Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

Machine Learning

Homework 1

10 class Classification

Ludovica Cartolano
cartolano.1796046@studenti.uniroma1.it

Course by
Prof. Luca Iocchi

Master in Control Engineering

Department of Computer, Control and Management Engineering
"Antonio Ruberti" (DIAG)
University of Rome "La Sapienza"
AY 2023/2024

Contents

List of Figures

Chapter 1

Introduction

Apply various classification problems on the training data-set and assess the results on the blind test data-set, given two data-sets with N samples each, each assigned one of ten classes, and distinct input space dimensions, d.

I have considered for each data-set Decision Tree Classifier, Support Vector Machine Classifier and K-Nearest Neighbours Classifier.

- **Decision Tree Classifier** is a *supervised* machine learning algorithm that creates a tree-like structure of decision rules. It keeps splitting the data into smaller groups until it finds the best way to predict outcomes for new data. It works like a flowchart, where each step (node) separates the data based on certain characteristics until it reaches the purest possible subset (leaf node). Once the tree is constructed, it can be used to make predictions on new, unseen data. Decision trees can be prone to over-fitting, especially when the tree becomes overly complex.

- **Support Vector Machine (SVM) Classifier** is a *supervised* machine learning algorithm used mainly for classifying intricate data-sets. It works by identifying the best hyperplane that separates different classes within the data's feature space. This hyperplane can be a line for two classes, but for more classes, SVMs use kernel functions to map data into higher-dimensional spaces. However, SVMs might be sensitive to the choice of kernel and parameters and could face challenges when dealing with large data-sets due to their computational complexity.

- **K-Nearest Neighbors (KNN) Classification** is a simple yet effective *supervised* machine learning algorithm. It relies on similarity, not predefined parameters, and stores the entire training data-set in memory. When predicting a new data point's class, KNN measures similarity to all training points, selects the closest K neighbors, and determines the class by majority voting among these neighbors. However, KNN has downsides: it needs significant memory for large data-sets, can be slow in predictions due to computing similarities for all data points, and might struggle when irrelevant or noisy features are present.

For each Classification Algorithm it has been used three different data normalization techniques:

- `StandardScaler()` method executes a normalization technique known as standardization. This process transforms data by adjusting it to have a mean of 0 and a standard deviation of 1. It operates on each feature of the data-set independently, making the data more comparable across different features. The formula used by to scale a feature x is

$$z = \frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$$

  Where: z represents the standardized value of x, mean(x) denotes the mean of the feature x and standard deviation(x) denotes the standard deviation of the feature x.

- `QuantileTransformer()` function is used for normalization and transforms features to conform to a chosen probability distribution function. When choosing the "uniform" distribution (output_distribution = "uniform"), it spreads the values uniformly across the range. This method works by computing the cumulative distribution function (CDF) of each feature and then applying the inverse of the CDF of the desired distribution to transform the data, ensuring it adheres to the specified distribution pattern.

- `Normalizer()` performs a type of normalization which rescales the values along rows such that the normalization is performed independently for each sample, treating each sample as a vector in a high-dimensional space performing L2 norm

$$x_{\text{normalized}} = \frac{x}{\|x\|_2}$$

  The transformation does not consider relationships between features, it operates on each sample independently.

And last, for each data normalization technique it has been used Hyperparameters Tuning to find the best hyperparameters to improve the evaluation task for each classification algorithm.

- **Decision Trees**:
  -*Criterion*: The function used to measure the quality of a split (e.g., "gini" or "entropy").
  -*Max Depth:* The maximum depth of the tree. Controls how deep the tree can grow, limiting its complexity.
  -*Min Samples Split*: Minimum number of samples required to split an internal node. Helps prevent splitting nodes that have very few samples.
  -*Min Samples Leaf*: Minimum number of samples required to be a leaf node. Ensures that each leaf has a minimum number of samples.
  Hyperparameter tuning is crucial to optimize the decision tree model's performance and prevent it from over-fitting or under-performing on unseen data.

- **SVMs**:
  -*C (Regularization parameter)*: Controls the trade-off between maximizing the margin and minimizing the classification error. Higher values of C allow more misclassifications on the training data but might generalize better, while lower

values enforce a larger margin but may lead to under-fitting.
-*Kernel*: SVMs can use different kernel functions, in this case linear or polynomial. The choice of kernel significantly affects the decision boundary.
-*Degree (Degree of polynomial kernel)*: Applicable for polynomial kernels. It determines the degree of the polynomial function.
Proper tuning considering the problem domain and experimentation with different values is crucial for improving the SVM's predictive ability and generalization to unseen data.

- **KNNs**:
  -*Number of Neighbors (K)*: The number of nearest neighbors considered when making predictions. Choosing an appropriate value for 'k' is crucial and affects the model's bias-variance trade-off.
  -*Weights*: Options include uniform weights (all points in the neighborhood are weighted equally) ans distance weights (closer neighbors have more influence).
  -*Distance Metric*: The distance metric used to measure the distance between points. It has been considered Euclidean distance, Manhattan distance and Chebyshev distance.
  Optimizing the hyperparameters in KNN is essential for improving the model's accuracy and generalization ability.

For each classification algorithm it has been used `RandomizedSearchCV()` which randomly samples hyperparameter values from defined distributions. This method is often more efficient than `GridSearchCV()` and can sometimes find better results. It uses Cross-Validation (CV) which is essential for evaluating different hyperparameter sets to ensure robustness and avoid over-fitting on the validation set.

Chapter 2

Classification Task

After having imported **important libraries** and defined **useful functions** I have proceeded by **loading** the training data-sets `data1.csv` and `data2.csv` and the blind tests for evaluation `blind_test1.csv` and `blind_test2.csv` provided by Prof. Iocchi.

Each training set and blind test sets have the following structures

Index	X	Y
0	$[V^0_1, V^0_2, V^0_3, V^0_4, \dots V^0_j, \dots, V^0_d]$	C_k
...	...	...
i	$[V^i_1, V^i_2, V^i_3, V^i_4, \dots V^i_j, \dots, V^i_d]$	C_k
...	...	...
N	$[V^N_1, V^N_2, V^N_3, V^N_4, \dots V^N_j, \dots, V^N_d]$	C_k

(a) Structure of training sets `.csv` files

Index	X
0	$[V^0_1, V^0_2, V^0_3, V^0_4, \dots V^0_j, \dots, V^0_d]$
...	...
i	$[V^i_1, V^i_2, V^i_3, V^i_4, \dots V^i_j, \dots, V^i_d]$
...	...
N	$[V^N_1, V^N_2, V^N_3, V^N_4, \dots V^N_j, \dots, V^N_d]$

(b) Structure of blind test sets `.csv` files

Figure 2.1: Structure of the different sets `.csv` files

For "Data-set 1" (`data1.csv`) one calls the two columns showed in Fig(2.1a) X_1 and Y_1 where the latter represents the labeled classes and for "Data-set 2" (`data2.csv`) one calls the two columns X_2 and Y_2 where the latter represents the labeled classes.
For "Blind Test 1" (`blind_test1.csv`) one calls the column showed in Fig(2.1b) X_b1 for which one wants to predict the classes in Y and for "Blind Test 2" (`blind_test2.csv`) one calls the column X_b2 for which one wants to predict the classes in Y.

Subsequently it has been done some **Data Exploration** and printed information on all four data-sets.

- Data-set1 has number of rows : 50 000, number of features: 100, number of classes: 10 called ["class: 0", "class: 1", "class: 2", "class: 3", "class: 4", "class: 5", "class: 6", "class: 7", "class: 8", "class: 9"] and number of samples: 50 000.

- Data-set2 has number of rows : 50 000, number of features: 1 000, number of classes: 10 called ["class: 0", "class: 1", "class: 2", "class: 3", "class:

4", "class: 5", "class: 6", "class: 7", "class: 8", "class: 9"] and number of samples: 50 000.

- Blind test 1 has number of rows : 10 000, number of features: 100, number of classes: 10 called ["class: 0", "class: 1", "class: 2", "class: 3", "class: 4", "class: 5", "class: 6", "class: 7", "class: 8", "class: 9"] and number of samples: 10 000.
- Blind test 2 has number of rows : 10 000, number of features: 1 000, number of classes: 10 called ["class: 0", "class: 1", "class: 2", "class: 3", "class: 4", "class: 5", "class: 6", "class: 7", "class: 8", "class: 9"] and number of samples: 10 000.

In the code [Cartolano 2023] it has also been printed random samples for each data-set.

```
x37432 = array([0.        , 0.5190806 , 0.30598775, 4.772967  , 3.5126166 ,
       0.        , 0.10551864, 3.1504204 , 0.        , 0.        ,
       0.        , 0.6847926 , 0.        , 0.        , 0.        ,
       0.        , 0.84311545, 0.        , 2.3414414 , 1.6788423 ,
       0.2540909 , 4.5153937 , 3.1205025 , 1.5244623 , 0.        ,
       0.        , 3.9803476 , 0.601688  , 0.25237313, 4.2377048 ,
       0.29189223, 0.        , 0.        , 0.6902828 , 0.        ,
       2.2398107 , 4.2515464 , 2.023187  , 7.3794384 , 0.        ,
       0.        , 0.        , 0.        , 6.4360566 , 0.3013732 ,
       0.        , 0.06469406, 1.606888  , 2.2645967 , 3.7000377 ,
       5.399876  , 5.7400374 , 0.        , 1.5829873 , 1.343856  ,
       2.6018646 , 0.        , 1.9570346 , 0.74797237, 0.        ,
       0.        , 0.9107065 , 0.        , 0.        , 0.        ,
       0.        , 0.        , 3.0825107 , 0.        , 0.        ,
       0.        , 2.8441463 , 1.392822  , 0.        , 1.053345  ,
       2.2318614 , 0.        , 0.8666289 , 0.        , 0.        ,
       0.        , 0.10659737, 0.        , 0.        , 0.        ,
       0.        , 0.        , 5.5952835 , 0.86208504, 2.0737646 ,
       0.        , 0.        , 0.        , 0.        , 4.9996853 ,
       0.13834059, 1.781134  , 3.8594704 , 0.        , 0.        ])
y37432 = 0 ['class:0']
```

Figure 2.2: Sample example for Data-set 1

Last, before proceeding with the application of classifier algorithms data-sets have been **split** in training set and test set by using the $\frac{2}{3}$ for training and $\frac{1}{3}$ for testing rule as follows:

- Data-set 1: `X1_train = 33 350`, `X1_test = 16 650` and `Y1_train`, `Y1_test`;
- Data-set 2: `X2_train = 33 350`, `X2_test = 16 650` and `Y2_train`, `Y2_test`;

like shown in the next figure.

```
First training sample.
    X1_train_0 = array([0.        , 5.8201475 , 0.        , 0.599935  , 2.4683702 ,
       0.        , 0.        , 3.0287304 , 0.        , 0.        ,
       0.28825417, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.42990297, 0.        , 3.4474413 , 0.3799751 ,
       0.37828726, 1.4302074 , 3.829352  , 2.059434  , 1.8962059 ,
       0.        , 3.6437218 , 2.1495094 , 2.4954965 , 0.8296877 ,
       0.        , 0.        , 0.        , 5.306748  , 0.        ,
       0.23007278, 3.3863564 , 0.        , 8.071031  , 0.        ,
       0.30797762, 0.912771  , 1.6363027 , 1.7607147 , 0.        ,
       2.1633503 , 0.        , 0.9634709 , 2.1906996 , 0.        ,
       0.        , 5.3184896 , 0.96073544, 4.386307  , 3.6712434 ,
       0.        , 0.        , 1.612142  , 3.520365  , 0.        ,
       0.        , 1.9248971 , 0.        , 0.        , 0.        ,
       3.3850787 , 0.        , 1.516204  , 1.7820904 , 3.1804948 ,
       0.41685063, 1.3082802 , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       1.3032343 , 0.        , 4.838368  , 0.        , 1.8475261 ,
       0.        , 3.8408687 , 2.1109064 , 7.094224  , 2.3411043 ,
       0.53535473, 0.        , 7.3551054 , 1.1085356 , 1.6881721 ,
       0.67120856, 3.3199337 , 7.4148364 , 0.        , 0.        ])
    Y1_train_0 = 8 ['class:8']
```

