C.I.6. Doing a k-means algorithm from scratch

When done, the results that the algorithm gives us is the following :

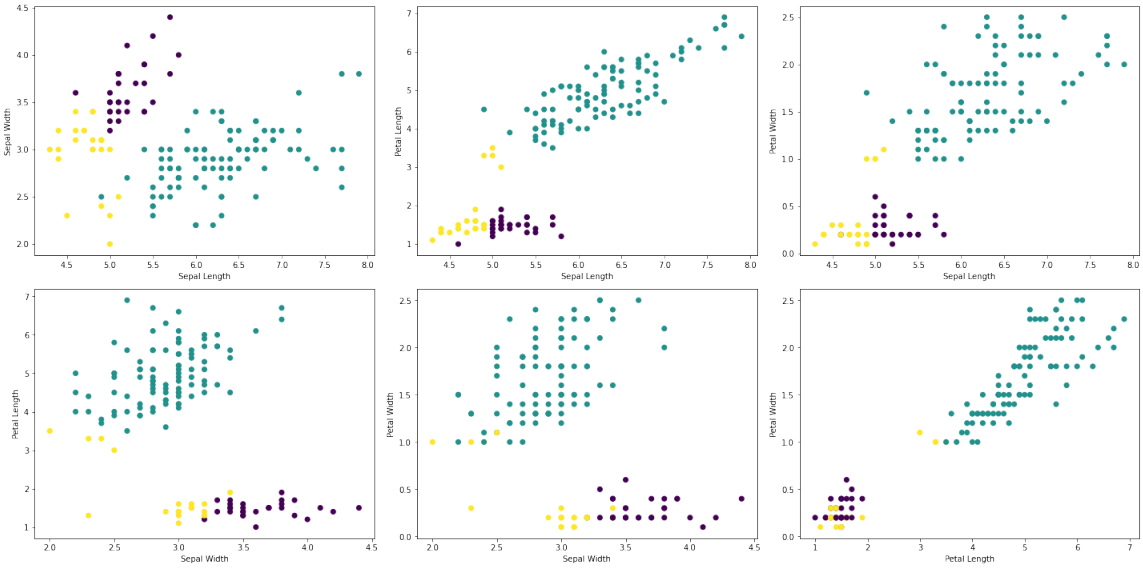

Figure 11: Plots made with our custom made k-mean algorithm.

When calculating the accuracy and the confusion matrix, we have :

Accuracy: 0.51

$$\textit{Confusion matrix: } \begin{pmatrix} 30. & 0. & 0. \\ 0. & 46. & 50. \\ 20. & 4. & 0. \end{pmatrix}$$

This means that the algorithm provided in the file, in the C.I.6 part, is more suitable for our problem than the sklearn model.

3.2 C.II. Clustering of Three Dimensional Shape Dataset Using DBScan

3.2.1 C.II.1. Explaining DBScan

The DBScan algorithm is a density-based clustering algorithm. It is based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density. The DBSCAN algorithm uses two parameters: epsilon and the minimum number of points required to form a dense region (minPts). It starts with an arbitrary starting point that has not been visited. This point's epsilon-neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized epsilon-environment of a different point and hence be made part of a cluster.

3.2.2 C.II.2. Plotting ulu.csv dataset with DBScan algorithm

When using the algorithm provided in the task_C.ipynb file, we get this 3D plot :

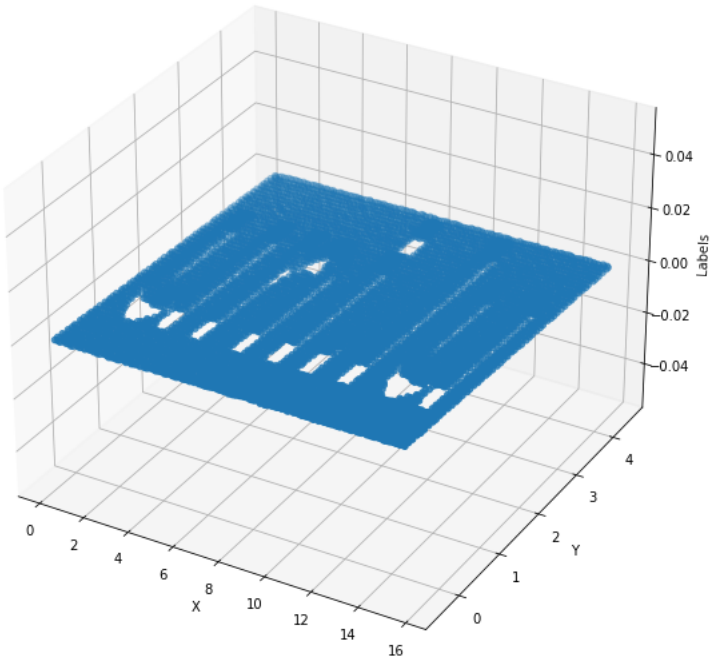

Figure 12: 3D plot of the ulu.csv dataset using DBScan.

