MACHINE LEARNING
Programming Exercise 2

Emma Fraxenet, Josep Ferrer

November 17th 2020

1 DECISION TREES

For this first task, we picked the Gisette binary classification data set (# of classes: 2, # of training data: 6000, # of testing data: 1000 and # of features: 5000). All the programming has been done with *python*, using the *Jupyter notebook* environment. The Decision Tree algorithm has been imported from the *scikit-learn* library. We use the *graphviz* library to visualize the decision trees.

Since we used a training and a testing dataset separately, the first partition was not needed. However, a partition of the training data set was done with *train_test_split*, in order to study the effect of different training data on the final shape of the tree. Each resulting training set has half of the data points of the original data set.

We first explored what happened if we didn't set a maximum depth for the tree. As we can see in Fig.1 the trees were build with very different shapes. However, if we study the total error classification (error on the test data set: presented in table below) we find that it is very similar.

Then, setting a maximum depth of 4 nodes, we see the trees have a very similar shape. The total error classification is maintained at a very similar value as well. Note that we use the same partition (we use the same data points as the first case).

From this results we derive that for different training sets we can obtain a very different decision logic, but the accuracy of those will always be around the same value.

No maximum depth	Tree 1 (50% training set)	Tree 2 (50% training set)
Total classification error	0.094	0.083
Maximum depth = 4	Tree 3 (50% training set)	Tree 4 (50% training set)
Total classification error	0.091	0.085

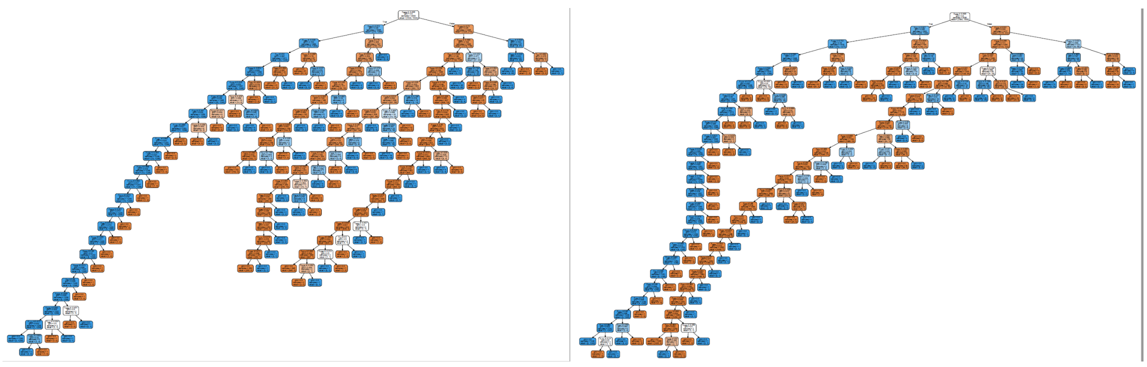

Figure 1: Tree 1 and 2. No maximum depth.

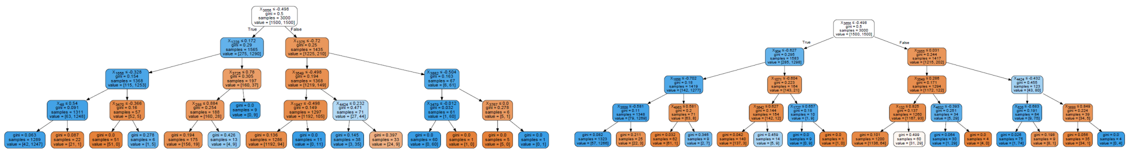

Figure 2: Tree 3 and 3. Maximum depth=4.

2 SUPPORT VECTOR MACHINES

For this second task, we also picked the Gisette binary classification data set (# of classes: 2, # of training data: 6000, # of testing data: 1000 and # of features: 5000). As well as the previous part, all the programming has been done with *python*, using the *Jupyter notebook* environment. The SVM algorithm has been imported from the *scikit-learn* library.

To train this classifier we are using the kernel function *radial basis*. In order to choose the appropriate values we first do a cross-validation to evaluate the possible combinations of the parameters. In this case the parameters are C (acts as a regularization parameter, influences on the size of the soft margin and penalizes miss classified data points) and γ (inverse of the distance influenced by each data point that is used as a support vector). To do this cross validation we use a randomly chosen 20% of the training set (using *train_test_split*) and obtain the accuracy for a grid structure of the parameter's combinations (Fig.3).

To obtain Fig.3 we chose logarithmic ranges of the parameters, by first doing a study of broader ranges we decided to only show the ones in the figure, since those are the ranges where we observe a change in accuracy. Given that the change in accuracy was sudden in a certain region, we normalized the color map around those values to get a more detailed color change.