(a) First Training Sample for Data-set 1

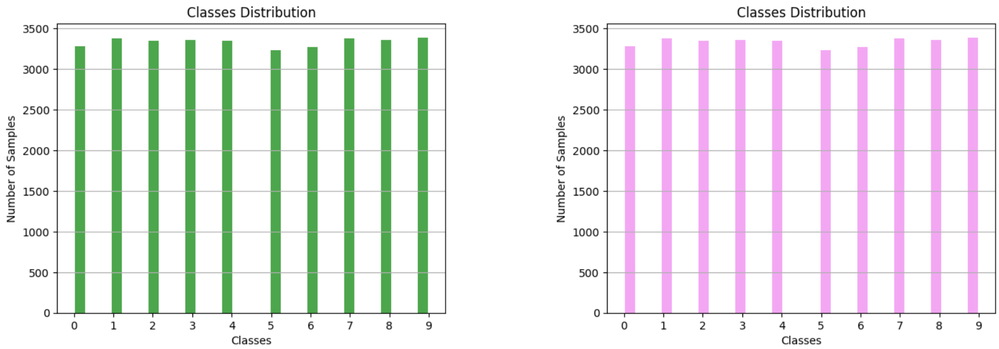
```
First test sample.
    X1_test_0 = array([0.20980507, 1.3535604 , 0.        , 2.0637276 , 3.748593  ,
       0.        , 0.        , 2.1363246 , 0.        , 0.7763224 ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 4.469309  , 0.        , 2.4846914 , 0.37974206,
       1.8277754 , 0.        , 0.        , 4.567697  , 2.7500098 ,
       0.        , 1.8390062 , 0.        , 3.3453805 , 0.        ,
       0.        , 0.        , 0.        , 6.1921763 , 0.        ,
       0.95278746, 1.8127362 , 2.0986607 , 0.        , 0.28177735,
       4.8344064 , 4.7709174 , 0.9442932 , 0.27956587, 0.        ,
       3.2208989 , 0.        , 0.08414418, 3.2619717 , 0.53714645,
       0.        , 0.        , 0.        , 2.3924844 , 0.9586686 ,
       0.        , 0.        , 0.13373508, 4.2655897 , 4.72677   ,
       1.684463  , 0.        , 0.        , 0.        , 0.        ,
       0.29633105, 0.        , 2.0892994 , 1.6338704 , 1.160551  ,
       0.        , 0.53534126, 0.        , 0.        , 0.9916296 ,
       2.1132782 , 0.        , 0.        , 0.        , 0.        ,
       0.36520627, 0.        , 0.        , 0.        , 2.0112069 ,
       2.6255965 , 3.0847576 , 1.2835877 , 2.4557698 , 0.24052368,
       3.142144  , 0.2877283 , 2.4557703 , 0.28008094, 1.7032326 ,
       0.        , 0.604147  , 0.        , 0.        , 0.        ])
    Y1_test_0 = 3 ['class:3']
```

(b) First Test Sample for Data-set 1

Figure 2.3: First Train and Test Samples for Data-set 1

Please note that only Data-set 1 examples have been shown in Fig.(2.2) and Fig.(2.3) because of the bigger size of Data-set 2.

Also observe in Fig.(2.4) that the Data-set 1 and Data-set 2 used for training are weakly unbalanced.



(a) Classes Distribution of the training set for Data-set 1 (`X1_train, Y1_train`)

(b) Classes Distribution of the training set for Data-set 2 (`X2_train, Y2_train`)

Figure 2.4: Classes Distribution of the training set for Data-set 1

Since this, follows my decision to not preprecess the data-sets to render them balanced.

After training and test one uses classification report, accuracy and confusion matrix to **evaluate** the best classification model for the data-sets.

Classification Report is a summary of the performance of a classification model that provides different metrics (such as precision, recall, f1-score and support) to evaluate the model's predictive ability on a specific data-set.

- Precision: measures the accuracy of positive predicted observations to the total predicted positive observations. High precision indicates a low false positive rate.
- Recall: measures the model's ability to correctly identify all positive instances to the actual positives in the data-set. High recall indicates a low false negative rate.
- F1-score: provides a measure of a model's accuracy that considers both the precision and recall of the model that represent the unbalance level for each class. A higher F1-score indicates better performing model for a particular class.
- Support: represents the number of samples in each class in the specified data-set. Classes with higher support values have more samples, while lower support values indicate fewer samples for that class.

Accuracy (`accuracy_score(Y_test, Y_predicted)`) provides a general understanding of how often the model's predictions are correct across all classes in a classification problem. It is computed as

$$accuracy = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

It prints as output also Macro Average (`Macro Avg`) and Weighted Average (`Weighted Avg`)

- `Macro Avg`: evaluates the overall performance of the classifier without considering class imbalance. Each class contributes equally to the final score. High Macro Avg score typically indicates that the model performs well across all classes without favoring or biasing towards any particular class.
- `Weighted Avg`: provides a better representation of the overall performance by considering the contribution of each class proportional to its size. High `Weighted Avg` score indicates that the model performs well across classes while accounting for class imbalance.

Confusion Matrix is a table that is used to describe and understand the performance of a classifier by presenting a comprehensive summary of the model's predictions versus the actual true values.

- True Positives (TP): The main diagonal elements of the matrix represent the number of instances correctly predicted for each class.
- True Negatives (TN): The number of instances that were correctly predicted as negative. In a multi-class confusion matrix, the term "True Negatives" is not directly applicable since it's a binary classification concept.
- False Positives (FP): The values in each row but not on the diagonal represent instances of other classes that were incorrectly predicted as belonging to that specific class.
- False Negatives (FN): The instances of the class being evaluated that were incorrectly classified as other classes. in a multi-class confusion matrix, the term "False Negatives" is not directly applicable as in binary classification.

2.1 Data-set 1

In the following sections it will be shown different models with different normalization techniques and using then hyperparameters tuning.

2.1.1 Decision Tree Classifier

After calling the function `dt1_classifier = tree.DecisionTreeClassifier()`, which creates a decision tree model for classification tasks, it has been created a pipeline for each normalization technique described in Ch.(1): Introduction. Thereafter the model has been trained on (`X1_train, Y1_train`) and then tested on (`X1_test, Y1_test`).

StandardScaler()

Fig.(2.5a) shows the expansion of the Decision Tree Classifier for Data-set1 with normalization `StandardScaler()`.

Fig.(2.5b) shows the classification report and the accuracy score of the considered model. Here one notices that

- Precision have high values which means low false positive rate, for example for class 0 is 0.99.
- Recall have high values which means low false negative rate for example for class 0 is 0.98.
- F1-score have high values which means very balanced classes and a well performing model as it was possible to see from Fig.(2.4a), for example for class 0 is 0.99.
- Support have almost the same value for all classes. Notice that class 0 = 1 720, class 5 = 1 769 and class 6 = 1 723 have highest values than the other classes.
- Accuracy of `0,976 036 036 036 036 1` and training time of `5,253 447 771 072 388` seconds.
- `Macro Avg`: 0.98 is high, then the model performs well and does not favours any particular class.
- `Weighted Avg`: 0.98 is high, then the model performs well taking into account imbalances which are very weak, as shown in Fig.(2.4a).

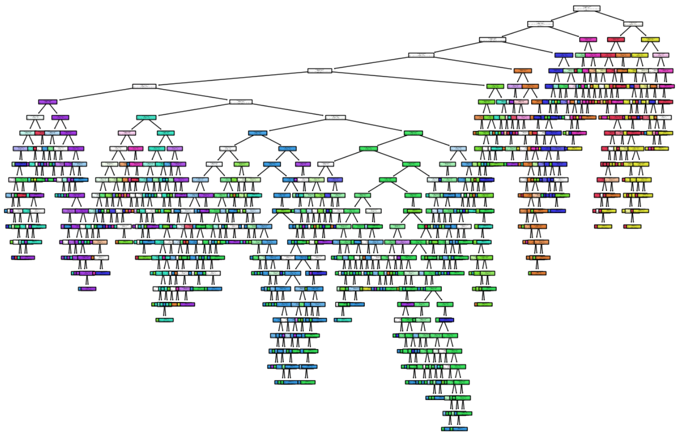
Observe that that the classification report for `QuantileTransformer()` (Fig.(2.6b)) and `Normalizer()` (Fig.(2.7b)) methods is almost the same, especially for the values of Support, `Macro Avg` and `Weighted Avg`. This is because these depend on the form of the data-set.

Fig.(2.5c) shows the confusion matrix from which one can notice that

- TP: in the main diagonal elements there are the number of instances correctly predicted, for example, in the cell corresponding to class 0, the value 1 692 represents instances of the said class that were correctly classified as class 0.
- TN: not possible to read.
- FP: the elements outside the main diagonal represent instances that were "confused" for another class, for example in the row corresponding to class 0, the values (\, 1, 10, 2, 1, 0, 4, 0, 9, 1) represent instances of other classes misclassified as class 0.
- FN: not possible to read.

Notice that one of the worse performing classes is class 3: 1 545 which is mostly mistaken as class 5 (61 times) and vice-versa (70 times). It has performing value equal to 0.93, recall equal to 0.94 and f1-score equal to 0.94.

Overall the performances of this model are very good.



(a) Decision Tree for Data-set 1 with normalization `StandardScaler()`

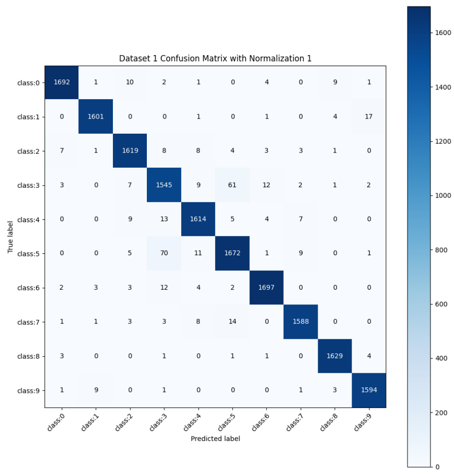
```
Training Set Evaluation on Dataset 1 using Decision Tree Classifier with Normalization 1:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      1720
           1       0.99      0.99      0.99      1624
           2       0.98      0.98      0.98      1654
           3       0.93      0.94      0.94      1642
           4       0.97      0.98      0.98      1652
           5       0.95      0.95      0.95      1769
           6       0.98      0.98      0.98      1723
           7       0.99      0.98      0.98      1618
           8       0.99      0.99      0.99      1639
           9       0.98      0.99      0.99      1609

    accuracy                           0.98     16650
   macro avg       0.98      0.98      0.98     16650
weighted avg       0.98      0.98      0.98     16650

Accuracy: 0.9760360360360361
```

(b) Class Report for Data-set 1 with normalization `StandardScaler()`



(c) Decision Matrix for Data-set 1 with normalization `StandardScaler()`

Figure 2.5: Visualization and Evaluation of Decision Tree Classifier for Data-set 1 with normalization `StandardScaler()`

QuantileTransformer(output_distribution = 'uniform')

Fig.(2.6a) shows the development of the Decision Tree Classifier for Data-set1 with normalization `QuantileTransformer(output_distribution = 'uniform')`.

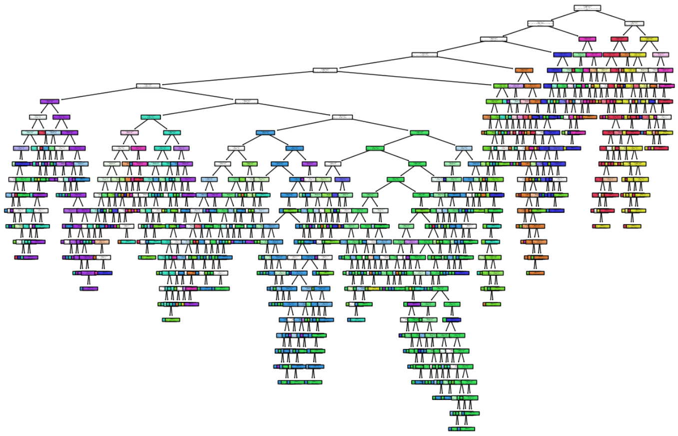
Fig.(2.6b) shows the classification report and the accuracy score of the considered model where it is possible to notice that

- Precision is the lowest for class 3: 0.93.
- Recall is the highest for class 8: 1.00 which has TP: 1 631.
- F1-score have lowest value for class 3: 0.93 as it was possible to observe from Fig.(2.4a).
- Accuracy of `0,976 816 816 816 816 9` and training time of `6,509 792 089 462 28` seconds.

