FTML practical session 7

17 avril 2025

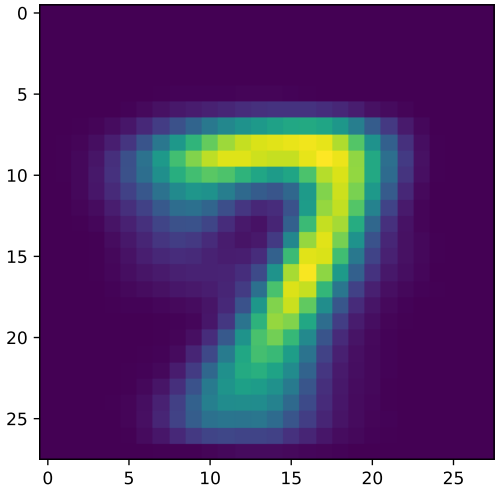


TABLE DES MATIÈRES

1 APPLICATION OF VECTOR QUANTIZATION TO CLASSIFICATION

In this exercise, we build a classifier based on an unsupervised preprocessing of the data. We will apply **vector quantization** to the MNIST classification problem. Namely, we will compute **an average representer** (prototype) for of each class, and for a new sample, predict the class of the nearest prototype. Some of the prototypes are represented in figures 1, 2, 4, 3, 6, 5. We might not expect a very high classification accuracy with this classifier, but this is rather interesting as a simple benchmark for this classification problem, on which we can achieve more than 0.98 accuracy with neural networks for instance, in a couple of minutes. We can see some examples of misclassified digits in figure 10 and 11.

https://en.wikipedia.org/wiki/Vector_quantization
You can fetch the data using vector_quantization/fetch_data.py.

A **Gaussian blur** of the input digits (see figures 7, 8, 9) might slightly improve the classification performance.

Implement this classification method based on vector quantization in order to classify the MNIST dataset and evaluate the test accuracy of this method. You might optionnaly use a hyperparameter optimization method in order to find a good value of the σ parameter of the GaussianBlur method of OpenCV.
https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11a

For this exercise, a decomposition of the dataset into a train/validation/test might be enough, but you can also use a cross validation.

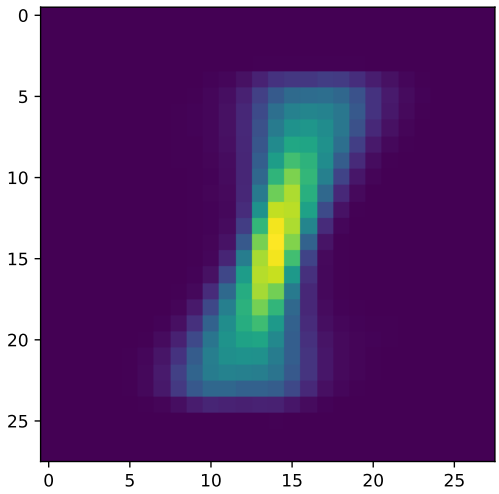


FIGURE 1 – Average of the 1 class (prototype).

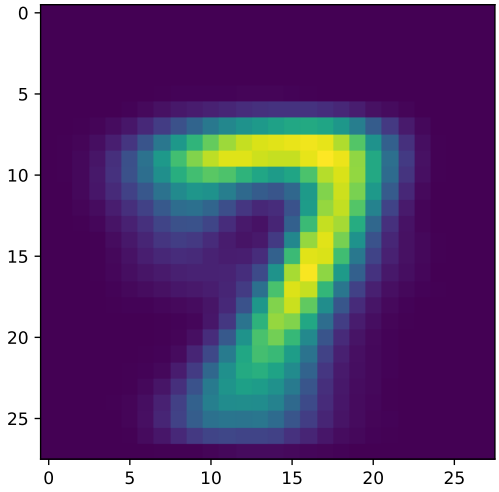


FIGURE 2 – Average of the 7 class (prototype).

