MACHINE LEARNING ASSIGNMENT 2

ELİF HANGÜL

1. 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.*

Answer: Breast cancer dataset is used. The shape of the dataset is 683 x 10. For max depth 3 and 7 different partitions of the training and test sets are evaluated and tree is plotted. Test size %20, %30 and %40 are tried. Zero one loss is calculated for each combination.

0/1 error for different test sizes with max depth of 3 are

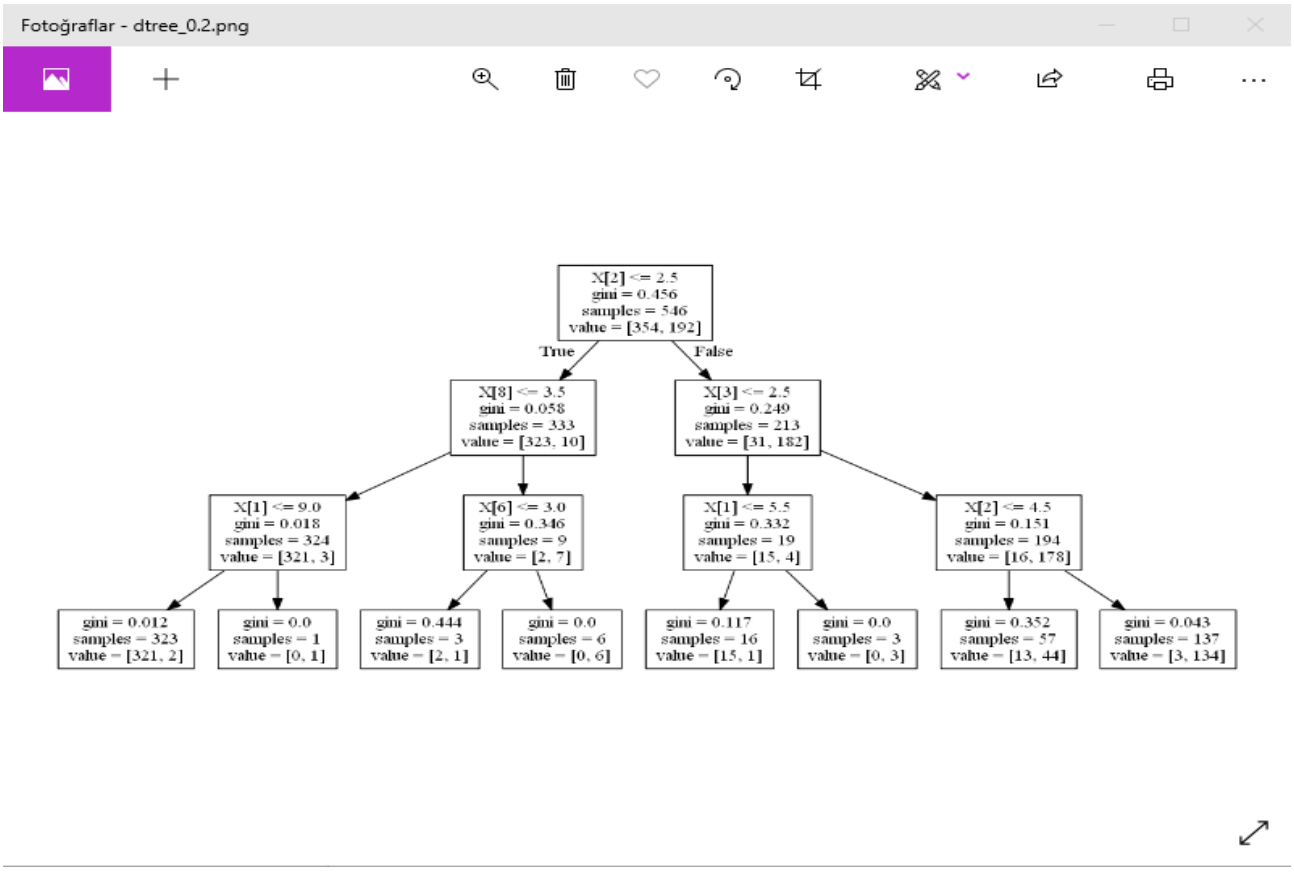
0/1 Loss for test size of: 0.2 ,max depth of 3 0.04379562043795615
0/1 Loss for test size of: 0.3 ,max depth of 3 0.0536585365853659
0/1 Loss for test size of: 0.4 ,max depth of 3 0.06569343065693434