Fig.(2.6c) shows the confusion matrix from which one can notice that

- TP: class 6: 1 698 represents instances of the said class that were correctly classified as class 6 and it has in fact the highest value of correct guesses.
- TN: not possible to read.
- FP: class 3: 1 545 is mostly mistaken for class 5: 1 672 (59 times) and vice-versa (70 times) which in fact have low precision and recall values.
- FN: not possible to read.

Overall the performances of this model are very good.



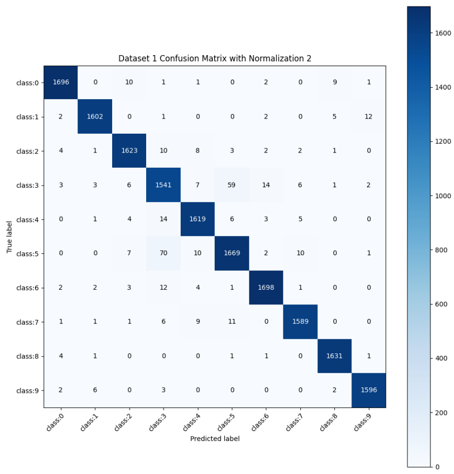
(a) Decision Tree for Data-set 1 with normalization `QuantileTransformer(output_distribution = 'uniform')`

Training Set Evaluation on Dataset 1 using Decision Tree Classifier with Normalization 2:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1720
1	0.99	0.99	0.99	1624
2	0.98	0.98	0.98	1654
3	0.93	0.94	0.93	1642
4	0.98	0.98	0.98	1652
5	0.95	0.94	0.95	1769
6	0.98	0.99	0.99	1723
7	0.99	0.98	0.98	1618
8	0.99	1.00	0.99	1639
9	0.99	0.99	0.99	1609
accuracy			0.98	16650
macro avg	0.98	0.98	0.98	16650
weighted avg	0.98	0.98	0.98	16650

Accuracy: 0.9768168168168169

(b) Class Report for Data-set 1 with normalization `QuantileTransformer()`



(c) Decision Matrix for Data-set 1 with normalization `QuantileTransformer()`

Figure 2.6: Visualization and Evaluation of Decision Tree Classifier for Data-set 1 with normalization `QuantileTransformer(output_distribution = 'uniform')`

Normalizer()

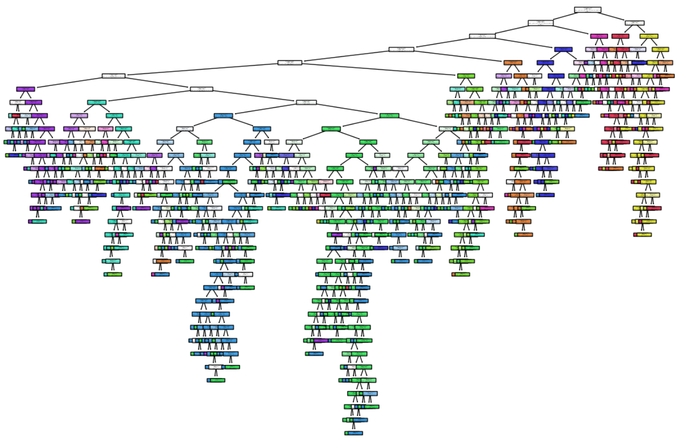
Fig.(2.7a) shows the development of the Decision Tree Classifier for Data-set1 with normalization `Normalizer()`.

Fig.(2.7b) and Fig.(2.7c) shows the classification report, the accuracy score and the confusion matrix of the considered model. It is possible to do the same considerations done on previous normalization techniques especially for class 3, class 5, class 6 and class 8. This is because they all work in the same data-set, that has been split in the same way as shown in Fig.(2.4a).

- Accuracy of `0,976 756 756 756 756 8` and training time of `5,334 468 603 134 155` seconds.

Notice once again that class 3: 1 534 is mostly "confused" as class 5: 1 688 (63 times) and vice-versa (57 times).

Overall the performances of this model are very good.



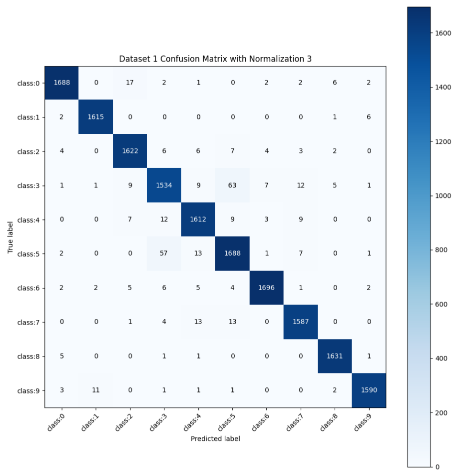
(a) Decision Tree for Data-set 1 with normalization `Normalizer()`

Training Set Evaluation on Dataset 1 using Decision Tree Classifier with Normalization 3:

	precision	recall	f1-score	support
0	0.99	0.98	0.99	1720
1	0.99	0.99	0.99	1624
2	0.98	0.98	0.98	1654
3	0.95	0.93	0.94	1642
4	0.97	0.98	0.97	1652
5	0.95	0.95	0.95	1769
6	0.99	0.98	0.99	1723
7	0.98	0.98	0.98	1618
8	0.99	1.00	0.99	1639
9	0.99	0.99	0.99	1609
accuracy			0.98	16650
macro avg	0.98	0.98	0.98	16650
weighted avg	0.98	0.98	0.98	16650

Accuracy: 0.9767567567567568

(b) Class Report for Data-set 1 with normalization `Normalizer()`



(c) Decision Matrix for Data-set 1 with normalization `Normalizer()`

Figure 2.7: Visualization and Evaluation of Decision Tree Classifier for Data-set 1 with normalization `Normalizer()`

In **conclusion** the decision tree classifier with different normalisation techniques have all more or less the same performances but the best model is:
`Decision Tree Classifier with QuantileTransformer(output distribution = 'uniform') with an accuracy of 0,976 816 816 816 816 9 and a training time of 6,509 792 089 462 28 seconds`.
The fastest model is:
`Decision Tree Classifier with StandardScaler() with an accuracy of 0,976 036 036 036`
and a training time of `5,253 447 771 072 388` seconds.

Hyperparameter Tuning

For Hyperparameter Tuning it has been implemented `RandomizedSearchCV` with number of iterations `n_iter = 10` and Cross-Validation `cv = 3`.

Let us compare the models with different normalization:

- `StandardScaler()` has best Decision Tree Classifier with { `"classifier__criterion": "gini", "classifier__max_depth": 12, "classifier__min_samples_leaf": 1, "classifier__min_samples_split": 2` } with accuracy of `0,979 459 459 459 459 4` and training time of `53,860 623 836 517 334` seconds.

- `QuantileTransformer(output_distribution = 'uniform')` has best Decision Tree Classifier with { `"classifier__criterion": "gini", "classifier__max_depth"` `12, "classifier__min_samples_leaf": 1, "classifier__min_samples_split": 2` } with accuracy of `0,980 360 360 360 360 3` and training time of `83,673 105 001 449 58` seconds.

- `Normalizer()` has best Decision Tree Classifier with { `"classifier__criterion": "gini", "classifier__max_depth": 11, "classifier__min_samples_leaf": 3, "classifier__min_samples_split": 9` } with accuracy of `0,980 660 660 660 660 7` and training time of `64,113 917 350 769 04` seconds.

In this case the decision tree classifier with different normalisation techniques performing hyperparameter tuning have all more or less the same performances but the best model is:
`Decision Tree Classifier with Normalizer() with an accuracy of 0,980 660 660 660 660`
`and a training time of 64,113 917 350 769 04 seconds`.
The fastest model is:
`Decision Tree Classifier with StandardScaler() with an accuracy of 0,979 459 459 459`
and a training time of `53,860 623 836 517 334` seconds.

The Best Decision Tree Model for Data-set 1

The best decision tree classifier for Data-set 1 is the one with hyperparameters:
{ `"classifier__criterion": "gini", "classifier__max_depth": 11, "classifier__min_s`
`3, "classifier__min_samples_split": 9` }
that uses `Normalizer()` normalization,
has accuracy of `0,980 660 660 660 660 7`
training and time of `64,113 917 350 769 04` seconds.
The fastest decision tree classifier for Data-set 1 is the one with no hyperparameter tuning that uses `StandardScaler()` normalization

has accuracy of `0,976 036 036 036 036 1` and a training time of `5,253 447 771 072 388` seconds.

Please observe that with hyperparameter tuning one has the best performances but slower training time, on the other hand without hyperparameter tuning one has worse performances but the fastest training time.

2.1.2 Support Vector Machine (SVM) Classifier

After calling the function `svm1_model = svm.SVC()`, which creates a Support Vector Classification (SVC) model, which is an implementation of the Support Vector Machine (SVM) algorithm for classification tasks, it has been created a pipeline for each normalization technique described in Ch.(1): Introduction. Thereafter the model has been trained on (`X1_train, Y1_train`) and then tested on (`X1_test, Y1_test`).

StandardScaler()

Fig.(2.8a) shows the classification report and the accuracy score of the considered model where it is possible to notice that

- Precision is the lowest for class 3: 0.97 but still very good performing.
- Recall have high values in general, for example for class 1: 1.00 which has TP: 1 616.
- F1-score have high values in general so very good performing. For example class 9: 1.00 with TP: 1 605.
- Support have almost the same value for all classes. Notice that class 0 = 1 720, class 5 = 1 769 and class 6 = 1 723 have highest values than the other classes.
- Accuracy of `0,989 309 309 309 309 3` and training time of `3,667 876 958 847 046` seconds.
- `Macro Avg` is high, then the model performs well and does not favours any particular class.
- `Weighted Avg` is high, then the model performs well taking into account imbalances which are very weak, as shown in Fig.(2.4a).

Observe that that the classification report for `QuantileTransformer()` (Fig.(2.9a)) and `Normalizer()` (Fig.(2.10a)) methods is almost the same, especially for the values of Support, `Macro Avg` and `Weighted Avg`. This is because these depend on the form of the data-set.

Fig.(2.8b) shows the confusion matrix from which one can notice that

- TP: for example class 6: 1 714 represents instances of the said class that were correctly classified as class 6.
- TN: not possible to read.
- FP: class 3 is mostly mistaken for class 5 (24) and vice-versa (36) which in fact have lower precision and recall values.

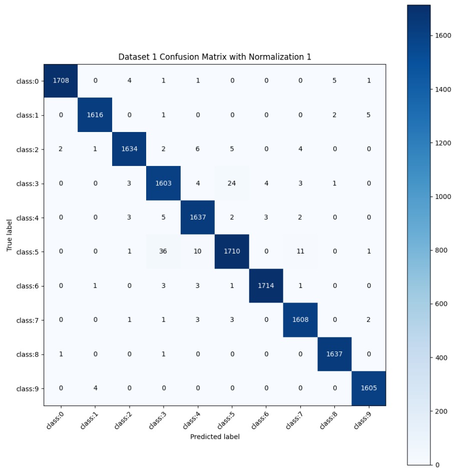
```
Training Set Evaluation on Dataset 1 using SMV with Normalization 1:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1720
           1       1.00      1.00      1.00      1624
           2       0.99      0.99      0.99      1654
           3       0.97      0.98      0.97      1642
           4       0.98      0.99      0.99      1652
           5       0.98      0.97      0.97      1769
           6       1.00      0.99      1.00      1723
           7       0.99      0.99      0.99      1618
           8       1.00      1.00      1.00      1639
           9       0.99      1.00      1.00      1609

    accuracy                           0.99     16650
   macro avg       0.99      0.99      0.99     16650
weighted avg       0.99      0.99      0.99     16650

Accuracy: 0.9893093093093093
```

(a) Class Report for Data-set 1 with normalization `StandardScaler()`



(b) Decision Matrix for Data-set 1 with normalization `StandardScaler()`

Figure 2.8: Evaluation of SVM Classification for Data-set 1 with normalization `StandardScaler()`

- FN: not possible to read.

Overall the performances of this model are almost perfect.