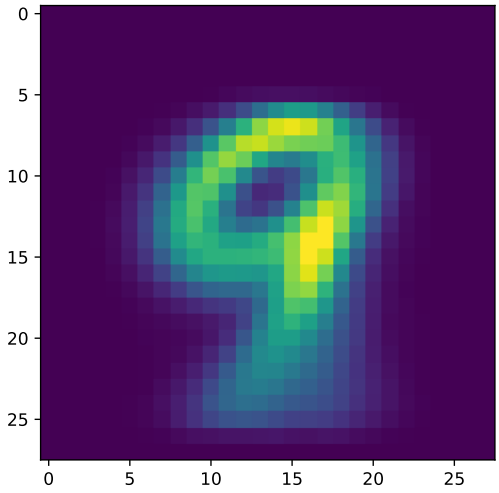


FIGURE 3 – Average of the 9 class (prototype).

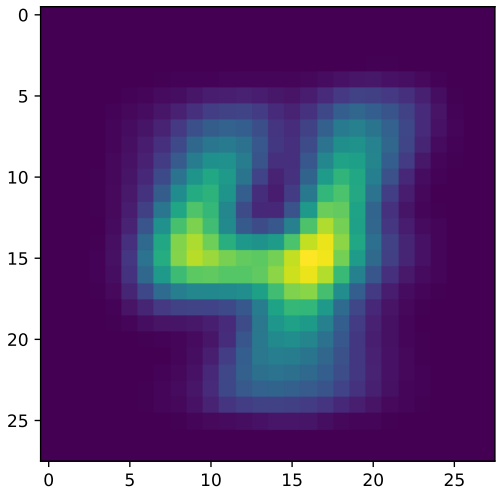


FIGURE 4 – Average of the 4 class (prototype).

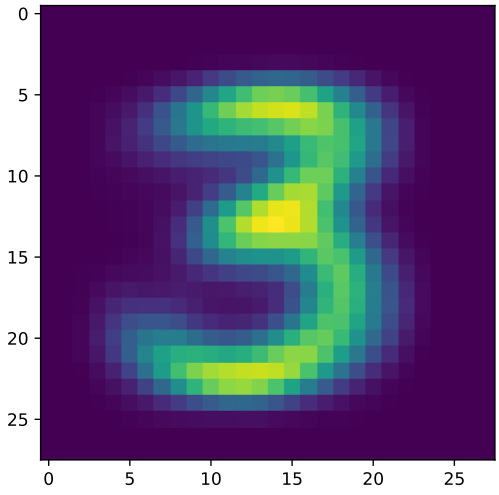


FIGURE 5 – Average of the 3 class (prototype).

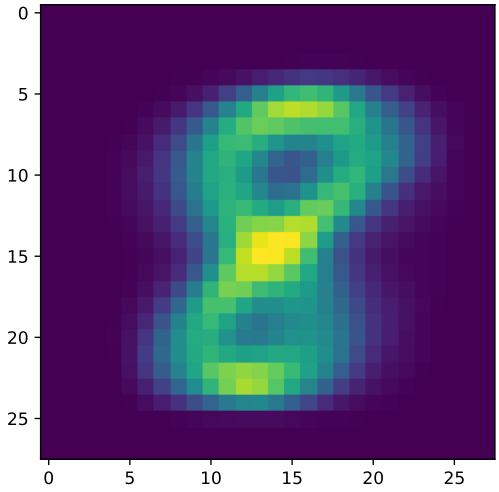


FIGURE 6 – Average of the 8 class (prototype).

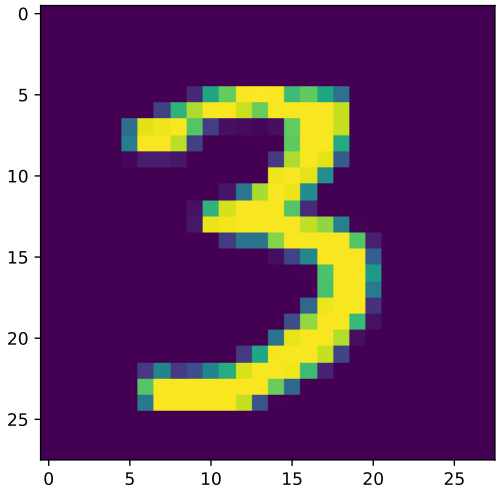


FIGURE 7 – A digit from the dataset.

