Programming Exercise 2

Decision Trees

Partition the dataset into a training and a testing set. Run a decision tree learning algorithm using the training set. Test the decision tree on the testing dataset and report the total classification error (i.e. 0/1 error). Repeat the experiment with a different partition. Plot the resulting trees. Are they very similar, or very different? Explain why.

Advice: it can be convenient to set a maximum depth for the tree.

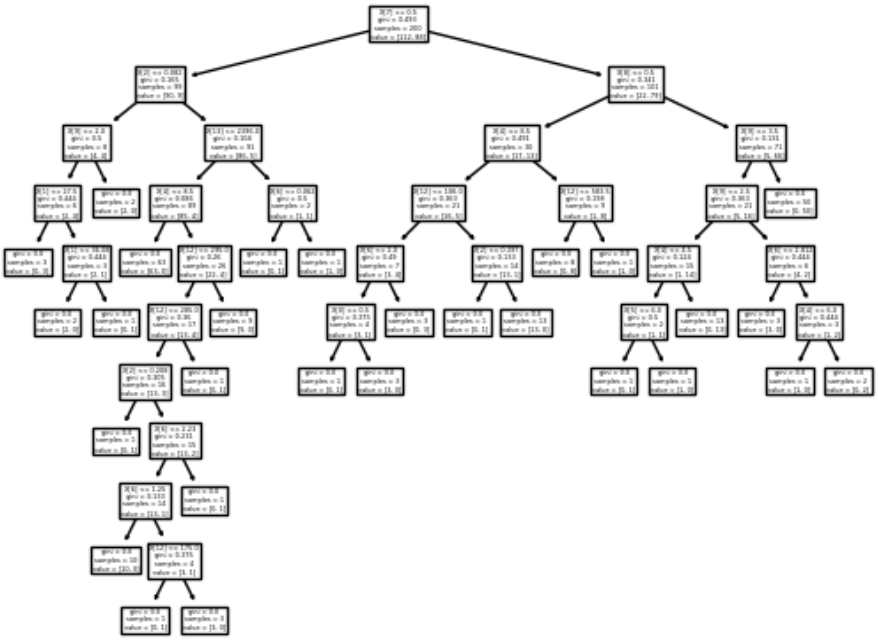


Figure 1: Resulting tree for *australian.txt* data (690 samples), Partition with a training set of 200 samples. Maximum depth equal to 20. Testing the decision tree on the testing dataset (490 samples), the total classification error obtained is equal to 86.

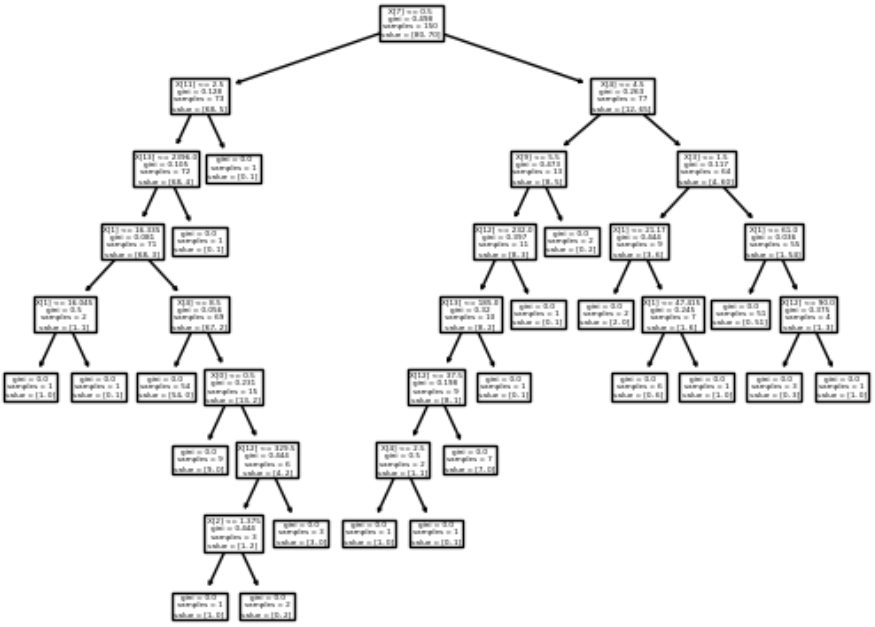


Figure 2: Resulting tree for *australian.txt* data (690 samples), Partition with a training set of 150 samples. Maximum depth equal to 20. Testing the decision tree on the testing dataset (540 samples), the total classification error obtained is equal to 113.