2.1 Analysis of the cross-validation grid search

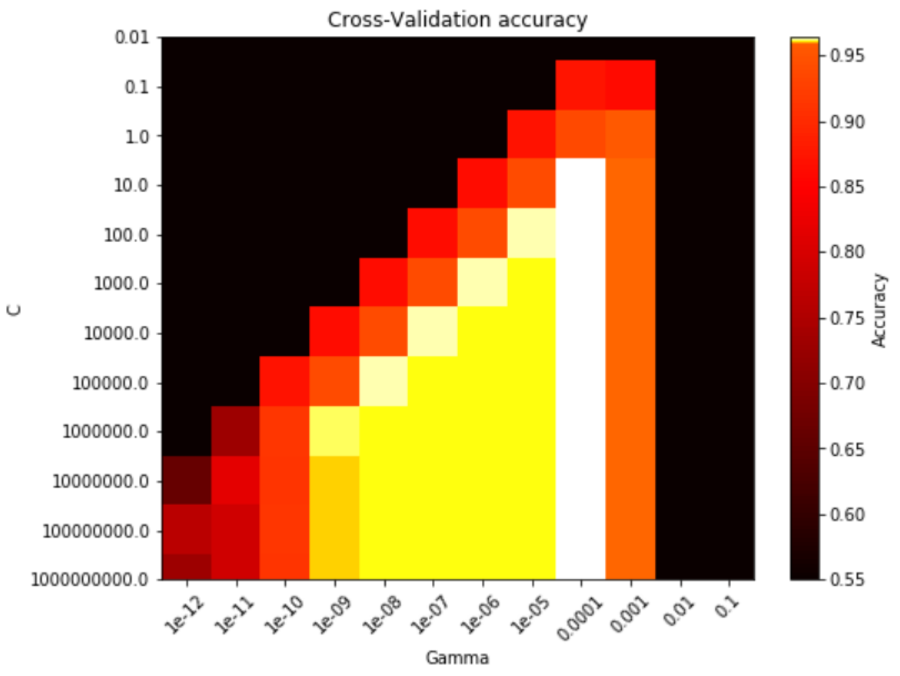

Figure 3: Heatmap of cross validation error for a grid of the parameter's values.

First lets focus on the combinations that don't provide a good accuracy: for lower C (more regularization) we obtain a simpler decision function, which is a larger margin, but we sacrifice accuracy. At the same time, if γ is too low (bigger radius) the support vectors influence the whole training set and all the complexity is lost: the resulting model will be too simple and we will loose accuracy again. This can be slightly balanced by bigger C, which leads to less regularization and more complexity, but only helps up to a certain point. For larger γ the influence of the support vectors doesn't reach far and this leads to over-fitting.

In the figure we observe a diagonally shaped growth of accuracy, together with a clear winner in the column of $\gamma = 0.0001$. This is due to the balance of larger radius of influence (lower γ) and complexity of the decision function (larger C, less regularization).

However, when C gets larger (less regularization, smaller margin) and we have medium values of γ (range $1^{-9}, 1^{-4}$), accuracy is still maintained in a relatively good balance. The reason why that happens is because we balance the over-fitting due to lack of regularization (C is too large) with the provided structural regularization of a larger radius of influence(γ is smaller).

Our best parameters will then be those with best accuracy. Given that there are several options (different values of C), we choose the one with a lower C in order to use a simpler decision function, which is beneficial for memory storage and speed of predictions.

We provide the score results on this classifier trained with the best parameter combination (found in the grid search) and the whole training set in Table.2.1.

Best parameters	Accuracy in grid seach	Accuracy for testing set (after training in the whole training set)
C =10.0	0.964 ± 0.012	0.978
$\gamma = 0.0001$		Total classification error $= 0.022$

Table 1: Score results on SVM model

3 NEURAL NETWORKS

Train a Multi-Layer perceptron using the crossentropy loss with '-2 regularization (weight decay penalty). In other words, the activation function equals the logistic function.

1. **Plot curves of the training and validation error as a function of the penalty strength .**
2. **How do the curves behave? Explain why.**

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

For this third task, we picked the Mushrooms classification dataset (# of classes: 3, # of data: 8124 and # of features: 112). All the programming has been done with python, using the Jupyter notebook environment. The Multi-Layer perceptron algorithm has been imported from the scikit-learn library. A second dataset has been used as well in order to compare different results.