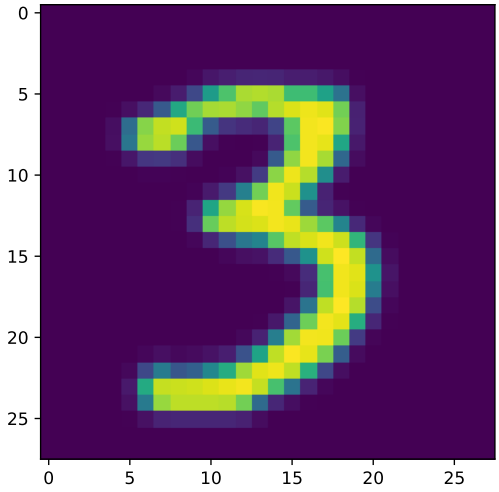


Figure 8 – The same digit, blurred with $\sigma = 0.5$ (Gaussian blur)

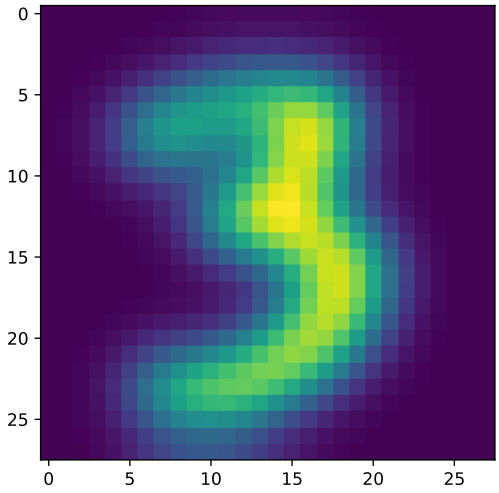


Figure 9 – The same digit, blurred with $\sigma = 2$

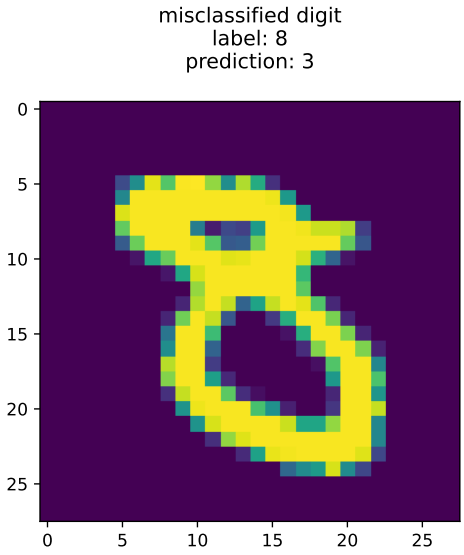


FIGURE 10 – Example of misclassified digit

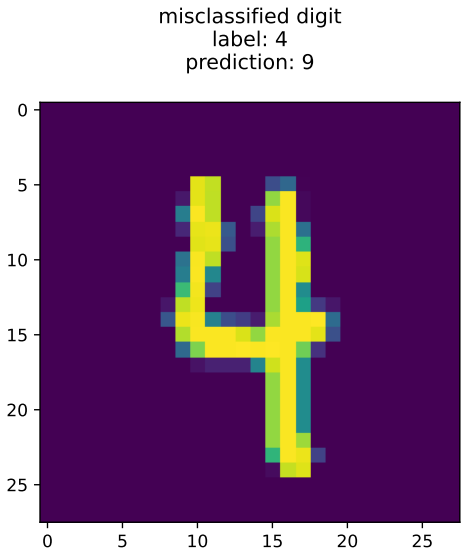


FIGURE 11 – Example of misclassified digit

2 COMPATIBILITY GRAPHS AND CUSTOM METRICS

In this exercise we build a custom distance in a dataset in order to observe a given compabtility graph (a graph where there is an edge between nodes that are closer than some threshold value). This illustrates the fact that the choice of the metric determines which samples are considered as similar or not in a dataset, which can be very important for practical applications, especially in unsupervised learning.