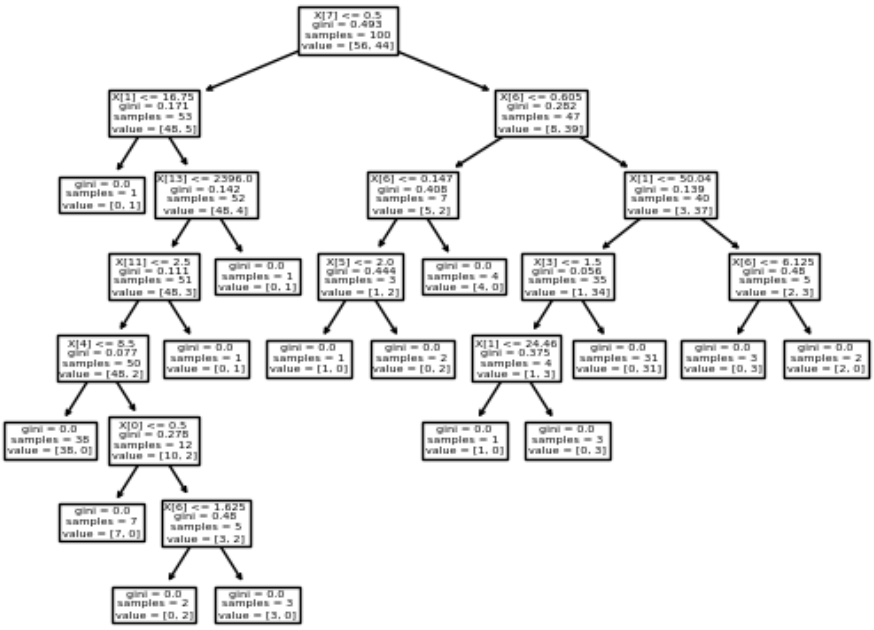


Figure 3: Resulting tree for *australian.txt* data (690 samples), Partition with a training set of 100 samples. Maximum depth equal to 20. Testing the decision tree on the testing dataset (590 samples), the total classification error obtained is equal to 171.

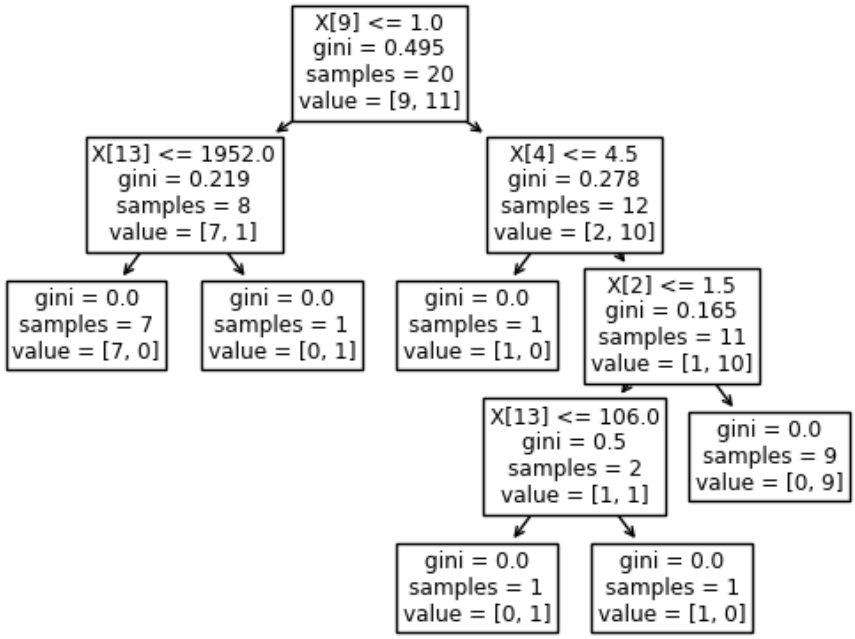


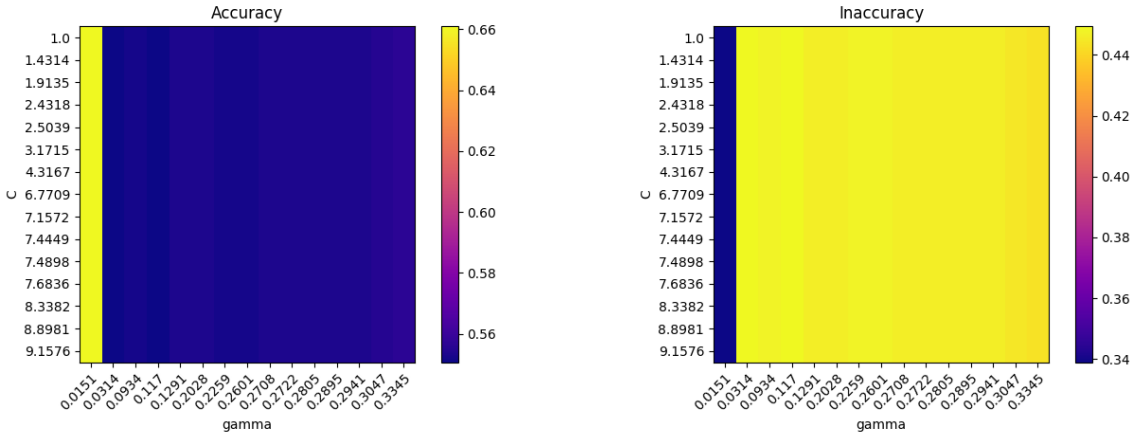
Figure 4: Resulting tree for *australian.txt* data (690 samples), Partition with a training set of 20 samples. Maximum depth equal to 20. Testing the decision tree on the testing dataset (670 samples), the total classification error obtained is equal to 191.