QuantileTransformer(output_distribution = 'uniform')

Fig.(2.9a) and Fig.(2.9b) shows the classification report, the accuracy score and the confusion matrix of the considered model. It is possible to do again the same considerations done on previous normalization techniques. This is because they all work in the same data-set, that has been split in the same way as shown in Fig.(2.4a).

- Accuracy of `0,987 987 987 987 988 8` and training time of `5,309 691 905 975 342` seconds.

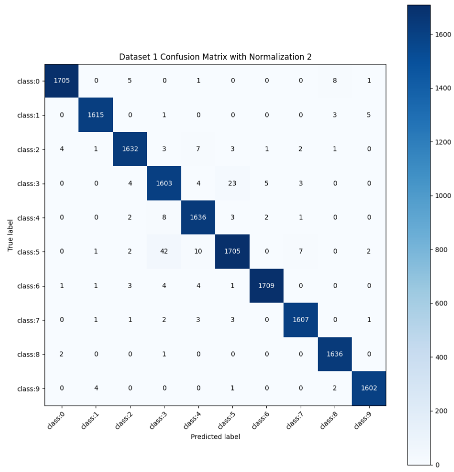
Notice again that class 3 is "confused" by class 5 (23) and vice-versa (42).
Overall the performances of this model are almost perfect.

Training Set Evaluation on Dataset 1 using SMV with Normalization 2:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	1720
1	1.00	0.99	0.99	1624
2	0.99	0.99	0.99	1654
3	0.96	0.98	0.97	1642
4	0.98	0.99	0.99	1652
5	0.98	0.96	0.97	1769
6	1.00	0.99	0.99	1723
7	0.99	0.99	0.99	1618
8	0.99	1.00	0.99	1639
9	0.99	1.00	1.00	1609
accuracy			0.99	16650
macro avg	0.99	0.99	0.99	16650
weighted avg	0.99	0.99	0.99	16650

Accuracy: 0.987987987987988

(a) Class Report for Data-set 1 with normalization
`QuantileTransformer()`



(b) Decision Matrix for Data-set 1 with normalization
`QuantileTransformer()`

Figure 2.9: Evaluation of SVM Classifier for Data-set 1 with normalization `QuantileTransformer(output_distribution = 'uniform')`

Normalizer()

Fig.(2.10a) and Fig.(2.10b) shows the classification report, the accuracy score and the confusion matrix of the considered model.

- Accuracy of 0,989 729 729 729 729 7 and training time of 2,357 007 026 672 363 3 seconds.

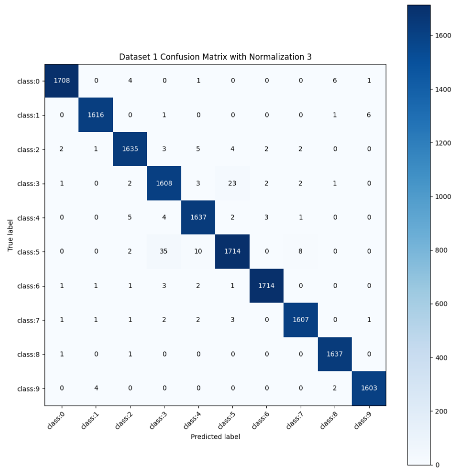
Notice again that class 3 is "confused" by class 5 (23) and vice-versa (35). Overall the performances of this model are almost perfect.

Training Set Evaluation on Dataset 1 using SMV with Normalization 3:

	precision	recall	f1-score	support
0	1.00	0.99	0.99	1720
1	1.00	1.00	1.00	1624
2	0.99	0.99	0.99	1654
3	0.97	0.98	0.98	1642
4	0.99	0.99	0.99	1652
5	0.98	0.97	0.97	1769
6	1.00	0.99	1.00	1723
7	0.99	0.99	0.99	1618
8	0.99	1.00	1.00	1639
9	1.00	1.00	1.00	1609
accuracy			0.99	16650
macro avg	0.99	0.99	0.99	16650
weighted avg	0.99	0.99	0.99	16650

Accuracy: 0.9897297297297297

(a) Class Report for Data-set 1 with normalization
`Normalizer()`



(b) Decision Matrix for Data-set 1 with normalization
`Normalizer()`

Figure 2.10: Evaluation of SVM Classifier for Data-set 1 with normalization `Normalizer()`

In **conclusion** the SVM classifier with different normalisation techniques have all more or less the same performances but the best model is:
`SVM Classifier with Normalizer() with an accuracy of 0,989 729 729 729 729 7 and a training time of 2,357 007 026 672 363 3 seconds.`
The fastest model happens to be the best model for SVM classifier.

Hyperparameter Tuning

For Hyperparameter Tuning it has been implemented `RandomizedSearchCV` with number of iterations `n_iter = 10` and Cross-Validation `cv = 3`.

Let us compare the models with different normalization:

- `StandardScaler()` has best SVM Classifier with { `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } with accuracy of `0,989 069 069 069 069 1` and training time of `118,774 129 629 135 13` seconds.

- `QuantileTransformer(output_distribution = 'uniform')` has best SVM Classifier with { `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } with accuracy of `0,987 807 807 807 807 9` and training time of `96,234 198 093 414 3` seconds.

- `Normalizer()` has best SVM Classifier with { `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } with accuracy of `0,989 909 909 909 909 9` and training time of `80,153 636 548 187 26` seconds.

In this case the SVM classifier with different normalisation techniques performing hyperparameter tuning have all more or less the same performances but the best model is:
`SVM Classifier with Normalizer() with an accuracy of 0,989 909 909 909 909 9 and a training time of 80,153 636 548 187 26 seconds.`
The fastest model happens to be the same as the best model for SVM Classifier.

The Best SVM Model for Data-set 1

The best SVM Classifier for Data-set 1 is the one with hyperparameters:
{ `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } that uses `Normalizer()` normalization,
has accuracy of `0,989 909 909 909 909 9` and training time of `80,153 636 548 187 26` seconds.
The fastest SVM Classifier for Data-set 1 is the one with no hyperparameter tuning that uses `SVM Classifier with Normalizer() with an accuracy of 0,989 729 729 729 729 7 and a training time of 2,357 007 026 672 363 3 seconds.`

Please observe that also with this model with hyperparameter tuning one has the best performances but slower training time, on the other hand without hyperparameter tuning one has worse performances but the fastest training time.

2.1.3 K-Nearest Neighbours (KNN) Classifier

After calling the function `knn1_model = KNeighboursClassifier()`, which makes predictions based on the majority class among its k-nearest neighbors in the feature space, it has been created a pipeline for each normalization technique described in Ch.(1): Introduction. Thereafter the model has been trained on (`X1_train, Y1_train`) and then tested on (`X1_test, Y1_test`).

StandardScaler()

Fig.(2.11a) shows the classification report and the accuracy score of the considered model where it is possible to notice that

- Precision is the lowest for class 3: 0.96.
- Recall is the lowest for class 3: 0.96 which has TP: 1 590.
- F1-score have lowest value for class 3: 0.96 and class 5: 0.96 as it was possible to observe from Fig.(2.4a).
- Support have almost the same value for all classes. Notice that class 0 = 1 720, class 5 = 1 769 and class 6 = 1 723 have highest values than the other classes.
- Accuracy of `0,986 606 606 606 606 6` and training time of `0,081 340 789 794 921 88` seconds.
- `Macro Avg` is high, then the model performs well and does not favours any particular class.
- `Weighted Avg` is high, then the model performs well taking into account imbalances which are very weak, as shown in Fig.(2.4a).

Observe that that again the classification report for `QuantileTransformer()` (Fig.(2.12a)) and `Normalizer()` (Fig.(2.13a)) methods is almost the same, especially for the values of Support, `Macro Avg` and `Weighted Avg`. This is because these depend on the form of the data-set.

Fig.(2.11b) shows the confusion matrix from which one can notice that

- TP: for example class 5: 1 693 represents instances of the said class that were correctly classified as class 5.
- TN: not possible to read;
- FP: in fact class 3 is mostly mistaken for class 5 (34) and vice-versa (50) which in fact have low precision and recall values.
- FN: not possible to read.

Overall the performances of this model are almost perfect.

QuantileTransformer(output_distribution = 'uniform')

Fig.(2.12a) and Fig.(2.12b) shows the classification report, the accuracy score and the confusion matrix of the considered model. It is possible to do again the same considerations done on previous normalization techniques. This is because they all work in the same data-set, that has been split in the same way as shown in Fig.(2.4a).

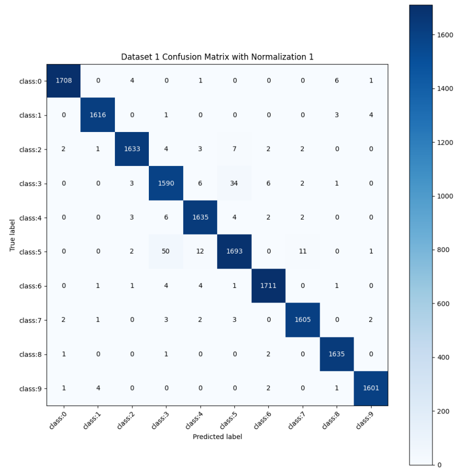
```
Training Set Evaluation on Dataset 1 using KNN with Normalization 1:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1720
           1       1.00      1.00      1.00      1624
           2       0.99      0.99      0.99      1654
           3       0.96      0.97      0.96      1642
           4       0.98      0.99      0.99      1652
           5       0.97      0.96      0.96      1769
           6       0.99      0.99      0.99      1723
           7       0.99      0.99      0.99      1618
           8       0.99      1.00      1.00      1639
           9       1.00      1.00      1.00      1609

    accuracy                           0.99     16650
   macro avg       0.99      0.99      0.99     16650
weighted avg       0.99      0.99      0.99     16650

Accuracy: 0.9866066066066066
```

(a) Class Report for Data-set 1 with normalization `StandardScaler()`



(b) Decision Matrix for Data-set 1 with normalization `StandardScaler()`

Figure 2.11: Evaluation of KNN Classification for Data-set 1 with normalization `StandardScaler()`

- Accuracy of `0,981 141 141 141 141 1` and training time of `0,787 373 781 204 223 6` seconds.

Notice once again that class 3 is mostly "confused" as class 5 (44) and vice-versa (68).

Overall the performances of this model are very good.

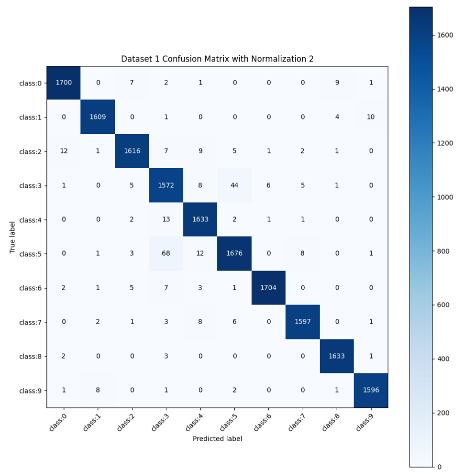
```
Training Set Evaluation on Dataset 1 using KNN with Normalization 2:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1720
           1       0.99      0.99      0.99      1624
           2       0.99      0.98      0.98      1654
           3       0.94      0.96      0.95      1642
           4       0.98      0.99      0.98      1652
           5       0.97      0.95      0.96      1769
           6       1.00      0.99      0.99      1723
           7       0.99      0.99      0.99      1618
           8       0.99      1.00      0.99      1639
           9       0.99      0.99      0.99      1609

    accuracy                           0.98     16650
   macro avg       0.98      0.98      0.98     16650
weighted avg       0.98      0.98      0.98     16650

Accuracy: 0.9811411411411411
```

(a) Class Report for Data-set 1 with normalization `QuantileTransformer()`



(b) Decision Matrix for Data-set 1 with normalization `QuantileTransformer()`

Figure 2.12: Evaluation of KNN Classifier for Data-set 1 with normalization `QuantileTransformer(output_distribution = 'uniform')`

Normalizer()

Fig.(2.13a) and Fig.(2.13b) shows the classification report, the accuracy score and the confusion matrix of the considered model.

- Accuracy of `0,987 567 567 567 567 6` and training time of `0,038 421 869 277 954 1` seconds.

Notice again that class 3 is "confused" by class 5 (28) and vice-versa (47). Overall the performances of this model are almost perfect.