Even for geometric (and thus numerical data), the classical euclidean distance is not the only available metric. If we take a look at the documention of **cdist** from scipy or **numpy.linalg.norm**, we see that many metrics exist.

`https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html`

`https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html`

Use the notebook **build _graphs_geometric_data.ipynb** inside **exercise _2 _metrics/** n order to build compabtility graphs for the data contained in **data/data.npy** (displayed in figure 12), in order to obtain the graphs shown in figures 13, 14, 15, 16, 17. You will need to choose the right **metric** for each graph. Try to think of the metric only mentally **before** implementing it!

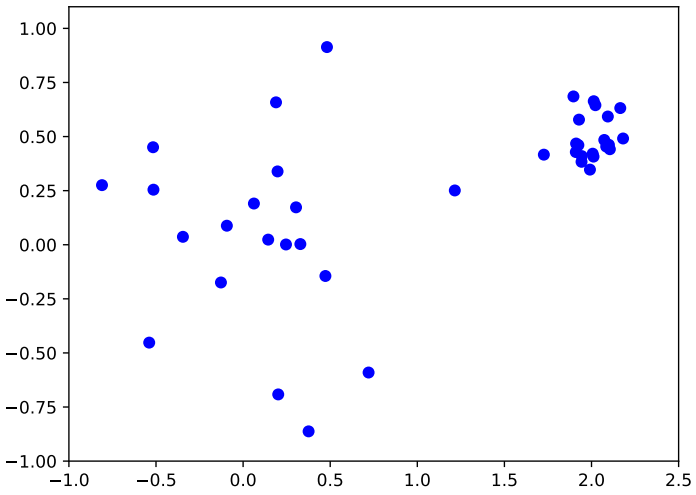


FIGURE 12 – The data to build compabtility graphs from.