First, we start performing this third task trying different training / test data sets sizes and MLP models with different number of hidden layers to see how it affected the obtained results. As we expected, the general behavior of the error function versus the alpha value is always the same because the overfitting or generalization is a universal phenomenon, that does not depend on the size or the number of hidden layers used. However, we do observe that the larger the training data set is, the clearer this effect becomes, and thus the obtained functions are smoother, since you get rid of finite-size effects. As it can be observed in the following figure:

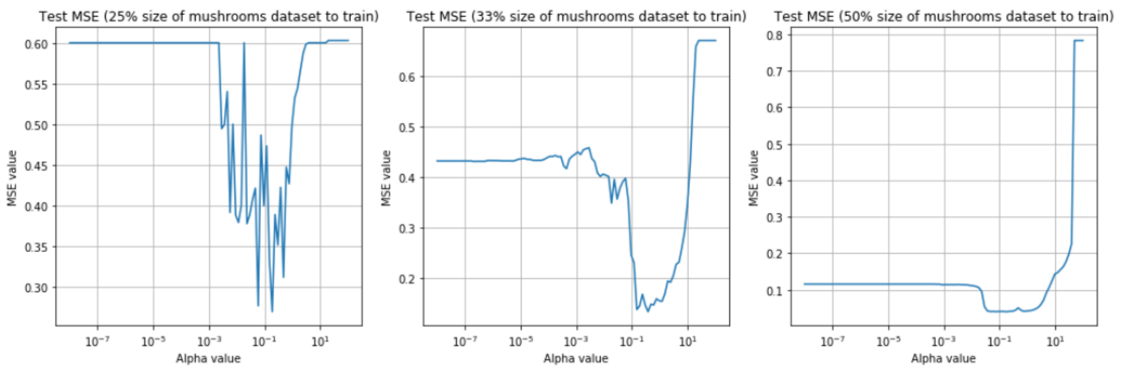

Figure 4: Comparison of the error function depending on the training data set size.

The same behavior can be observed when comparing the training error variance with the training data set size.

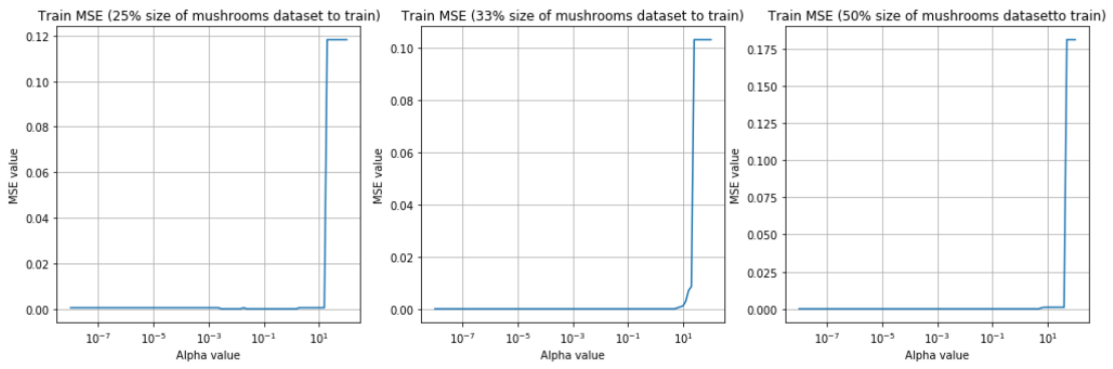

Figure 5: Comparison of the error function depending on the training data set size.

When talking about how many hidden layers are used to generate our model, there is a trade-off between dataset size and model complexity. Basically, a 3-layer model is more complex than 1 or 2 layers. However, if for a constant dataset size we use an increasing number of hidden layers, we will be using a too complex model for too simple data and, hence, error functions will be less smooth for larger numbers of parameters. As we can observe in the obtained functions behavior, even though the results tendency are the same, for a 3-hidden-layer system the function are more noisy. It can be observed in the test error function attached below:

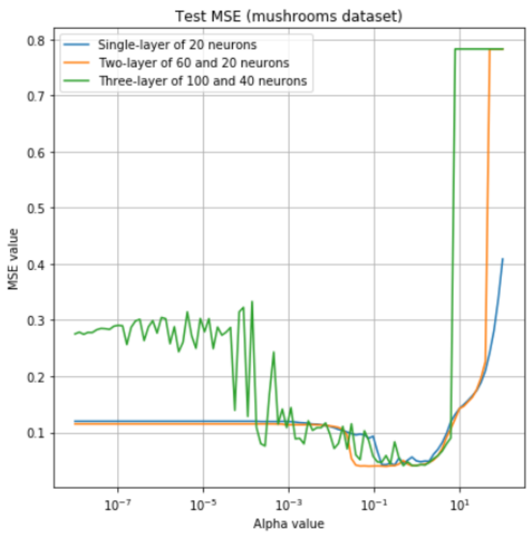

Figure 6: Comparison of the error function depending on the number of hidden layers.

