1 Wine

Python 3.8 and sklearn 0.23.1 was used for this part of the exercise.

1.1 Decision Tree

1.1.1 Naive Comparison of Gini and Entropy

The initial Decision Tree Model, where all default parameters were used, with *test_size=0.1* and *random_state=42* yielded the following result:

Class	Precision	Recall	f1-Score
class_0	1.00	0.86	0.92
class_1	0.78	1.00	0.88
class_2	1.00	0.75	0.86

Figure 1: Classification report for the initial Decision Tree with an Accuracy of 89%

Using the parameter *criterion="entropy"* instead of the default "gini" for the DecisionTreeClassifier yields the following results:

Class	Precision	Recall	f1-Score
class_0	0.88	1.00	0.93
class_1	0.78	1.00	0.88
class_2	1.00	0.25	0.40

Figure 2: Classification report for an Decision Tree with the Criterion "entroy" and an Accuracy of 83%

This specific tree suffers in overall accuracy when using the "entropy" criterion instead of "gini" for splitting, as the accuracy drops in 6That does not lead to the conclusion that the "gini" criterion is overall better than "entropy" as they may lead to different best performances when using different parameters not only on the DecisionTreeClassifier but also maybe on the way the test data is generated.

1.1.2 Varying the Test Sizes

The following Experiments measured the Accuracies for *criterion="gini"* and *criterion="entropy"* respectively for an increasing test data size.

Test Size	Accuracy for Gini	Accuracy for Entropy
0.1	0.89	0.83
0.2	0.94	0.91
0.3	0.96	0.85
0.4	0.93	0.83

Figure 3: Accuracies for different Information Gain criteria and increasing test sizes

Using different training sizes do lead to different Accuracies as one would expect. While the "entropy" criterion does perform worse than the "gini" criterion with the same test sizes it also does have a different "optimum" for test size, around 20% for this dataset, where the accuracy its at its maximum, while "gini" has its maximum at around 30% test size.

1.1.3 Crossvalidation

Integrating the given snippet for cross validation with the DecisionTreeClassifier and using 10-fold Cross Validation with an "accuracy" scoring algorithm leads to an accuracy of $86.5\% \pm 9.135$. This mean is lower than the accuracy of a manually splitted dataset, but the deviation is very broad and could include the previous manually found optimum for test size.

Iteration	Accuracy
0	0.88888889
1	0.88888889
2	0.66666667
3	0.88888889
4	0.83333333
5	0.83333333
6	0.94444444
7	0.94117647
8	0.76470588

Figure 4: Accuracies for different Split Iterations of this 10-fold Crossvalidation

1.1.4 Grid Search

The following the Experiment searches for the best Hyperparameter for *criterion* and *random_state* for a DecisionTreeClassifier using a GridSearch and 10-fold Crossvalidation. For the following Gridsearch a range of 0-49 (inclusive) is used for the random_state parameter and "gini" or "entropy" for criterion.

Unexpectedly it found the best Accuracy of 92% for the Hyperparameter *criterion="entropy"* and *random_state=2*.
Using Gridsearch for only random_state and the default "gini" criterion yields an Accuracy of 88% for a random_state of 21.
The unexpected results could be explained by the fact that Gridsearch of sklearn uses crossvalidation, which may lead to underperformance with the "gini" criterion.
This experiment is not really useful, as the random_state is used for splitting the data into training and test set and functions as a random seed for selecting the data.

1.2 MLP

1.2.1 Comparing "adam" against "sgd"

The following experiments compares the Accuracy of the MLP-Network with two hidden layers each having 10 hidden nodes and a maximum iteration of 10 for the solvers "adam" and "sgd".

Class	Precision	Recall	f1-Score
class_0	0.00	0.00	0.00
class_1	0.50	0.43	0.46
class_2	0.33	1.00	0.50

Figure 5: Classification report for an MLP with the Solver "Adam" having an Accuracy of 38%

Class	Precision	Recall	f1-Score
class_0	0.00	0.00	0.00
class_1	0.00	0.00	0.00
class_2	0.24	1.00	0.38

Figure 6: Classification report for an MLP with the Solver "sgd" having an Accuracy of 22%

Looking only at the Accuracy one can see that "Adam" outperforms "sgd" by a large margin for this dataset and the given parameters. "sgd" classified only some of class_2 correctly while not correctly classifying any of class_0 or class_1 while "Adam" missing the true positives for class_0 only. None of the MLP in this experiments did converge.

1.2.2 Variying the Test size

The following Experiments measured the Accuracies for *solver="Adam"* for an increasing test data size. A MLPClassifier with two hidden layers each having 10 hidden nodes is used.