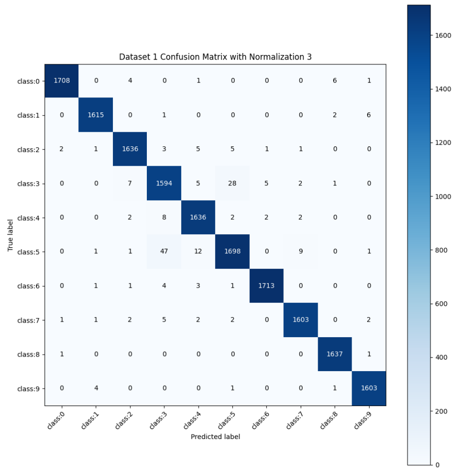
```
Training Set Evaluation on Dataset 1 using KNN with Normalization 3:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1720
           1       1.00      0.99      0.99      1624
           2       0.99      0.99      0.99      1654
           3       0.96      0.97      0.96      1642
           4       0.98      0.99      0.99      1652
           5       0.98      0.96      0.97      1769
           6       1.00      0.99      0.99      1723
           7       0.99      0.99      0.99      1618
           8       0.99      1.00      1.00      1639
           9       0.99      1.00      0.99      1609

    accuracy                           0.99     16650
   macro avg       0.99      0.99      0.99     16650
weighted avg       0.99      0.99      0.99     16650

Accuracy: 0.9875675675675676
```

(a) Class Report for Data-set 1 with normalization
`Normalizer()`



(b) Decision Matrix for Data-set 1 with normalization
`Normalizer()`

Figure 2.13: Evaluation of KNN Classifier for Data-set 1 with normalization `Normalizer()`

In **conclusion** the KNN classifier with different normalisation techniques have all more or less the same performances but the best model is:
`KNN Classifier with Normalizer() with an accuracy of 0,987 567 567 567 567 6 and a training time of 0,038 421 869 277 954 1 seconds.`
The fastest model happens to be the best model for KNN classifier.

Hyperparameter Tuning

For Hyperparameter Tuning it has been implemented `RandomizedSearchCV` with number of iterations `n_iter = 10` and Cross-Validation `cv = 3`.

Let us compare the models with different normalization:

- `StandardScaler()` has best KNN Classifier with { `"knn__n_neighbors": 12, "knn__weights": "uniform", "knn__metric": "manhattan",` } with accuracy of `0,988 228 228 228 228 2` and training time of `605,846 521 615 982` seconds.

- `QuantileTransformer(output_distribution = 'uniform')` has best KNN Classifier with { `"knn__n_neighbors": 12, "knn__weights": "uniform", "knn__metric": "manhattan",` } with accuracy of `0,986 426 426 426 426 4` and training time of `631,349 884 510 040 3` seconds.

- `Normalizer()` has best KNN Classifier with { `"knn__n_neighbors": 12, "knn__weights": "uniform", "knn__metric": "manhattan",` } with accuracy of `0,988 828 828 828 828 8` and training time of `611,407 084 226 608 3` seconds.

In this case the KNN Classifier with different normalisation techniques performing hyperparameter tuning have all more or less the same performances but the best model is:
`KNN Classifier with Normalizer()` with an accuracy of `0,988 828 828 828 828 8` and training time of `611,407 084 226 608 3` seconds.
The fastest model is `KNN Classifier with StandardScaler()` with an accuracy of `0,988 228 228 228 228 2` and training time of `605,846 521 615 982` seconds.

The Best KNN Model for Data-set 1

The best KNN Classifier for Data-set 1 is the one with hyperparameters:
{`"knn__n_neighbors": 12, "knn__weights": "uniform", "knn__metric": "manhattan"`}
that uses `Normalizer()` normalization,
has accuracy of `0,988 828 828 828 828 8` and training time of `611,407 084 226 608 3` seconds.
The fastest decision tree classifier for Data-set 1 is the one with no hyperparameter tuning that uses `KNN Classifier with Normalizer() with an accuracy of 0,987 567 567 567 567 6 and a training time of 0,038 421 869 277 954 1 seconds`.

Please observe that also with this model with hyperparameter tuning one has the best performances but slower training time, on the other hand without hyperparameter tuning one has worse performances but the fastest training time.

2.1.4 Best Classifier for Data-set 1

Overall the best selected model for Data-set 1 is SVM Classifier using `Normalizer()` normalization with hyperparameter tuning:
{`"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2`}
that has accuracy of `0,989 909 909 909 909 9` and training time of `80,153 636 548 187 26` seconds.
The fastest selected training time model is `KNN Classifier with Normalizer() with an accuracy of 0,987 567 567 567 567 6 and a training time of 0,038 421 869 277 954 seconds`.

Please observe that by using Hyperparameter Tuning one has to take into account the trade off between performances and training time.

From these results in fact one has better performances with hyperparameter tuning but faster training time without it.

With this type of data-set the differences between performances with and without hyperparameter tuning are on the order of decimals and so almost irrelevant.

With this best model one have performed Blind Test 1 evaluation and printed the results on a `.csv` file, how you can see in the code [Cartolano 2023].

2.2 Data-set 2

In the following sections it will be shown different models with different normalization techniques and using then hyperparameters tuning.

2.2.1 Decision Tree Classifier

After calling the function `dt2_classifier = tree.DecisionTreeClassifier()`, which creates a decision tree model for classification tasks, it has been created a pipeline for each normalization technique described in Ch.(1): Introduction. Thereafter the model has been trained on (`X2_train, Y2_train`) and then tested on (`X2_test, Y2_test`).

StandardScaler()

Fig.(2.14a) shows the expansion of the Decision Tree Classifier for Data-set 2 with normalization `StandardScaler()`.

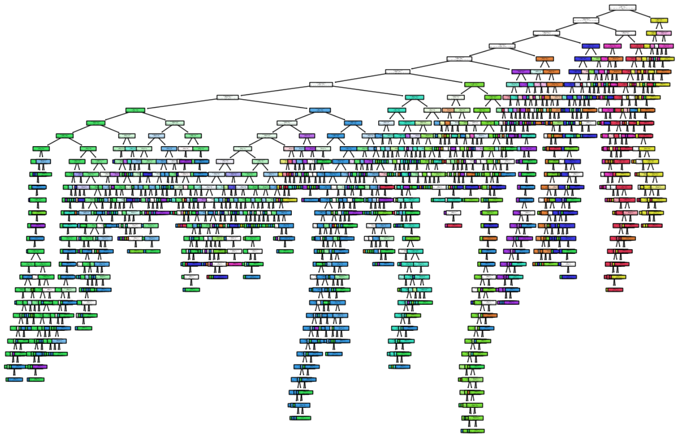
Fig.(2.14b) shows the classification report and the accuracy score of the considered model. Here one notices that

- Precision have high values which means low false positive rate, for example for class 0 is 0.98.
- Recall have high values which means low false negative rate for example for class 0 is 0.97.
- F1-score have high values which means very balanced classes and a well performing model as it was possible to see from Fig.(2.4b), for example for class 0 is 0.98.
- Support have almost the same value for all classes. Notice that class 0 = 1 720, class 5 = 1 769 and class 6 = 1 723 have highest values than the other classes like it happens for Data-set 1.
- Accuracy of `0,950 330 330 330 330 3` and training time of `111,780 118 703 842 16` seconds.
- `Macro Avg`: 0.95 is high, then the model performs well and does not favours any particular class.
- `Weighted Avg`: 0.95 is high, then the model performs well taking into account imbalances which are very weak, as shown in Fig.(2.4b).

Observe that that the classification report for `QuantileTransformer()` (Fig.(2.15b)) and `Normalizer()` (Fig.(2.16b)) methods is almost the same, especially for the values of Support, `Macro Avg` and `Weighted Avg`. This is because these depend on the form of the data-set.

Fig.(2.14c) shows the confusion matrix from which one can notice that

- TP: in the main diagonal elements there are the number of instances correctly predicted, for example, in the cell corresponding to class 0, the value 1 670 represents instances of the said class that were correctly classified as class 0.



(a) Decision Tree for Data-set 2 with normalization `StandardScaler()`

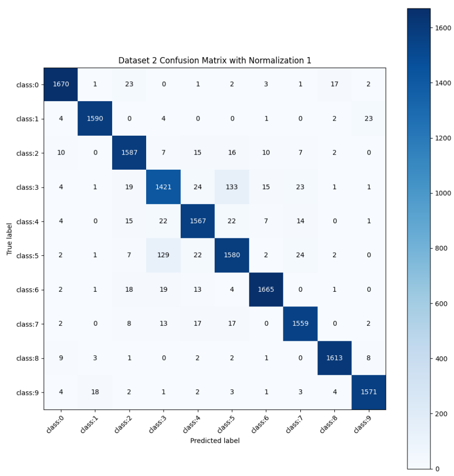
```
Training Set Evaluation on Dataset 2 using Decision Tree Classifier with Normalization 1:
              precision    recall  f1-score   support

           0       0.98      0.97      0.97      1720
           1       0.98      0.98      0.98      1624
           2       0.94      0.96      0.95      1654
           3       0.88      0.87      0.87      1642
           4       0.94      0.95      0.95      1652
           5       0.89      0.89      0.89      1769
           6       0.98      0.97      0.97      1723
           7       0.96      0.96      0.96      1618
           8       0.98      0.98      0.98      1639
           9       0.98      0.98      0.98      1609

    accuracy                           0.95     16650
   macro avg       0.95      0.95      0.95     16650
weighted avg       0.95      0.95      0.95     16650

Accuracy: 0.9503303303303303
```

(b) Class Report for Data-set 2 with normalization `StandardScaler()`



(c) Decision Matrix for Data-set 2 with normalization `StandardScaler()`

Figure 2.14: Visualization and Evaluation of Decision Tree Classifier for Data-set 2 with normalization `StandardScaler()`

- TN: not possible to read.
- FP: the elements outside the main diagonal represent instances that were "confused" with another class, for example in the row corresponding to class 0, the values (\, 1, 23, 0, 1, 2, 4, 1, 17, 2) represent instances of other classes misclassified as class 0.
- FN: not possible to read.

Notice that one of the worse performing classes is again class 3 which is mostly mistaken as class 5 (133 times) and vice-versa (129 times), has performing value equal to 0.88, recall equal to 0.87 and f1-score equal to 0.87.

Overall the performances of this model are very good.

QuantileTransformer(output_distribution = 'uniform')

Fig.(2.15a) shows the development of the Decision Tree Classifier for Data-set 2 with normalization `QuantileTransformer(output_distribution = 'uniform')`.

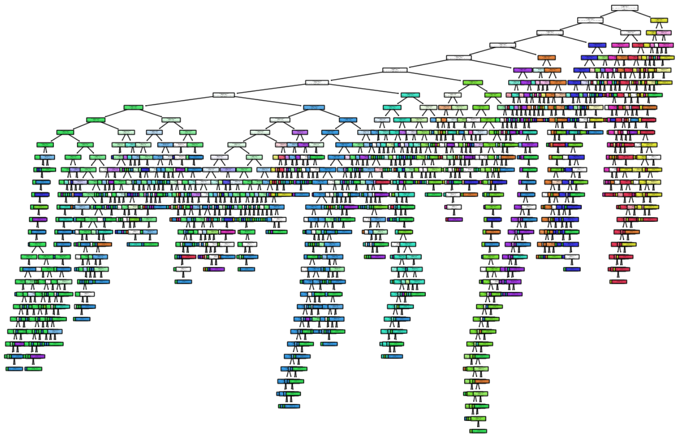
Fig.(2.15b) shows the classification report and the accuracy score of the considered model where it is possible to notice that

- Precision is the lowest for class 3: 0.87.
- Recall is the highest for class 8: 0.99 which has TP: 1 615.
- F1-score have lowest value for class 3: 0.87 as it was possible to observe from Fig.(2.4b).
- Accuracy of `0,950 990 990 990 990 9` and training time of `108,359 912 872 314 45` seconds.

Fig.(2.15c) shows the confusion matrix from which one can notice that

- TP: for example class 6: 1 681 represents instances of the said class that were correctly classified as class 6 and it has in fact the highest value of correct guesses.
- TN: not possible to read.
- FP: class 3 is mostly mistaken for class 5 (127) and vice-versa (131) which in fact have low precision and recall values.
- FN: not possible to read.