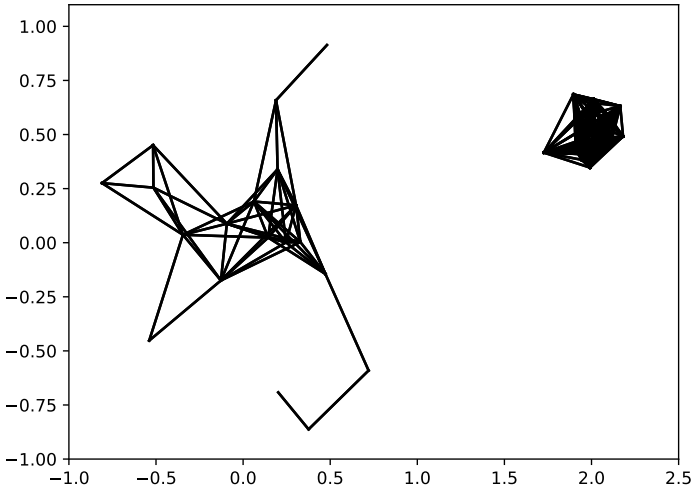


Figure 13 – Graph 1

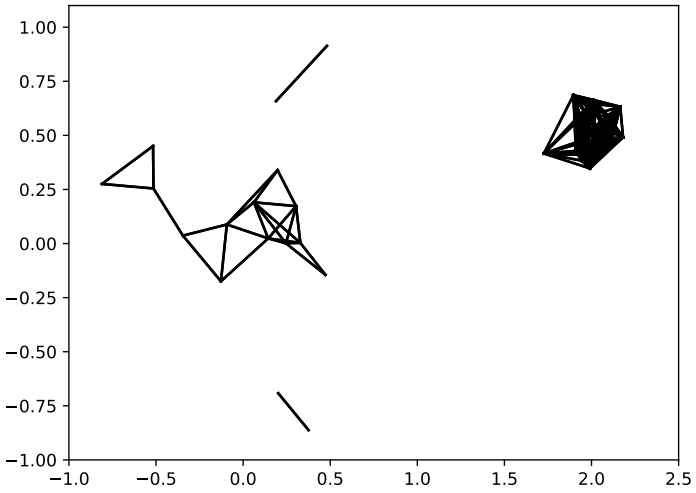


Figure 14 – Graph 2

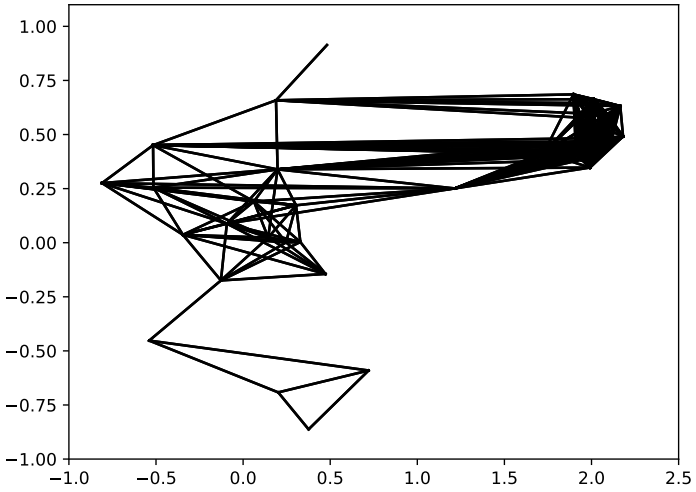


FIGURE 15 – Graph 3

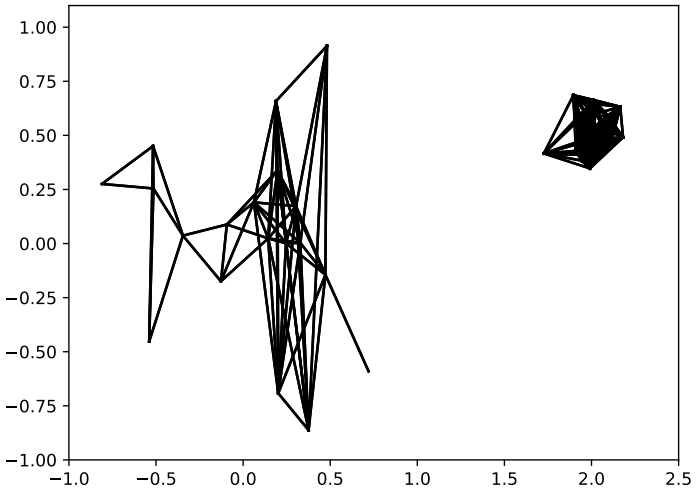


FIGURE 16 – Graph 4

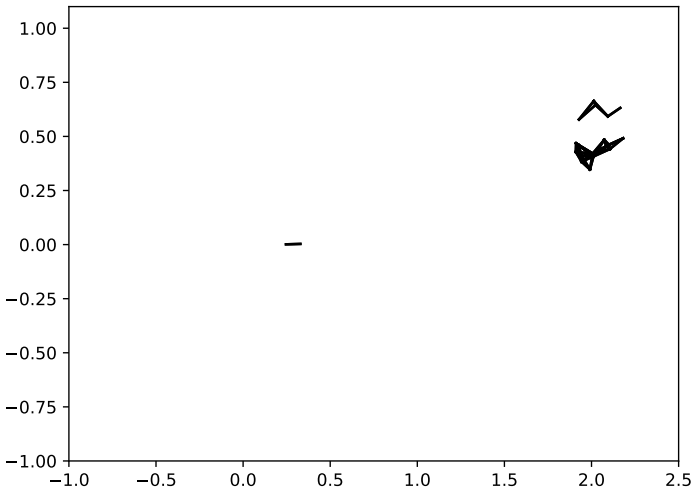


FIGURE 17 – Graph 5

3 INFLUENCE ON DATA SCALING ON THE CONVERGENCE OF SGD

We would like to perform a binary clasification on the following 3 datasets (figures 18, 19, 20). Each one has a different difficulty for a linear classifier, such as a SVM. We also note that the scales are different on the x and y axes.