0/1 error for different test sizes with max depth of 7 are

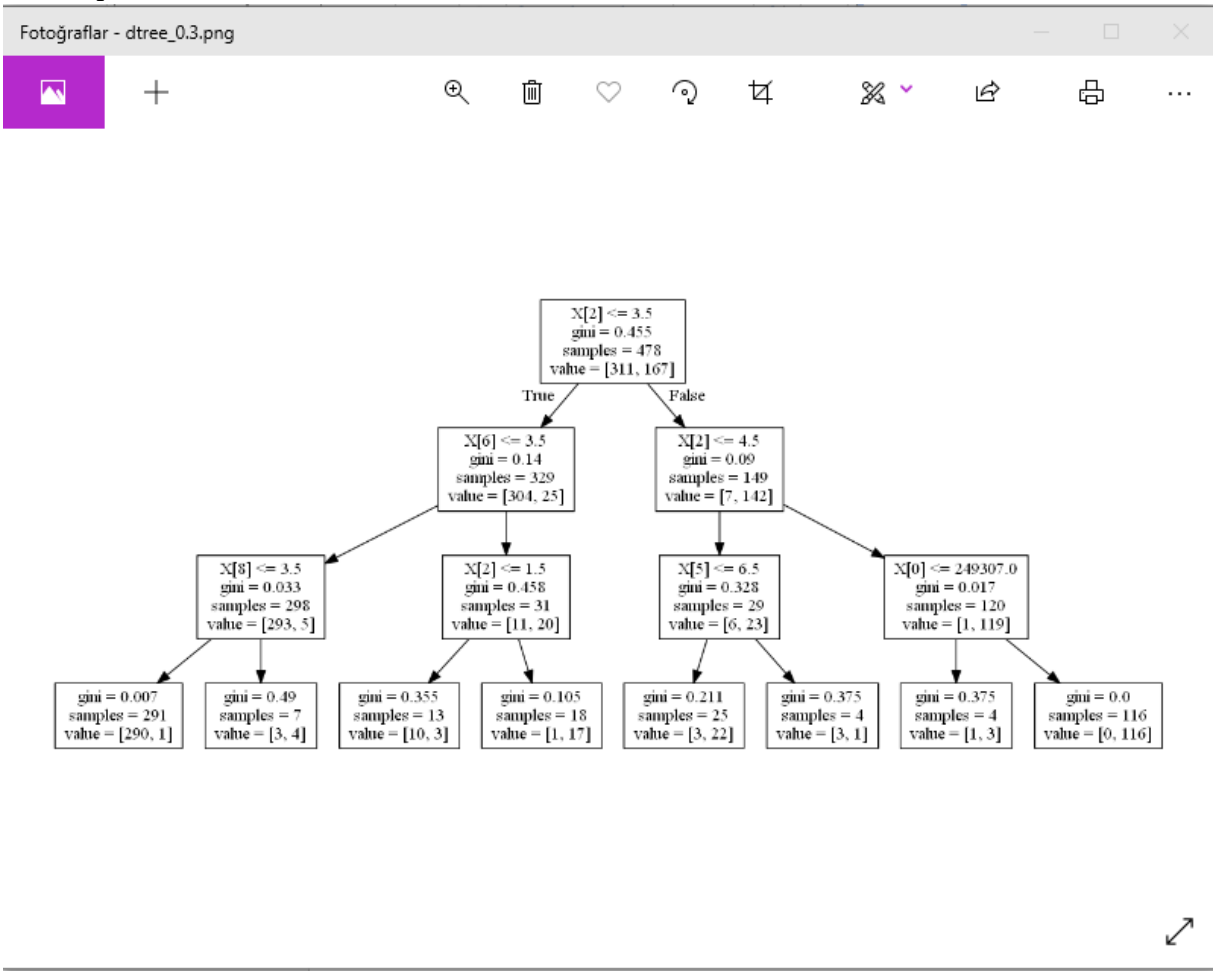
0/1 Loss for test size of: 0.2 ,max depth of 7 0.029197080291970767
0/1 Loss for test size of: 0.3 ,max depth of 7 0.0536585365853659
0/1 Loss for test size of: 0.4 ,max depth of 7 0.04744525547445255