As we can see, these trees are very different from each others. The reason is their different partitions. When the partition has a training set with a low size (i.e. 20 samples, see Figure 4), then the amount of information used to construct the tree is scarce, and it results in a less accurate (and consequently less deep) tree. When the partition has a training set with a high size (i.e. 200 samples, see Figure 1), then the amount of information used to construct the tree is higher, and it results in a more accurate (and consequently deeper) tree.

Support Vector Machines

Run SVM to train a classifier, using radial basis as kernel function. Apply cross-validation to evaluate different combinations of values of the model hyper-parameters (box constraint C and kernel parameter γ). How sensitive is the cross-validation error to changes in C and γ? Choose the combination of C and γ that minimizes the cross-validation error, train the SVM on the entire dataset and report the total classification error.

Advice: use a logarithmic range for γ.

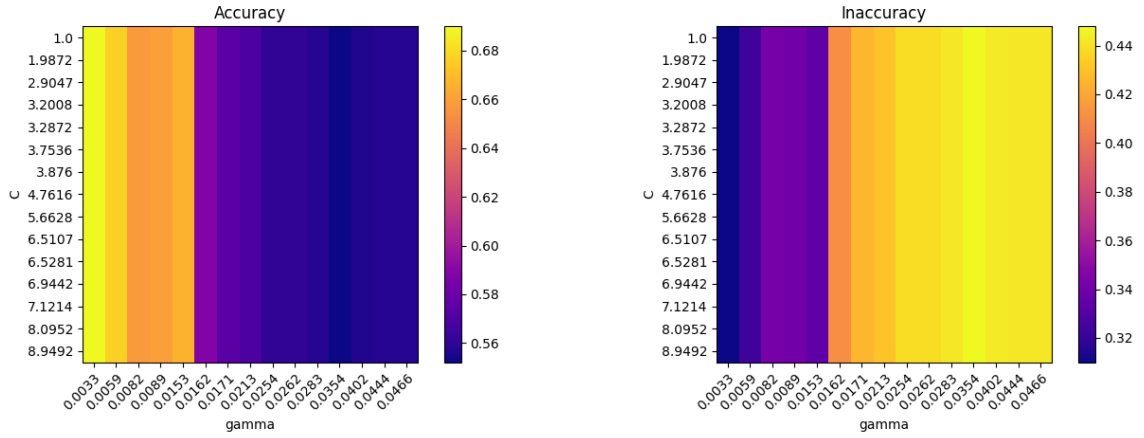


(a) The mean accuracy on the given test data and labels

(b) The mean inaccuracy on the given test data and labels

Figure 5: Heatmap showing how sensitive is the cross-validation accuracy to changes in C (Y axis) and γ (X axis). Apparently we can interpret that this accuracy is not sensitive to changes in C, while it is sensitive to changes in γ.

The minimum of the cross-validation inaccuracy seems to be achieved when γ es lower. We plot again this two graphics for lower values of γ in order to try to be more accurate.