The clusters form a uni.lu logo, when plotting in 3D. 8 clusters are gerenated, while using the program.

3.2.3 C.II.3. Scattering

As said previously, when clustering the dataset, we get a uni.lu logo in 3D (if executed with our program).

3.2.4 C.II.4. Running K-means on ulu.csv

When using the K-means algorithm on ulu.csv, we get this plot in 3D :

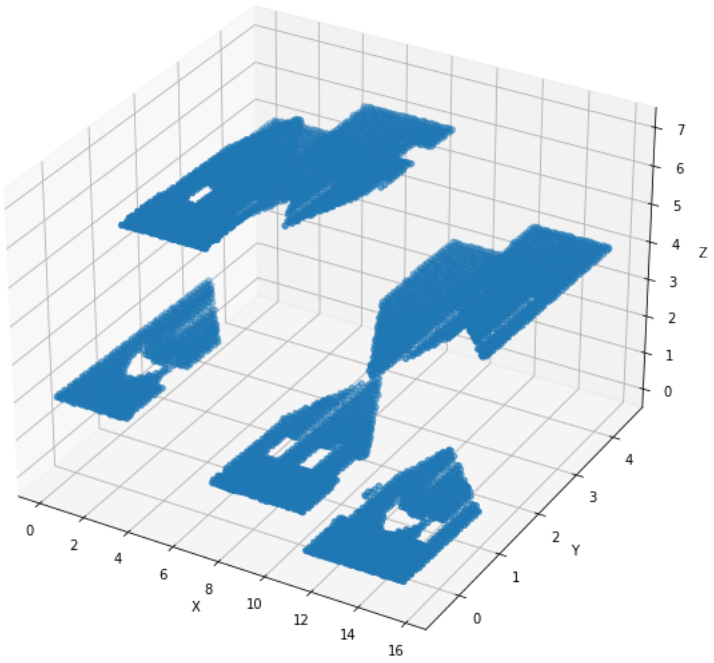

Figure 13: 3D plot of the ulu.csv dataset using K-means.

Those clusters doesn't make sense using the k-means algorithm since the clusters aren't forming the initial uni.lu logo shape that the DBSCAN algorithm did.

3.2.5 C.II.5. Computing silhouette scores

Upon launching the program, the k-means silhouette score is 0.373, while the one for the DBScan is completelly in the negative, with -0.21. Its like this since k-means is better at clustering the data than DBScan, even though the shape is better in DBScan.

3.2.6 C.II.6. Running PCA dimensionality reduction on dataset

When doing this process on the dataset, we get a plot like this :

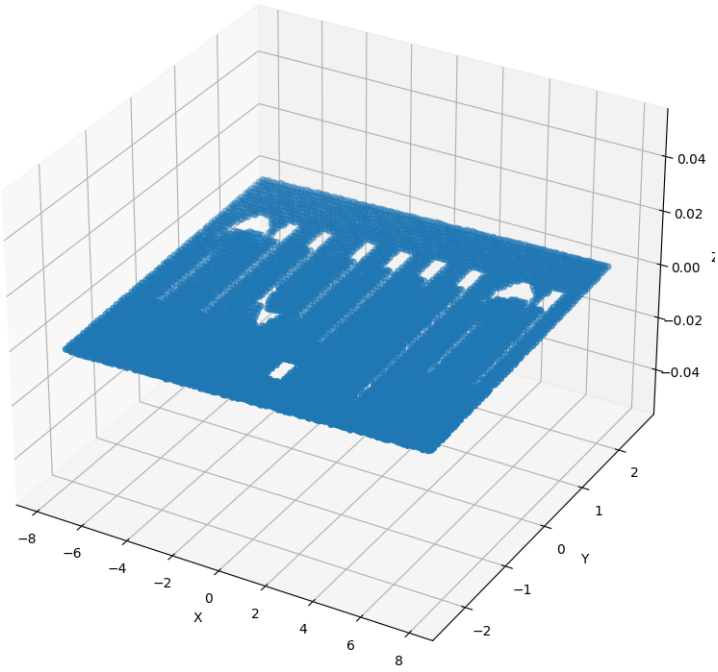

Figure 14: 3D plot of the dataset under PCA dimensionality reduction.

3.2.7 C.II.7. DBScan VS PCA'd DBScan

When using the PCA scan, we get a sort of mirrored version of the DBScan plot. This is due to the fact that PCA tries to find the directions of maximum variance in the data, which may not necessarily align with the original axes. Therefore, the PCA transformation can result in a reflection or rotation of the original data, which can cause the clusters to appear reversed or shifted.