The trees are

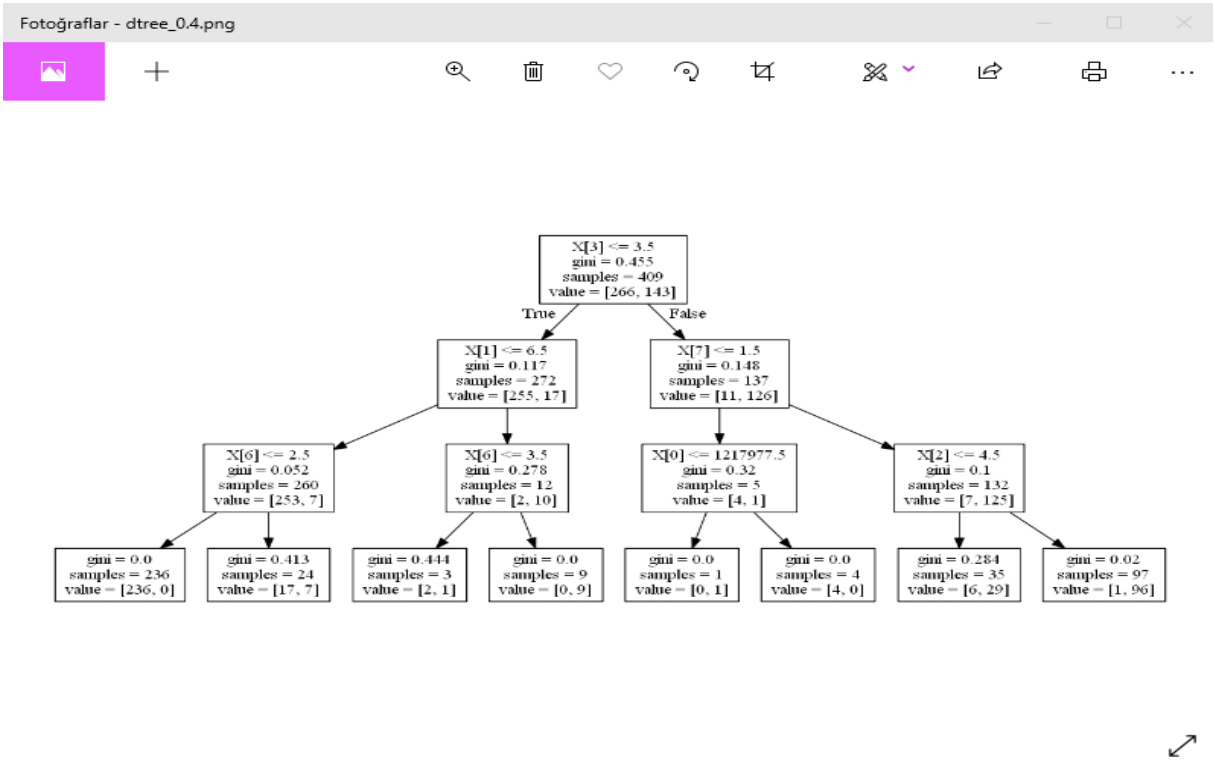
1) max depth = 3, test size = %20



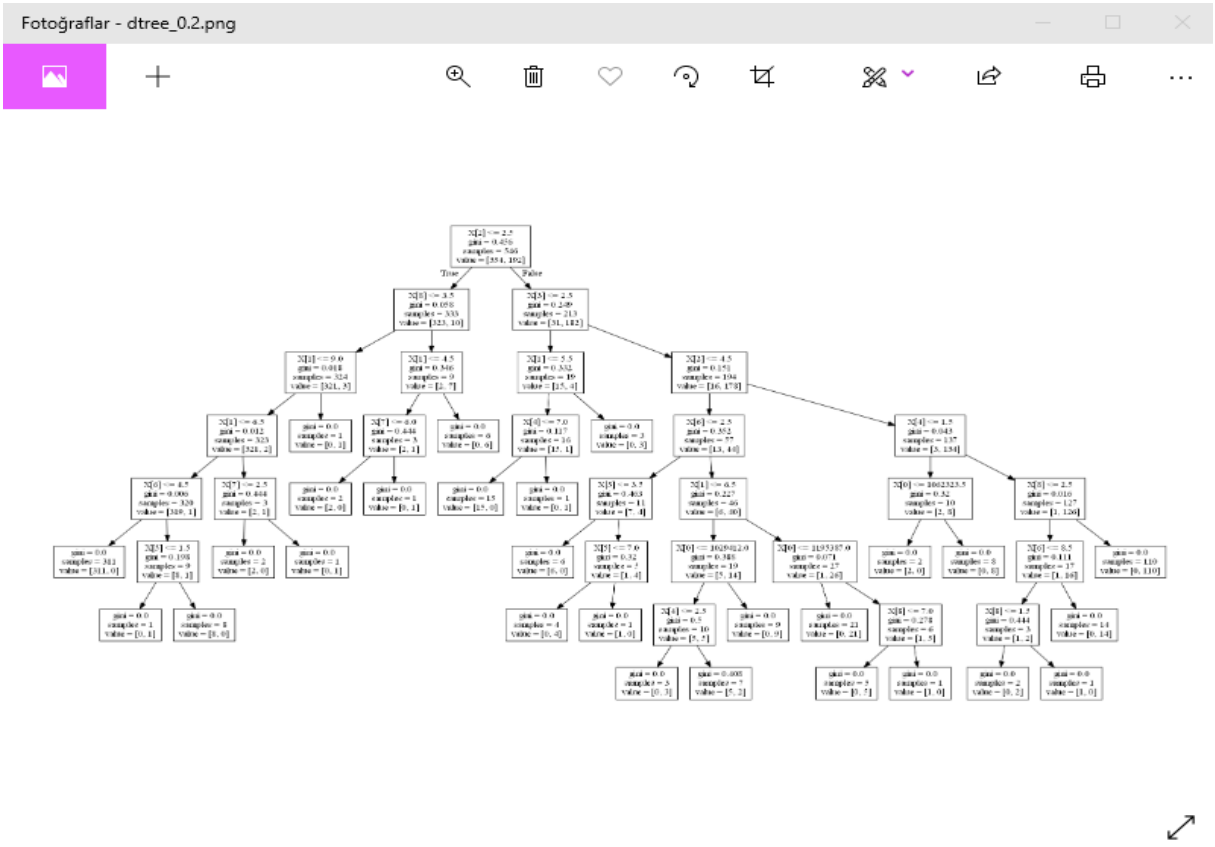
2) max depth = 3, test size = %30



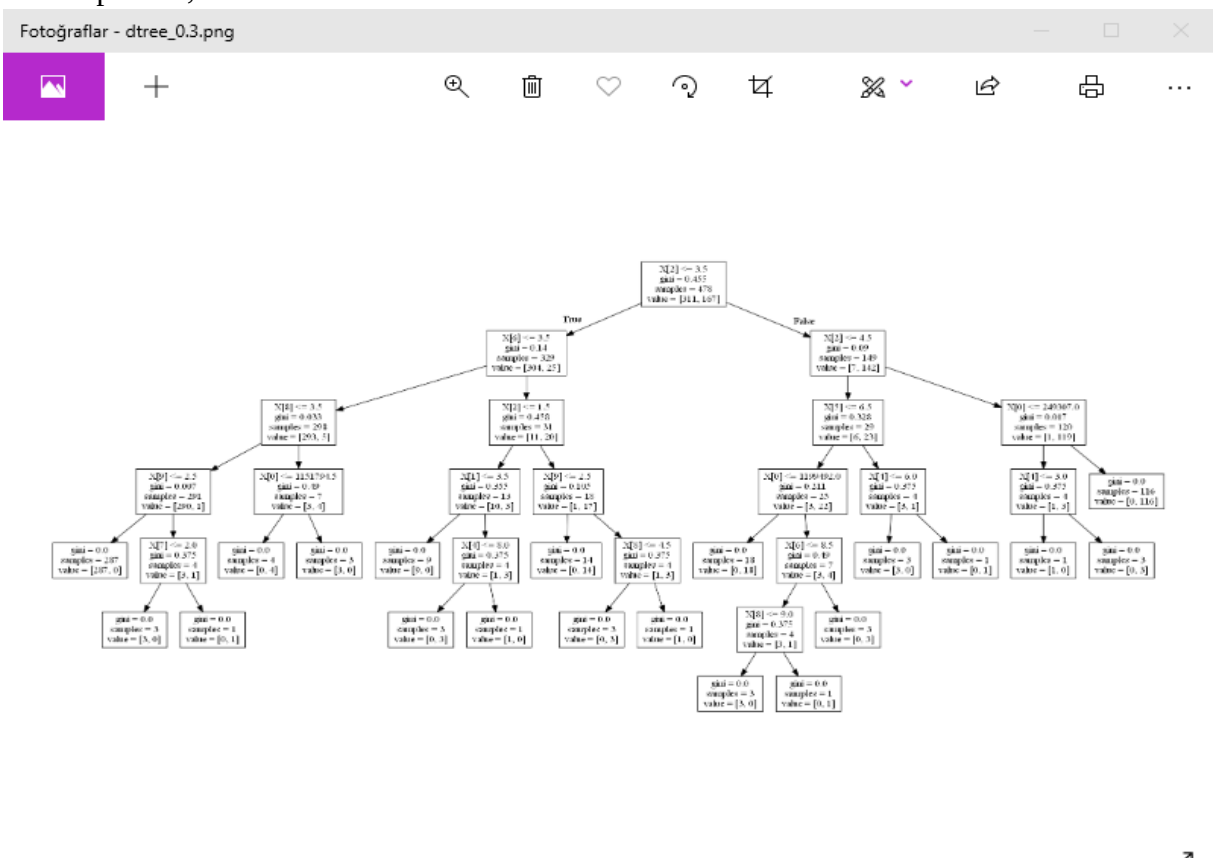