Test Size	Accuracy
0.1	0.38
0.2	0.38
0.3	0.25
0.4	0.36
0.5	0.38
0.6	0.43
0.7	0.49
0.8	0.41
0.9	0.26
0.99	0.39

Figure 7: Accuracies for a MLP with increasing test sizes

Increasing the test size does seem to have an effect for this particular classifier, but as the nature of MLP is that the weights are initialized at random, the accuracies for the same parameters do fluctuate very strong, sometimes $\pm 20\%$. For the experiments I used the first computed values only. As the accuracy do fluctuate very much one cannot make any valid assumptions what influence the test set has on the accuracy, based on this data. None of the MLP in this experiments did converge.

1.2.3 Crossvalidation

Integrating the given snippet for cross validation with the MLPClassifier and using 10-fold Cross Validation with an "accuracy" scoring algorithm leads to an accuracy of $34.2\% \pm 10.067$. This mean is lower than the accuracy of a manually splitted dataset, but the deviation is very broad and could include the previous manually found optimum for test size. None of the MLP in this experiments did converge.

Iteration	Accuracy
0	0.33333333
1	0.27777778
2	0.33333333
3	0.27777778
4	0.27777778
5	0.33333333
6	0.33333333
7	0.61111111
8	0.0.41176471
9	0.23529412

Figure 8: Accuracies for different Split Iterations of this 10-fold Crossvalidation

1.2.4 Grid Search

The following the Experiment searches for the best Hyperparameter for *max_iter* for a MLPClassifier using a GridSearch and 10-fold Crossvalidation. For the following Gridsearch a range of 10-910 (inclusive, 100er steps) is used for the max_iter parameter.
It found the best Accuracy of 57% for the Hyperparameter *max_iter=410*.
None of the MLP in this experiments did converge.

2 MNIST

2.1 MLP

First of all, we added a holdout validation to the sklearn implementation. A factor of approximately one-third of dataset has been chosen as the evaluation dataset. This is a common value for a train/test split. We used the build-in function from sklearn to perform this task. The accuracy is measured as the categorical cross-entropy and evaluated using the test data set.
We trained the MLP multiple times with varying parameters. Some of them are shown in the following part.

2.1.1 Learning rate

As expected we observed a higher accuracy if we decreased the learning rate. This also increased training time significantly.
Here we compare the different learning rates.
The following params were used:

```
"hidden_layer_sizes": (20, 20),
"activation": 'relu',
"solver": 'adam',
"batch_size": "auto",
"learning_rate": 'constant',
"learning_rate_init": 0.001,
"max_iter": 200,
"shuffle": True,
"random_state": None,
"tol": 0.0001,
"verbose": 1,
"early_stopping": True,
"validation_fraction": 0.2,
"n_iter_no_change": 10,
```

This setting resulted in 30 Epochs until the early stopping kicked in. The max accuracy was 0.949221.
The same settings with a learning rate of 0.0001 resulted in a max accuracy 0.950130 but needed 122 Epochs before being stopped by the early stopping. Setting the learning rate to 0.01 resulted in 17 epochs and an accuracy of 0.949394.
These results where expected.
Because we used the adam optimizer no learning rate decay had been applied.

2.1.2 Hidden Layers

Different hidden layer configs had been also evaluated.
We picked the following two:

- Two 20 neuron hidden layers as used in the config shown in 2.1.1. As already shown the maximum accuracy with our default learning rate of 0.001 was 0.949221.
- A 200 neuron layer followed by a 100 neuron layer. This significantly larger network took longer to train but resulted in an increased evaluation score of 0.977273 after 50 Epochs with a learning rate of 0.001. The other params are the same as described in 2.1.1.

Further numbers of layers and neurons had been evaluated but the results were not significant enough to discuss in detail.
A batch size of "auto" (sklearn sets it to the number of samples used for training) resulted in the best scores. We tried some over values like 512, 224, 64 with worse results.
The best overall score was 0.977273 with the larger mlp architecture.
The code can be found at GitHub.

2.2 CNN

We developed a simple convolutional neural network for the MNIST dataset using the keras framework build upon tensorflow.

The architecture:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_8 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               803072
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 861,386
Trainable params: 861,386
Non-trainable params: 0
```

The three convolutional layers are used for feature extraction. The two max-pooling layers reduce some spacial information. Dropout should help against the overfitting of the network. The last "logical" step consists of two dense layers. The last one uses softmax as an activation function to achieve a cleaner classification. All other layers use a normal relu activation.
The network achieves an accuracy of 0.9929 measured as categorical cross-entropy.
Code and params can be found at GitHub.