Overall the performances of this model are very good.



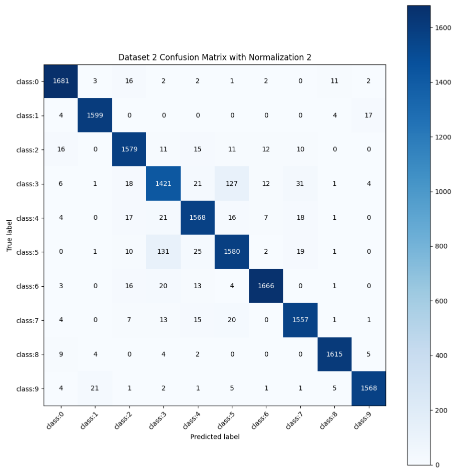
(a) Decision Tree for Data-set 2 with normalization `QuantileTransformer(output_distribution = 'uniform')`

Training Set Evaluation on Dataset 2 using Decision Tree Classifier with Normalization 2:

	precision	recall	f1-score	support
0	0.97	0.98	0.97	1720
1	0.98	0.98	0.98	1624
2	0.95	0.95	0.95	1654
3	0.87	0.87	0.87	1642
4	0.94	0.95	0.95	1652
5	0.90	0.89	0.89	1769
6	0.98	0.97	0.97	1723
7	0.95	0.96	0.96	1618
8	0.98	0.99	0.99	1639
9	0.98	0.97	0.98	1609
accuracy			0.95	16650
macro avg	0.95	0.95	0.95	16650
weighted avg	0.95	0.95	0.95	16650

Accuracy: 0.9509909909909909

(b) Class Report for Data-set 2 with normalization `QuantileTransformer()`



(c) Decision Matrix for Data-set 2 with normalization `QuantileTransformer()`

Figure 2.15: Visualization and Evaluation of Decision Tree Classifier for Data-set 2 with normalization `QuantileTransformer(output_distribution = 'uniform')`

Normalizer()

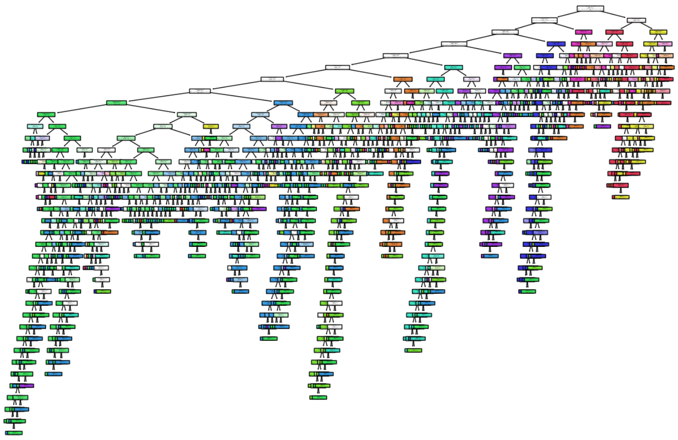
Fig.(2.16a) shows the development of the Decision Tree Classifier for Data-set 2 with normalization `Normalizer()`.

Fig.(2.16b) and Fig.(2.7c) shows the classification report, the accuracy score and the confusion matrix of the considered model. It is possible to do the same considerations done on previous normalization techniques especially for class 3, class 5, class 6 and class 8. This is because they all work in the same data-set, that has been split in the same way as shown in Fig.(2.4a).

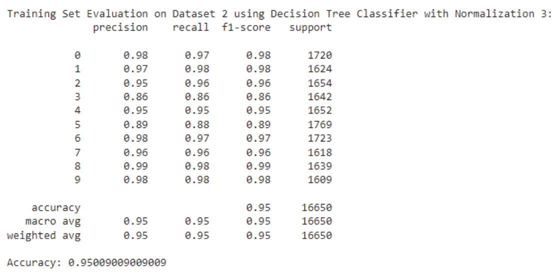
- Accuracy of `0,950 090 090 090 09` and training time of `68,616 325 139 999 39` seconds.

Notice once again that class 3 with TP: 1 418 is mostly "confused" as class 5 (146 times) and vice-versa (154 times).

Overall the performances of this model are very good.



(a) Decision Tree for Data-set 2 with normalization
`Normalizer()`

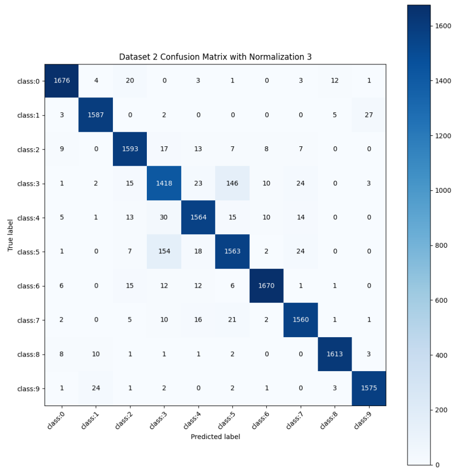


Training Set Evaluation on Dataset 2 using Decision Tree Classifier with Normalization 3:

	precision	recall	f1-score	support
0	0.98	0.97	0.98	1720
1	0.97	0.98	0.98	1624
2	0.95	0.96	0.96	1654
3	0.86	0.86	0.86	1642
4	0.95	0.95	0.95	1652
5	0.89	0.88	0.89	1769
6	0.98	0.97	0.97	1723
7	0.96	0.96	0.96	1618
8	0.99	0.98	0.99	1639
9	0.98	0.98	0.98	1609
accuracy			0.95	16650
macro avg	0.95	0.95	0.95	16650
weighted avg	0.95	0.95	0.95	16650

Accuracy: 0.95009009009009

(b) Class Report for Data-set 2 with normalization
`Normalizer()`



(c) Decision Matrix for Data-set 2 with normalization
`Normalizer()`

Figure 2.16: Visualization and Evaluation of Decision Tree Classifier for Data-set 2 with normalization `Normalizer()`

In **conclusion** the decision tree classifier with different normalisation techniques have all more or less the same performances but the best model is:
`Decision Tree Classifier with QuantileTransformer(output distribution = 'uniform') with an accuracy of 0,950 990 990 990 990 9 and a training time of 108,359 912 872 314 45 seconds`.
The fastest model is:
`Decision Tree Classifier with Normalizer() with an accuracy of 0,950 090 090 090 09` and a training time of 68,616 325 139 999 39 seconds.

Hyperparameter Tuning

For Hyperparameter Tuning it has been implemented `RandomizedSearchCV` with number of iterations `n_iter = 10` and Cross-Validation `cv = 3`.

Let us compare the models with different normalization:

- `StandardScaler()` has best Decision Tree Classifier with { `"classifier__criterion": "gini", "classifier__max_depth": 12, "classifier__min_samples_leaf": 1, "classifier__min_samples_split": 2` } with accuracy of `0,957 297 297 297 297 3` and training time of `547,149 544 000 625 6` seconds.

- `QuantileTransformer(output_distribution = 'uniform')` has best Decision Tree Classifier with { `"classifier__criterion": "gini", "classifier__max_depth"` `12, "classifier__min_samples_leaf": 1, "classifier__min_samples_split": 2` } with accuracy of `0,957 237 237 237 237 2` and training time of `770,375 849 962 234 5` seconds.

- `Normalizer()` has best Decision Tree Classifier with { `"classifier__criterion": "gini", "classifier__max_depth": 11, "classifier__min_samples_leaf": 3, "classifier__min_samples_split": 9` } with accuracy of `0,957 537 537 537 537 6` and training time of `506,349 608 421 325 7` seconds.

In this case the decision tree classifier with different normalisation techniques performing hyperparameter tuning have all more or less the same performances but the best model is:
`Decision Tree Classifier with Normalizer() with an accuracy of 0,957 537 537 537 537`
`and a training time of 506,349 608 421 325 7 seconds`.
The fastest model is:
`Decision Tree Classifier with Normalizer()` too

The Best Decision Tree Model for Data-set 2

The best decision tree classifier for Data-set 2 is the one with hyperparameters:
{ `"classifier__criterion": "gini", "classifier__max_depth": 11, "classifier__min_s`
`3, "classifier__min_samples_split": 9` }
that uses `Normalizer()` normalization,
has accuracy of `0,957 537 537 537 537 6`
training and time of `506,349 608 421 325 7` seconds.
The fastest decision tree classifier for Data-set 2 is the one with no hyperparameter tuning that uses `Normalizer()` normalization
has accuracy of `0,950 090 090 090 09` and a training time of `68,616 325 139 999 39` seconds. Please observe that, also with Data-set 2, with hyperparameter tuning one has the best performances but slower training time, on the other hand without hyperparameter tuning one has worse performances but the fastest training time.

2.2.2 Support Vector Machine (SVM) Classifier

After calling the function `svm2_model = svm.SVC()`, which creates a Support Vector Classification (SVC) model, which is an implementation of the Support Vector Machine (SVM) algorithm for classification tasks, it has been created a pipeline for each normalization technique described in Ch.(1): Introduction. Thereafter the model has been trained on (`X2_train, Y2_train`) and then tested on (`X2_test, Y2_test`).

StandardScaler()

Fig.(2.17a) shows the classification report and the accuracy score of the considered model where it is possible to notice that

- Precision is the lowest for class 3: 0.92 but still good performing.
- Recall have the lowest value for class 5: 0.92 which has TP: 1 632. This means it gets more false negatives than the other classes.
- F1-score have the lowest value for class 3: 0.92, so it is worse performing class.
- Support have almost the same value for all classes. Notice that class 0 = 1 720, class 5 = 1 769 and class 6 = 1 723 have highest values than the other classes.
- Accuracy of `0,972 252 252 252 252 2` and training time of `48,304 715 156 555 176` seconds.
- `Macro Avg`: 0.97 is high, then the model performs well and does not favours any particular class.
- `Weighted Avg`: 0.97 is high, then the model performs well taking into account imbalances which are very weak, as shown in Fig.(2.4b).

Observe that that the classification report for `QuantileTransformer()` (Fig.(2.18)) and `Normalizer()` (Fig.(2.19)) methods is almost the same, especially for the values of Support, `Macro Avg` and `Weighted Avg`. This is because these depend on the form of the data-set.

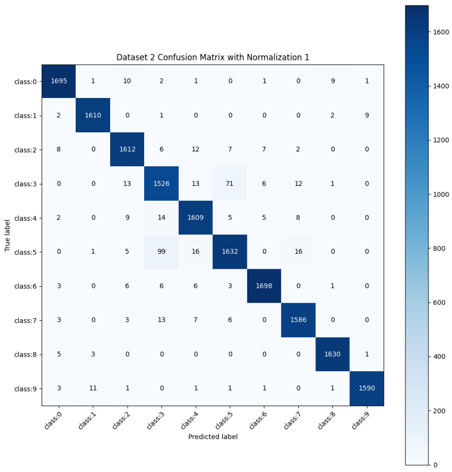
```
Training Set Evaluation on Dataset 2 using SMV with Normalization 1:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1720
           1       0.99      0.99      0.99      1624
           2       0.97      0.97      0.97      1654
           3       0.92      0.93      0.92      1642
           4       0.97      0.97      0.97      1652
           5       0.95      0.92      0.93      1769
           6       0.99      0.99      0.99      1723
           7       0.98      0.98      0.98      1618
           8       0.99      0.99      0.99      1639
           9       0.99      0.99      0.99      1609

    accuracy                           0.97     16650
   macro avg       0.97      0.97      0.97     16650
weighted avg       0.97      0.97      0.97     16650

Accuracy: 0.9722522522522522
```

(a) Class Report for Data-set 2 with normalization `StandardScaler()`



(b) Decision Matrix for Data-set 2 with normalization `StandardScaler()`

Figure 2.17: Evaluation of SVM Classification for Data-set 2 with normalization `StandardScaler()`

Fig.(2.17b) shows the confusion matrix from which one can notice that

- TP: for example class 6: 1 698 represents instances of the said class that were correctly classified as class 6.

- TN: not possible to read.
- FP: class 3: 1 526 is mostly mistaken for class 5 (71 times) and vice-versa (99 times) which in fact have lower precision and recall values.
- FN: not possible to read.

Overall the performances of this model are very good.