3) max depth = 3, test size = %40



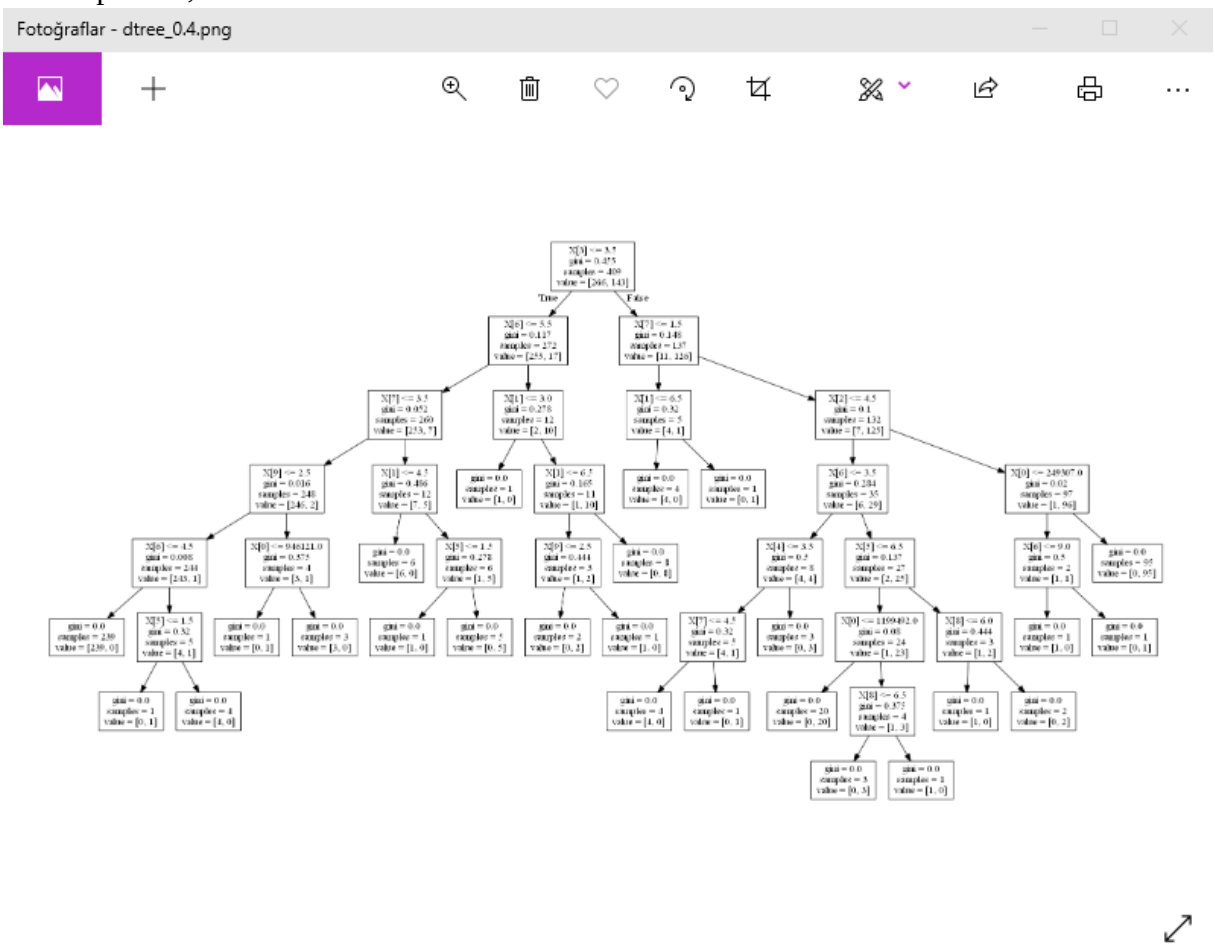
4) max depth = 7, test size = %20



5) max depth = 7, test size = %30



6) max depth = 7, test size = %40



If we look at the graphs, the numbers on the nodes and the branch models can be different for different partitions. The trees learn different things but they have similar qualities. When we look into the 0/1 error in general, we can see for smaller test size we will have less error, more accuracy. Likewise when the maximum depth of the tree increases there are less error and more accuracy. We can see this below.