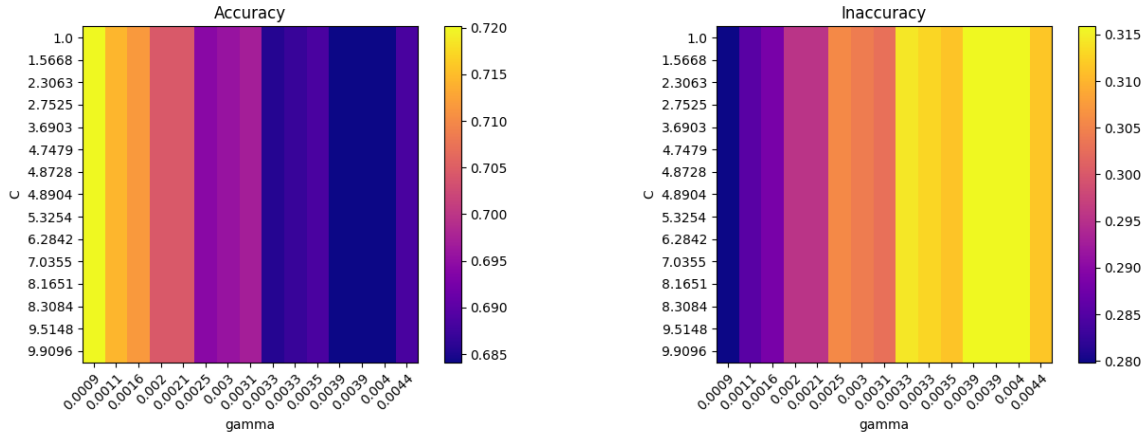


(a) The mean accuracy on the given test data and labels

(b) The mean inaccuracy on the given test data and labels

Figure 6: Heatmap showing how sensitive is the cross-validation accuracy to changes in C (Y axis) and lower values of γ (X axis). Apparently we can interpret again that this accuracy is not sensitive to changes in C, while it is sensitive to changes in γ.

Again, the minimum of the cross-validation inaccuracy seems to be achieved when γ even lower. We plot again this two graphics for even lower values of γ.



(a) The mean accuracy on the given test data and labels

(b) The mean inaccuracy on the given test data and labels

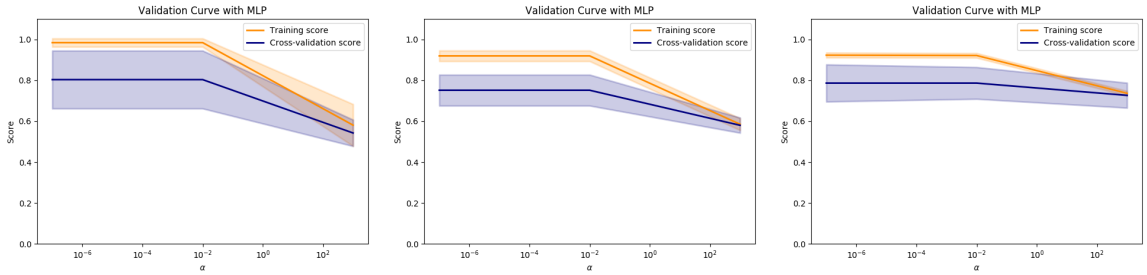
Figure 7: Heatmap showing how sensitive is the cross-validation accuracy to changes in C (Y axis) and even lower values of γ (X axis). Apparently we can interpret again that this accuracy is not sensitive to changes in C, while it is sensitive to changes in γ.

We can finally interpret that the minimum of the cross-validation inaccuracy seems to be achieved when the minimum value of γ is achieved, independently from the value of C.

Neural Networks

Train a Multi-Layer perceptron using the cross-entropy loss with $\ell - 2$ regularization (weight decay penalty). In other words, the activation function equals the logistic function. Plot curves of the training and validation error as a function of the penalty strength α. How do the curves behave? Explain why.

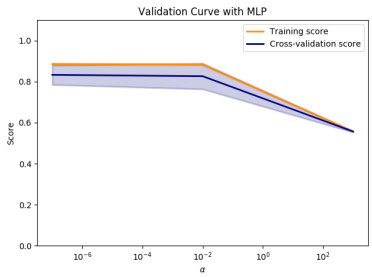
Advice: use a logarithmic range for hyper-parameter α. Experiment with different sizes of the training/validation sets and different model parameters (network layers).



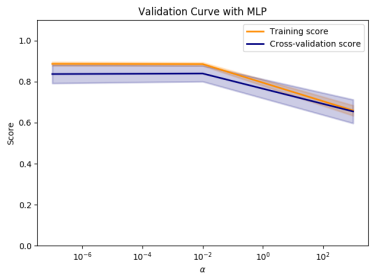
(a) Validation curve for a training set of $N = 50$

(b) Validation curve for a training set of $N = 100$

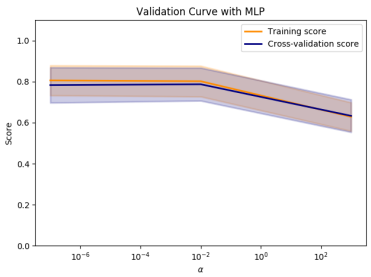
(c) Validation curve for a training set of $N = 150$



(d) Validation curve for a training set of $N = 300$



(e) Validation curve for a training set of $N = 400$



(f) Validation curve for a training set of $N = 500$

Figure 8: Training scores and validation scores of an MLP for different values of the parameter α.

We know that if the training score and the validation score are both low, the estimator will be underfitting. If the training score is high and the validation score is low, the estimator is overfitting and otherwise it is working very well. We can observe how, increasing the size of the training set, overfitting situation is overcomed.