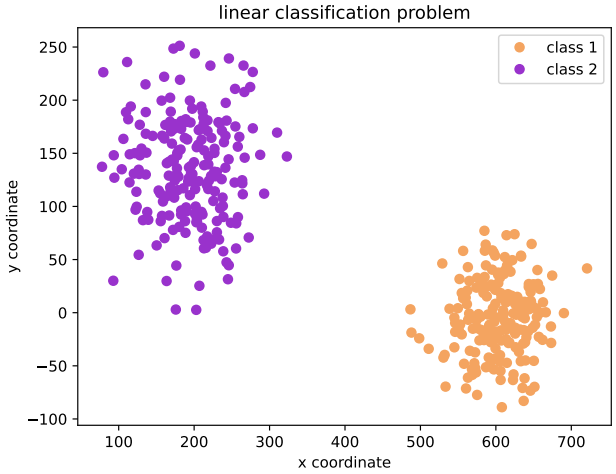


FIGURE 18 – Dataset 1

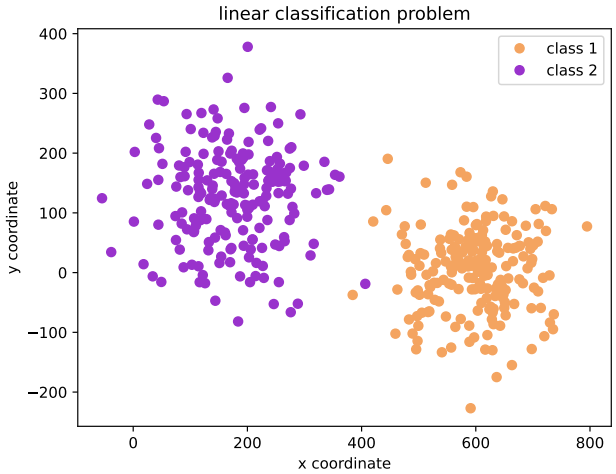


FIGURE 19 – Dataset 2

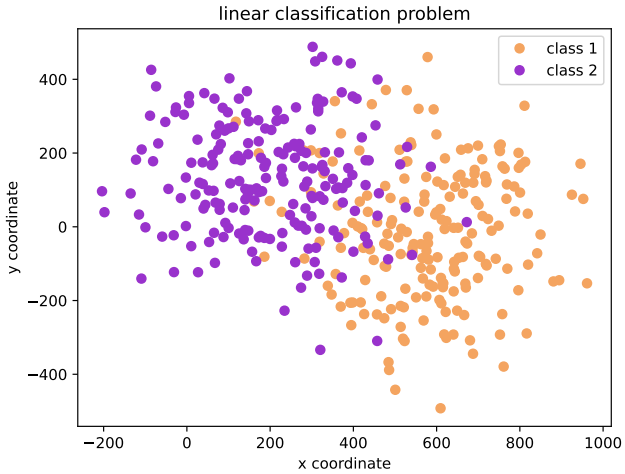


FIGURE 20 – Dataset 3

The data are located in **exercice _3 _svm_sgd/data/**.

Data standardization consists in transforming the data so that each feature (each column) is centered (zero mean) and has a variance equal to 1. It is experimentally often noticed that algorithms trained by SGD give a better performance (generalization error) when the data are standardized.

`https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler`.