QuantileTransformer(output_distribution = 'uniform')

Fig.(2.18a) and Fig.(2.18b) shows the classification report, the accuracy score and the confusion matrix of the considered model. It is possible to do again the same considerations done on previous normalization techniques. This is because they all work in the same data-set, that has been split in the same way as shown in Fig.(2.4b).

- Accuracy of `0,972 192 192 192 192 2` and training time of `49,397 754 669 189 45` seconds.

Notice again that class 3: 1 523 is "confused" by class 5: 1 628 (73 times) and vice-versa (99 time).

Overall the performances of this model are very good.

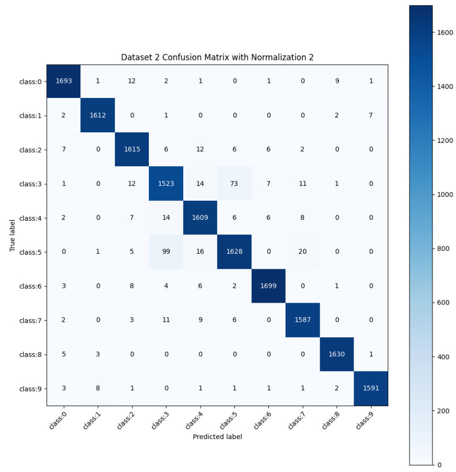
```
Training Set Evaluation on Dataset 2 using SMV with Normalization 2:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1720
           1       0.99      0.99      0.99      1624
           2       0.97      0.98      0.97      1654
           3       0.92      0.93      0.92      1642
           4       0.96      0.97      0.97      1652
           5       0.95      0.92      0.93      1769
           6       0.99      0.99      0.99      1723
           7       0.97      0.98      0.98      1618
           8       0.99      0.99      0.99      1639
           9       0.99      0.99      0.99      1609

    accuracy                           0.97     16650
   macro avg       0.97      0.97      0.97     16650
weighted avg       0.97      0.97      0.97     16650

Accuracy: 0.9721921921921922
```

(a) Class Report for Data-set 2 with normalization `QuantileTransformer()`



(b) Decision Matrix for Data-set 2 with normalization `QuantileTransformer()`

Figure 2.18: Evaluation of SVM Classifier for Data-set 2 with normalization `QuantileTransformer(output_distribution = 'uniform')`

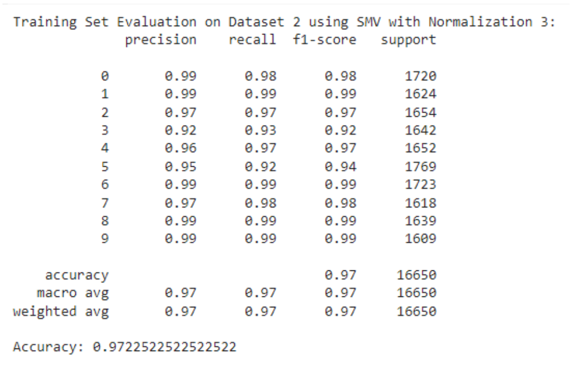
Normalizer()

Fig.(2.19a) and Fig.(2.19b) shows the classification report, the accuracy score and the confusion matrix of the considered model.

- Accuracy of `0,972 252 252 252 252 2` and training time of `35,268 704 652 786 255` seconds.

Notice again that class 3: 1 524 is "confused" with class 5: 1 633 (72 times) and vice-versa (96 times).

Overall the performances of this model are very good.

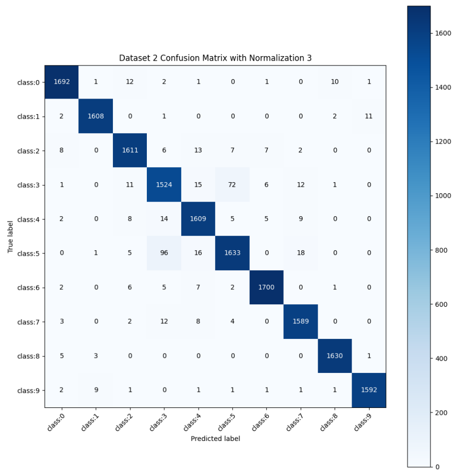


Training Set Evaluation on Dataset 2 using SMV with Normalization 3:

	precision	recall	f1-score	support
0	0.99	0.98	0.98	1720
1	0.99	0.99	0.99	1624
2	0.97	0.97	0.97	1654
3	0.92	0.93	0.92	1642
4	0.96	0.97	0.97	1652
5	0.95	0.92	0.94	1769
6	0.99	0.99	0.99	1723
7	0.97	0.98	0.98	1618
8	0.99	0.99	0.99	1639
9	0.99	0.99	0.99	1609
accuracy			0.97	16650
macro avg	0.97	0.97	0.97	16650
weighted avg	0.97	0.97	0.97	16650

Accuracy: 0.9722522522522522

(a) Class Report for Data-set 2 with normalization `Normalizer()`



(b) Decision Matrix for Data-set 2 with normalization `Normalizer()`

Figure 2.19: Evaluation of SVM Classifier for Data-set 2 with normalization `Normalizer()`

In **conclusion** the SVM classifier with different normalisation techniques have all more or less the same performances but the best model is:
`SVM Classifier with Normalizer()` and `SVM Classifier StandardScaler() with an accuracy of 0,972 252 252 252 252 2 equal and a training time of 35,268 704 652 786 255 seconds` and `48,304 715 156 555 176` seconds each.
The fastest model is `SVM Classifier with Normalizer()`.

Hyperparameter Tuning

For Hyperparameter Tuning it has been implemented `RandomizedSearchCV` with number of iterations `n_iter = 10` and Cross-Validation `cv = 3`.

Let us compare the models with different normalization:

- `StandardScaler()` has best SVM Classifier with { `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } with accuracy of `0,973 273 273 273 273 3` and training time of `1 427,270 799 398 422 2` seconds.

- `QuantileTransformer(output_distribution = 'uniform')` has best SVM Classifier with { `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } with accuracy of `0,971 831 831 831 831 9` and training time of `1 289,174 940 109 253` seconds.

- `Normalizer()` has best SVM Classifier with { `"svm__C": 1,660 186 404 424 365 3, "svm__kernel": linear, "svm__degree": 4,` } with accuracy of `0,972 492 492 492 492 5` and training time of `923,951 044 797 897 3` seconds.

In this case the SVM classifier with different normalisation techniques performing hyperparameter tuning have all more or less the same performances but the best model is:
`StandardScaler()` has best SVM Classifier with { `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } with accuracy of `0,973 273 273 273 273 3` and training time of `1 427,270 799 398 422 2` seconds.
The fastest model happens to be `Normalizer()` has best SVM Classifier with { `"svm__C": 1,660 186 404 424 365 3, "svm__kernel": linear, "svm__degree": 4,` } with accuracy of `0,972 492 492 492 492 5` and training time of `923,951 044 797 897 3` seconds.

The Best SVM Model for Data-set 2

The best SVM Classifier for Data-set 2 is:
`StandardScaler()` has best SVM Classifier with { `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } with accuracy of `0,973 273 273 273 273 3` and training time of `1 427,270 799 398 422 2` seconds.
The fastest SVM Classifier for Data-set 2 is the one with no hyperparameter tuning that uses `SVM Classifier with Normalizer()` with an accuracy of `0,972 252 252 252 252 2` and training time of `35,268 704 652 786 255` seconds.

Please observe that also with this model with hyperparameter tuning one has the best performances but slower training time, on the other hand without hyperparameter tuning one has worse performances but the fastest training time.

2.2.3 K-Nearest Neighbours (KNN) Classifier

After calling the function `knn2_model = KNeighboursClassifier()`, which makes predictions based on the majority class among its k-nearest neighbors in the feature space, it has been created a pipeline for each normalization technique described in Ch.(1): Introduction. Thereafter the model has been trained on (`X2_train, Y2_train`) and then tested on (`X2_test, Y2_test`).

StandardScaler()

Fig.(2.20a) shows the classification report and the accuracy score of the considered model where it is possible to notice that

- Precision is the lowest for class 3: 0.90, which is the lowest for StandardScaler() between the methods.
- Recall is the lowest for class 3: 0.91 which has TP: 1 499.
- F1-score have lowest value for class 3: 0.91 and class 5: 0.95 as it was possible to observe from Fig.(2.4b).
- Support have almost the same value for all classes. Notice that class 0 = 1 720, class 5 = 1 769 and class 6 = 1 723 have highest values than the other classes.

- Accuracy of `0,967 867 867 867 867 8` and training time of `0,449 068 854 629 516 6` seconds.

- `Macro Avg`: 0.97 is high, then the model performs well and does not favours any particular class.

- `Weighted Avg`: 0.97 is high, then the model performs well taking into account imbalances which are very weak, as shown in Fig.(2.4b).

Observe that that again the classification report for `QuantileTransformer()` (Fig.(2.21)) and `Normalizer()` (Fig.(2.22)) methods is almost the same, especially for the values of Support, `Macro Avg` and `Weighted Avg`. This is because these depend on the form of the data-set.

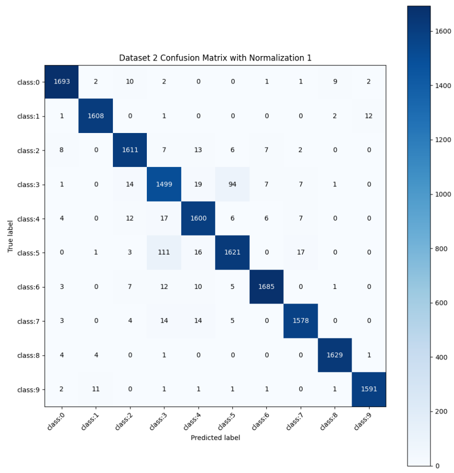
```
Training Set Evaluation on Dataset 2 using KNN with Normalization 1:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1720
           1       0.99      0.99      0.99      1624
           2       0.97      0.97      0.97      1654
           3       0.90      0.91      0.91      1642
           4       0.96      0.97      0.96      1652
           5       0.93      0.92      0.92      1769
           6       0.99      0.98      0.98      1723
           7       0.98      0.98      0.98      1618
           8       0.99      0.99      0.99      1639
           9       0.99      0.99      0.99      1609

    accuracy                           0.97     16650
   macro avg       0.97      0.97      0.97     16650
weighted avg       0.97      0.97      0.97     16650

Accuracy: 0.9678678678678678
```

(a) Class Report for Data-set 2 with normalization `StandardScaler()`



(b) Decision Matrix for Data-set 2 with normalization `StandardScaler()`

Figure 2.20: Evaluation of KNN Classification for Data-set 2 with normalization `StandardScaler()`

Fig.(2.20b) shows the confusion matrix from which one can notice that

- TP: for example class 5: 1 621 represents instances of the said class that were correctly classified as class 5.

- TN: not possible to read;

- FP: in fact class 3: 1 499 is mostly mistaken for class 5: 1 621 (94 times) and vice-versa (111 times) which in fact have low precision and recall values.

- FN: not possible to read.

Overall the performances of this model are very good.

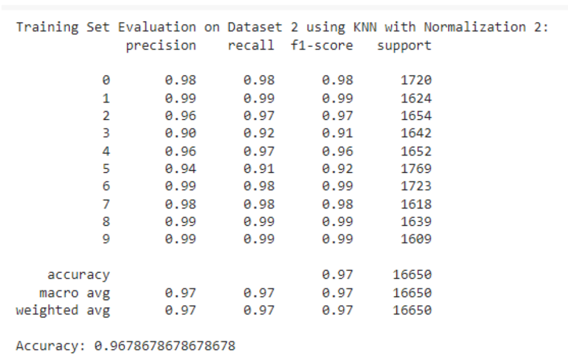
QuantileTransformer(output_distribution = 'uniform')

Fig.(2.21a) and Fig.(2.21b) shows the classification report, the accuracy score and the confusion matrix of the considered model. It is possible to do again the same considerations done on previous normalization techniques. This is because they all work in the same data-set, that has been split in the same way as shown in Fig.(2.4b).

- Accuracy of `0,967 867 867 867 867 8` and training time of `8,782 832 145 690 918` seconds.

Notice once again that class 3: 1 509 is mostly "confused" with class 5: 1 605 (79 times) and vice-versa (118 times).

Overall the performances of this model are very good.