One can easily observe that for our case, the two layer MLP model performs the best. The same effect can be observed when increasing the number of neurons for a fixed number of hidden layers, as it can be seen in the following figure for a two-layer model.

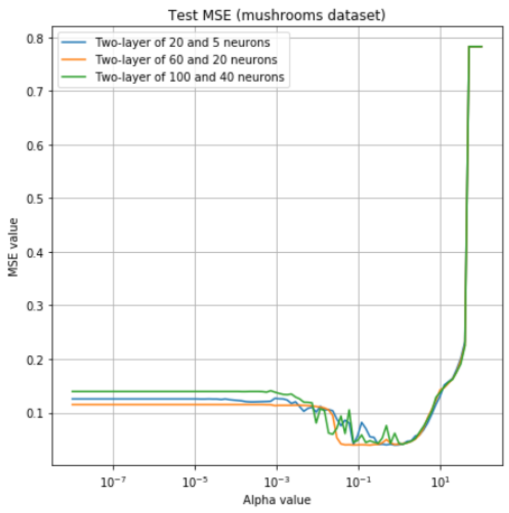

Figure 7: Comparison of the error function of a two hidden layer model depending on the number of neurons.

In the limit of a very large dataset and a very large model we should observe the same qualitative behavior with very smooth curves.

Once observed that the 2-hidden layer MLP model works the best, the corresponding training and test MSE functions for different alpha values are obtained. To compute the total error in our model, the MSE has been computed between the real y data and the predicted one, for both training and test data sets. The both obtained functions are attached in the following figure:

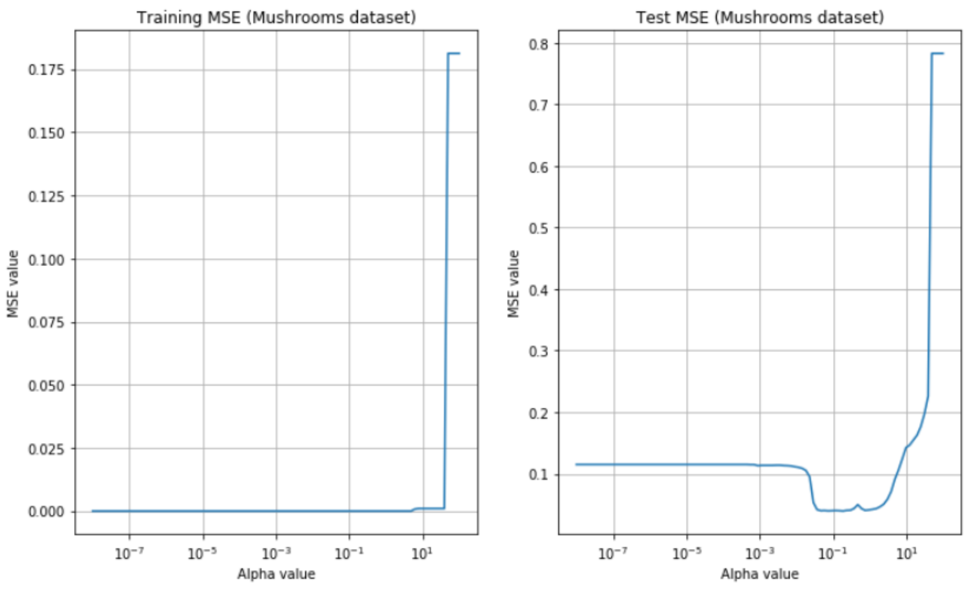

Figure 8: The obtained training and test MSE behaviour functions for a two hidden layer MLP model with 60 and 20 neurons.

When talking about the behaviour of the test error (the error obtained when generalizing our model to the whole dataset), when the alpha value is huge (>> 1) the MSE is really high, when having values between 10^{-2} and close to 1 the MSE keeps decreasing and for really small values the MSE converges to a lower constant value and remains there. This obtained behaviour can be be easily explained. First, as a huge value of alpha is being used, an overregularization is being performed. Therefore, the error of the model increases a lot as we are forcing the model to decay too fast. Then, when using intermediate values of alpha, the required regularization is applied and therefore, an improvement of the model is achieved by decreasing the MSE. In this range of alpha values we observe the best performance of our model. To end, when the alpha value is too small, no regularization is performed and thus the test error converges to the one without regularization.

When talking about the training error, a constant MSE function is always obtained for lower values of alpha, with a corresponding value of 0, that basically means that no error is obtained. This makes sense, as for our training dataset, the neural network models perfomance tend to be really good. However, this MSE abruptly increases when the alpha value gets really high (>> 1), indicating a really bad performance of the model. Again, this last behaviour can be explained because an overregularization is applied, and thus, we induce a bad performance of the model within the training data set.