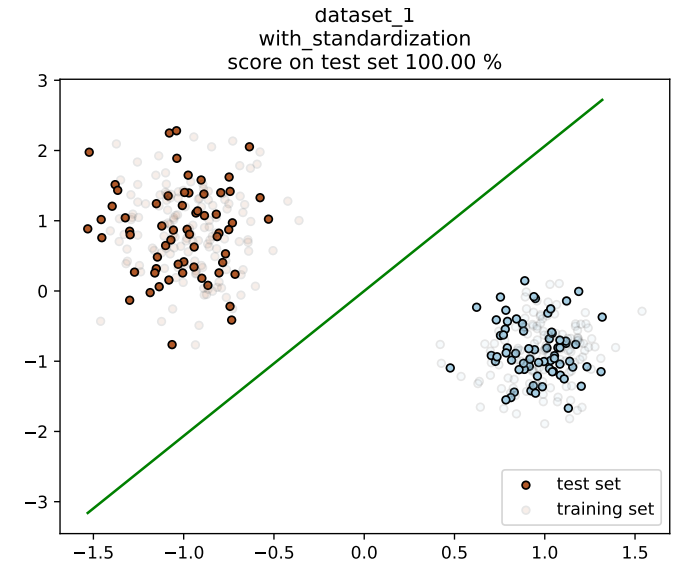
Perform a linear classification on these data, optimized by SGD, and compare quality of the results with and without preprocessing the data by standardizing them. You may use scikit-lean to do it, in particular :

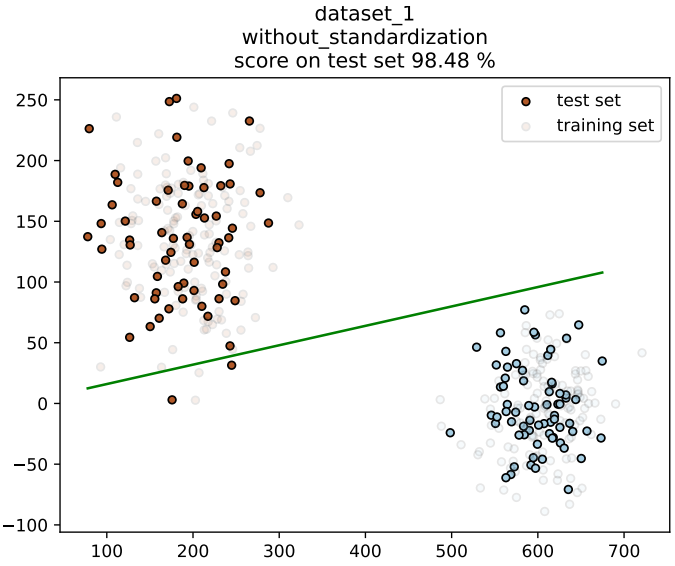
— `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html`. By default, this class trains a linear SVM (no kernel).

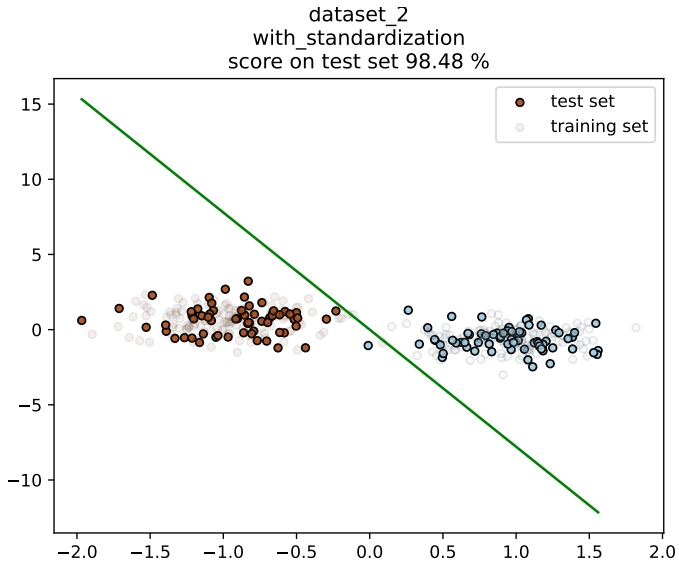
— `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