0/1 Loss for %20 test size

0/1 Loss for test size of: 0.2 ,max depth of 3 0.04379562043795615

0/1 Loss for test size of: 0.2 ,max depth of 7 0.029197080291970767

0/1 Loss for %30 test size

0/1 Loss for test size of: 0.3 ,max depth of 3 0.0536585365853659

0/1 Loss for test size of: 0.3 ,max depth of 7 0.0536585365853659

0/1 Loss for %40 test size

0/1 Loss for test size of: 0.4 ,max depth of 3 0.06569343065693434

0/1 Loss for test size of: 0.4 ,max depth of 7 0.04744525547445255

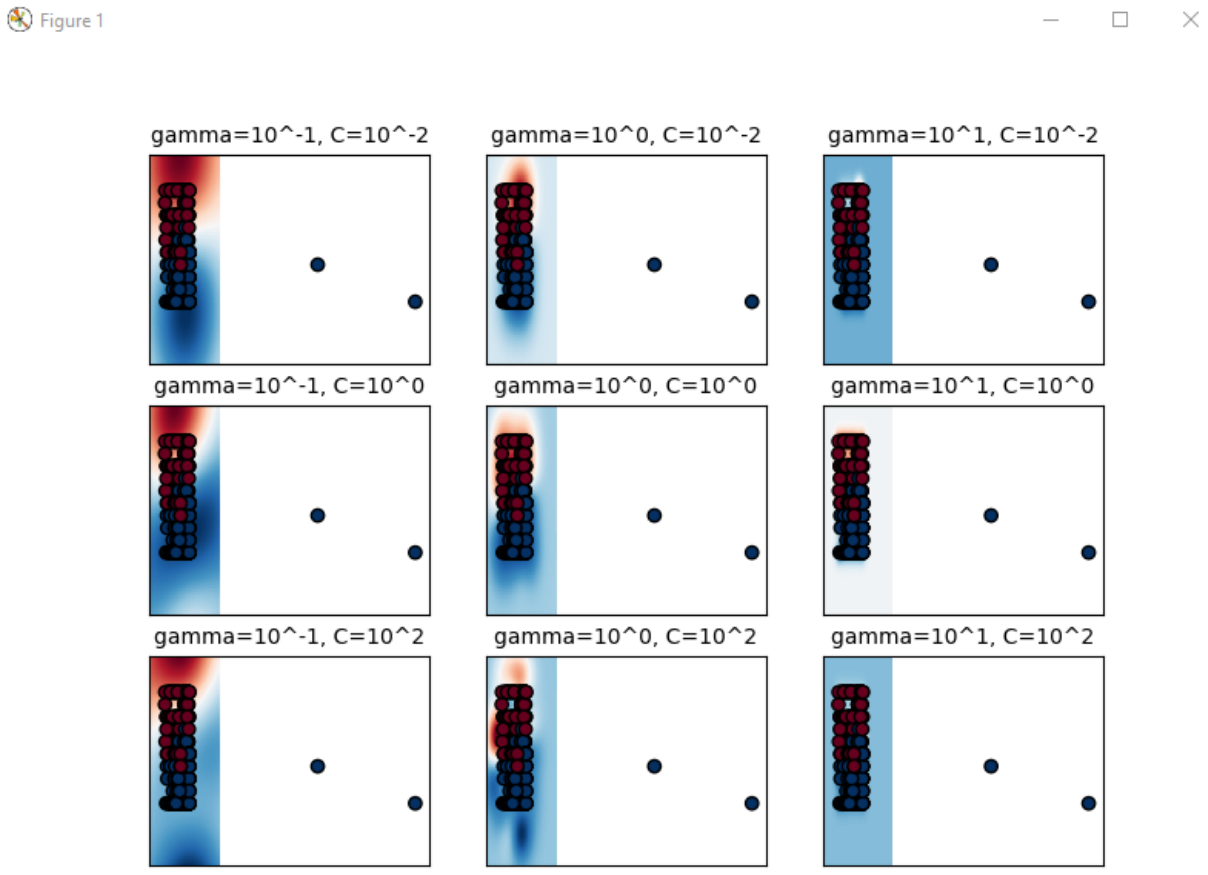
In summary more depth in the tree and less test dataset size give less error.

2. 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.

Answer: I used scaled breast-cancer dataset for SVM Classification. The dataset shape is 683x10. C and γ are chosen from a logarithmic range and different values of these parameters are used for evaluation. For cross validation 5 splits are performed, test size is chosen as %20. A grid is created for SVC and grid search is applied to get the best parameters and best results. A heatmap showing the correlation between C, γ and validation accuracy is plotted. Also, as an example the first two features of the dataset (age, menopause) are shown on a meshgrid for different gammas and C's to show the classification.

The below picture shows the meshgrid.