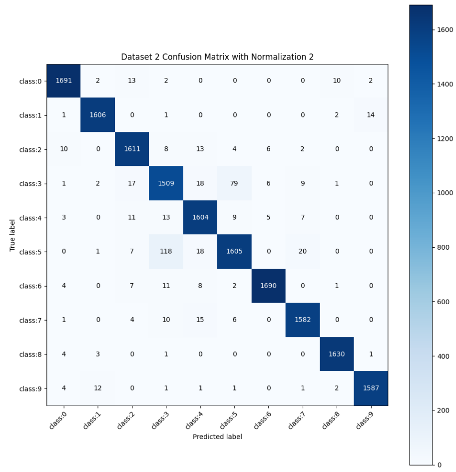
```
Training Set Evaluation on Dataset 2 using KNN with Normalization 2:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1720
           1       0.99      0.99      0.99      1624
           2       0.96      0.97      0.97      1654
           3       0.90      0.92      0.91      1642
           4       0.96      0.97      0.96      1652
           5       0.94      0.91      0.92      1769
           6       0.99      0.98      0.99      1723
           7       0.98      0.98      0.98      1618
           8       0.99      0.99      0.99      1639
           9       0.99      0.99      0.99      1609

    accuracy                           0.97     16650
   macro avg       0.97      0.97      0.97     16650
weighted avg       0.97      0.97      0.97     16650

Accuracy: 0.9678678678678678
```

(a) Class Report for Data-set 2 with normalization
`QuantileTransformer()`



(b) Decision Matrix for Data-set 2 with normalization
`QuantileTransformer()`

Figure 2.21: Evaluation of KNN Classifier for Data-set 2 with normalization `QuantileTransformer(output_distribution = 'uniform')`

Normalizer()

Fig.(2.22a) and Fig.(2.22b) shows the classification report, the accuracy score and the confusion matrix of the considered model.

- Accuracy of `0,969 249 249 249 249 3` and training time of `0,249 870 061 874 389 65` seconds.

Notice again that class 3: 1 506 is "confused" by class 5: 1 618 (80 times) and vice-versa (112 times).

Overall the performances of this model are very good.

In **conclusion** the KNN classifier with different normalisation techniques have all more or less the same performances but the best model is:
`KNN Classifier with Normalizer() with an accuracy of 0,969 249 249 249 249 3 and training time of 0,249 870 061 874 389 65 seconds.`
The fastest model happens to be the best model for KNN classifier.

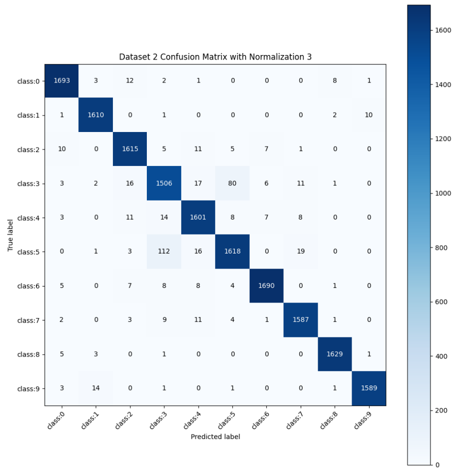
```
Training Set Evaluation on Dataset 1 using KNN with Normalization 3:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1720
           1       0.99      0.99      0.99      1624
           2       0.97      0.98      0.97      1654
           3       0.91      0.92      0.91      1642
           4       0.96      0.97      0.97      1652
           5       0.94      0.91      0.93      1769
           6       0.99      0.98      0.98      1723
           7       0.98      0.98      0.98      1618
           8       0.99      0.99      0.99      1639
           9       0.99      0.99      0.99      1609

    accuracy                           0.97     16650
   macro avg       0.97      0.97      0.97     16650
weighted avg       0.97      0.97      0.97     16650

Accuracy: 0.9692492492492493
```

(a) Class Report for Data-set 2 with normalization
`Normalizer()`



(b) Decision Matrix for Data-set 2 with normalization
`Normalizer()`

Figure 2.22: Evaluation of KNN Classifier for Data-set 2 with normalization `Normalizer()`

Hyperparameter Tuning

For Hyperparameter Tuning it has been implemented `RandomizedSearchCV` with number of iterations `n_iter = 10` and Cross-Validation `cv = 3`.

Let us compare the models with different normalization:

- `StandardScaler()` has best KNN Classifier with { `"knn__n_neighbors": 12, "knn__weights": "uniform", "knn__metric": "manhattan",` } with accuracy of 0,970 810 810 810 810 8 and training time of 53 790 372 145 652 771 seconds.
- `QuantileTransformer(output_distribution = 'uniform')` has best KNN Classifier with { `"knn__n_neighbors": 12, "knn__weights": "uniform", "knn__metric": "manhattan",` } with accuracy of 0,970 090 090 090 091 and training time of 5 610,021 097 898 483 seconds.
- `Normalizer()` has best KNN Classifier with { `"knn__n_neighbors": 10, "knn__weights": "distance", "knn__metric": "euclidean",` } with accuracy of 0,971 051 051 051 055 1 and training time of 5 371,534 497 976 303 seconds.

In this case the KNN Classifier with different normalisation techniques performing hyperparameter tuning have all more or less the same performances but the best model is:
`KNN Classifier with Normalizer()` with an accuracy of 0,971 051 051 051 055 1 and training time of 5 371,534 497 976 303 seconds.
The fastest model happens to be the best model.

The Best KNN Model for Data-set 2

The best KNN Classifier for Data-set 2 is:
`{"knn__n_neighbors": 12, "knn__weights": "uniform", "knn__metric": "manhattan"}`

that uses `Normalizer()` normalization,
has accuracy of `0,988 828 828 828 828 8` and training time of `611,407 084 226 608 3` seconds.
The fastest decision tree classifier for Data-set 1 is the one with no hyperparameter tuning that uses `KNN Classifier with Normalizer() with an accuracy of 0,969 249 249 249 249 3 and training time of 0,249 870 061 874 389 65 seconds`.

Please observe that also with this model with hyperparameter tuning one has the best performances but slower training time, on the other hand without hyperparameter tuning one has worse performances but the fastest training time.

2.2.4 Best Classifier for Data-set 2

Overall the best selected model for Data-set 2 is `StandardScaler()` has best SVM Classifier with { `"svm__C": 0,680 836 121 681 994 6, "svm__kernel": poly, "svm__degree": 2,` } with accuracy of `0,973 273 273 273 273 3` and training time of `1 427,270 799 398 422 2` seconds.
The fastest selected training time model is `KNN Classifier with Normalizer() with an accuracy of 0,969 249 249 249 249 3 and training time of 0,249 870 061 874 389 65 seconds`.

Please observe again that by using Hyperparameter Tuning one has to take into account the trade off between performances and training time.

From these results in fact one has better performances with hyperparameter tuning but faster training time without it.

With this type of data-set the differences between performances with and without hyperparameter tuning are on the order of decimals and so almost irrelevant.

With this best model one have performed Blind Test 2 evaluation and printed the results on a `.csv` file, how you can see in the code [Cartolano 2023].

Chapter 3

Conclusion

Given Data-set 1 and Data-set 2 this report has shown that in both cases one can draw more or less the same conclusions.

First, from the three methods of Classification one can see that in Data-set 1:

- *Decision Tree best classifier* is the one that uses `Normalizer()` normalization with hyperparameters:
  `{ "classifier__criterion": "gini",`
  `"classifier__max_depth": 11,`
  `"classifier__min_samples_leaf": 3,`
  `"classifier__min_samples_split": 9 }`
  has **accuracy** of 0,980 660 660 660 660 7
  and **training time** of 64,113 917 350 769 04 seconds.

- *Support Virtual Machine best classifier* (SVM) is the one that uses `Normalizer()` normalization with hyperparameters:
  `{ "svm__C": 0,680 836 121 681 994 6,`
  `"svm__kernel": poly,`
  `"svm__degree": 2 }`
  has **accuracy** of 0,989 909 909 909 909 9
  and **training time** of 80,153 636 548 187 26 seconds.

- *K-Nearest Neighbours best Classifier* (KNN) is the one that uses `Normalizer()` normalization, with hyperparameters
  `{"knn__n_neighbors": 12,`
  `"knn__weights": "uniform",`
  `"knn__metric": "manhattan"}`
  has **accuracy** of 0,988 828 828 828 828 8
  and **training time** of 611,407 084 226 608 3 seconds.

So the best Classifier for Data-set 1 is **SVM Classifier** using `Normalizer()` normalization with hyperparameters
`{"svm__C": 0,680 836 121 681 994 6,`
`"svm__kernel": poly,`
`"svm__degree": 2}`
that has accuracy of 0,989 909 909 909 909 9 and training time of 80,153 636 548 187 26 seconds.

It is possible to observe that one has the fastest training time for Data-set 1 if performing `KNN Classifier with Normalizer()` with an accuracy of `0,987 567 567 567 567 6` and a training time of 0,038 421 869 277 954 1 seconds.

For Data-set 2:

- *Decision Tree best classifier* is the one that uses `Normalizer()` normalization with hyperparameters
  `{ "classifier__criterion": "gini",`
  `"classifier__max_depth": 11,`
  `"classifier__min_samples_leaf": 3,`
  `"classifier__min_samples_split": 9 }`
  has **accuracy** of `0,957 537 537 537 537 6`
  and **training time** of `506,349 608 421 325 7` seconds.

- *SVM best classifier* is the one that uses `StandardScaler()` has best SVM Classifier with hyperparameters
  `{ "svm__C": 0,680 836 121 681 994 6,`
  `"svm__kernel": poly,`
  `"svm__degree": 2 }`
  with **accuracy** of `0,973 273 273 273 273 3`
  and **training time** of `1 427,270 799 398 422 2` seconds.

- *SVM best classifier* is the one that uses `Normalizer()` normalization, with hyperparameters
  `{"knn__n_neighbors": 12,`
  `"knn__weights": "uniform",`
  `"knn__metric": "manhattan"}`
  has **accuracy** of `0,988 828 828 828 828 8`
  and **training time** of `611,407 084 226 608 3` seconds.

Follows that the best classifier for Data-set 2 is **SVM Classifier** using `StandardScaler()` with hyperparameters
`{ "svm__C": 0,680 836 121 681 994 6,`
`"svm__kernel": poly,`
`"svm__degree": 2 }`
and accuracy of `0,973 273 273 273 273 3`
and training time of `1 427,270 799 398 422 2` seconds.

Please observe that one has the fastest training time for Data-set 2 if performing `KNN Classifier with Normalizer()` with an accuracy of `0,969 249 249 249 249 3` and training time of `0,249 870 061 874 389 65` seconds.

Moreover one notices that Data-set 1 performs slightly better than Data-set 2 because the latter is way larger then the first one.

When using Hyperparameter Tuning one needs to consider a trade off between performances and training time. In fact from the results in Chap.(2.1) and Chap.(2.2) one has better performances with hyperparameter tuning but very slow training time but since the differences in performances are very law and in general all the models have very good performances one can think of using the fastest models instead on the one using hyperparameter tuning.

For all three normalization techniques, independently of the data-set used and the classification model, the one can say that:

- Precision is the lowest for class 3.
- Recall is the lowest for class 3.
- F1-score have lowest value for class 3, in fact this is a consequence of the unbalance on the data-sets shown in Fig.(2.4a) and Fig.(2.4a).
- Support have almost the same value for all classes. Notice that class 0 = 1 720, class 5 = 1 769 and class 6 = 1 723 have highest values than the other classes.
- `Macro Avg`: is high for all cases, then the model performs well and does not favours any particular class.
- `Weighted Avg`: is high for all cases, then the model performs well taking into account imbalances which are very weak, as shown in Fig.(2.4a) and Fig.(2.4b).

Looking at the Confusion Matrices of both data-sets

- TP: represents instances of the considered class that were correctly classified as the considered class.
- TN: not possible to read.
- FP: represents instances of the considered class that were "confused" with other classes. Notice that for all data-sets, independently of the classification method and the normalization technique, class 3 always is mostly mistaken for class 5 and vice-versa which in fact have lower precision and recall values.
- FN: not possible to read.

In conclusion with these type of data-sets the differences between the considered models are on the order of decimals and so almost irrelevant which means that any model considered is a very good performing model.

Bibliography

Cartolano, Ludovica (2023). *Cartolano_1796046_HW1.py*. Python code for ML HW1.