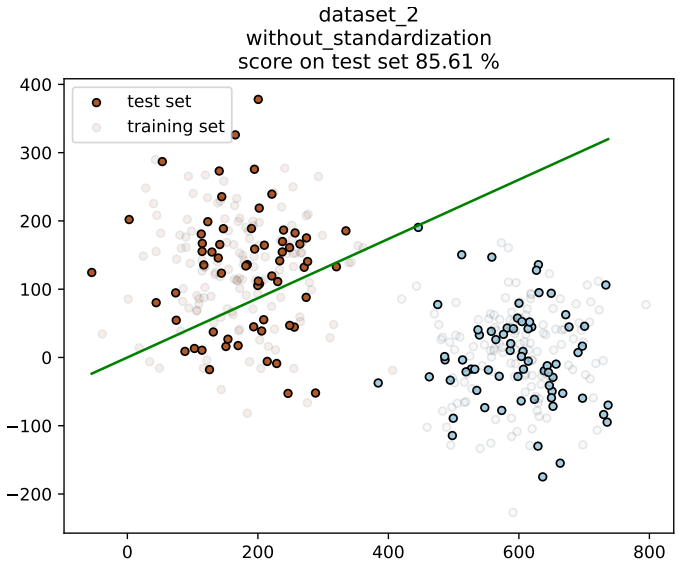
— `https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html`

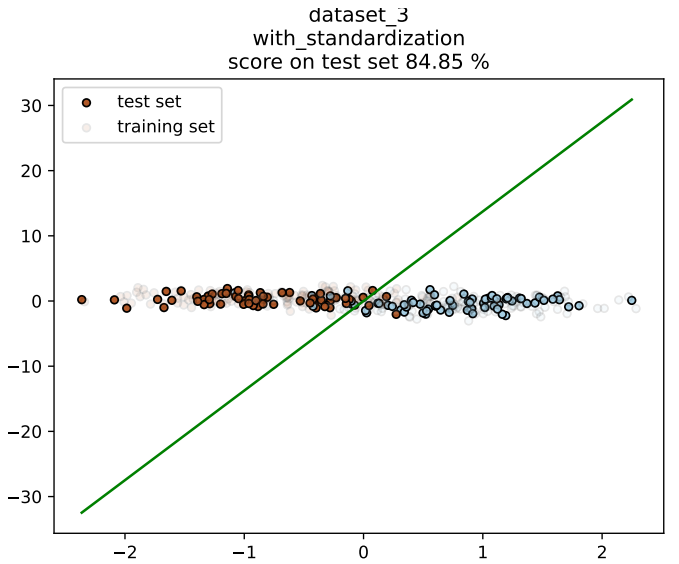
You should obtain results like the following figures.

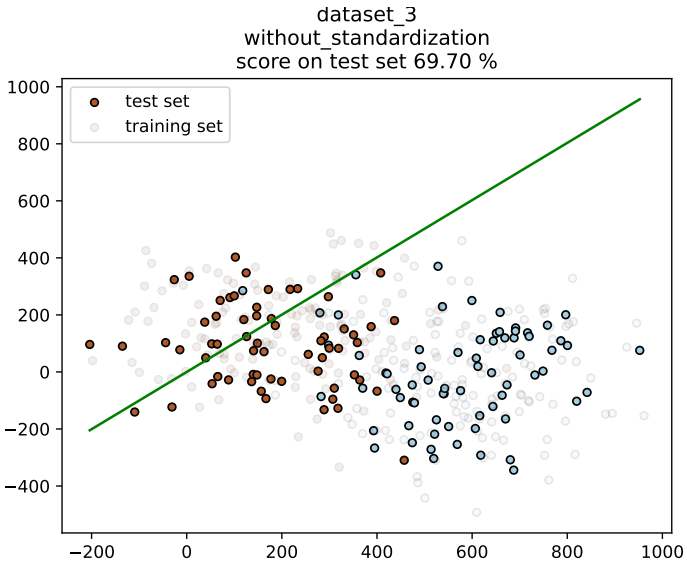





dataset_2
with_standardization
score on test set 98.48 %
test set
training set
15
10
5
0
−5
−10
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


dataset_2
without_standardization
score on test set 85.61 %
test set
training set
400
300
200
100
0
−100
−200
0
200
400
600
800


dataset_3
with_standardization
score on test set 84.85 %
test set
training set
30
20
10
0
−10
−20
−30
−2
−1
0
1
2



In this example, we saw that data standardization may give an improved performance, on a linear SVM trained by SGD. You can perform the same study on different datasets such as the toy datasets from scikit or other datasets.