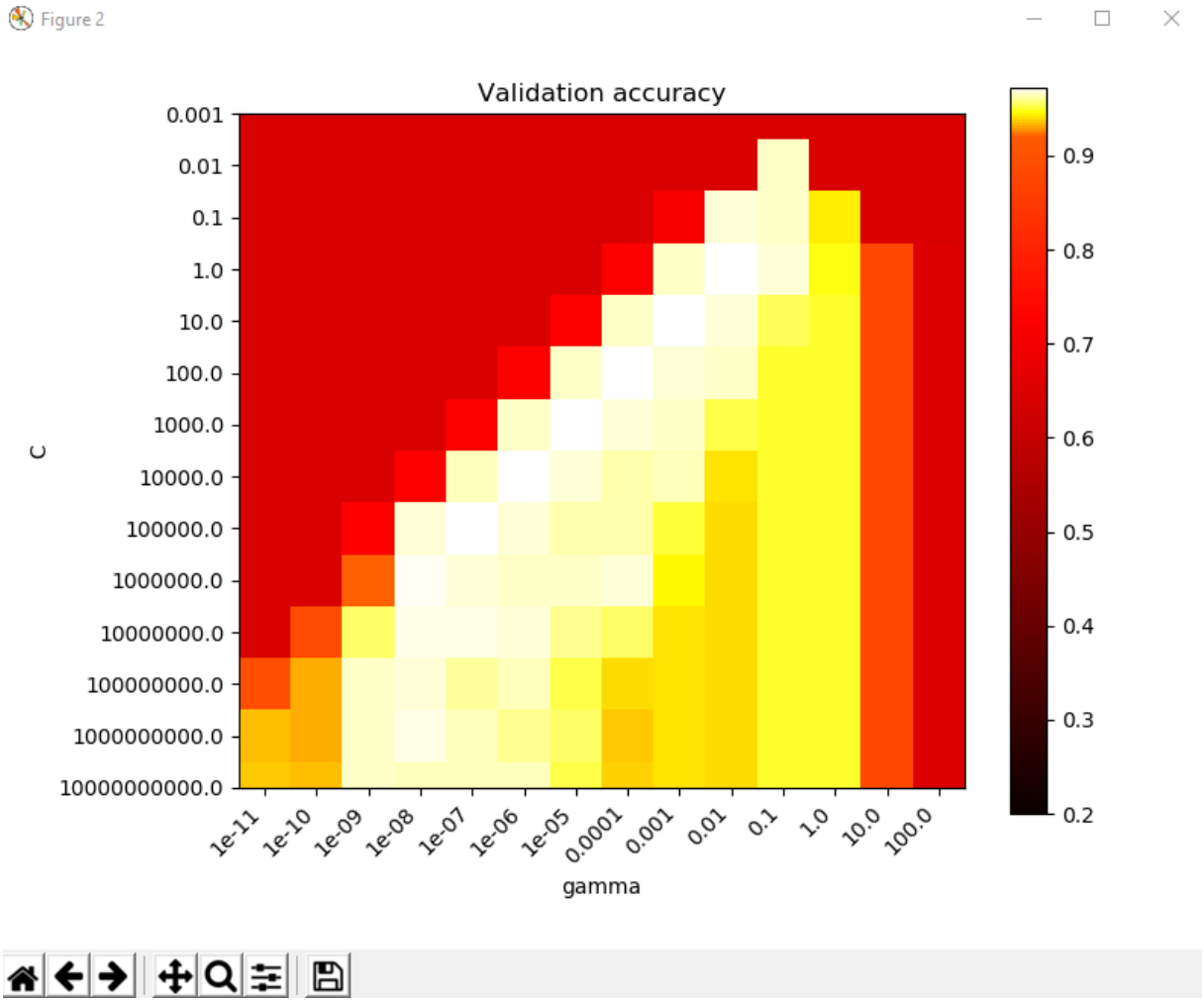
The validation accuracy score is calculated from the mean result of the cross validation splits, so the error can be stated as 1-accuracy, as this metric will give us the classification error.

After the run of SVM, the best parameters and score are calculated as in the below picture.

```
Console
<terminated> svm.py [D:\Python\Python36-32\python.exe]
The best parameters are {'C': 1.0, 'gamma': 0.01} with a score of 0.96
```

Total classification error is: 0.04

The heatmap below illustrates the effect of the parameters gamma and C of the Radial Basis Function (RBF) kernel SVM.



Now, let's interpret our results to see how sensitive of cross validation error to changes in C and γ. Lighter colors in the heatmap indicate the better results. First let's start with defining the meanings of C and gamma parameters.

The C parameter trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy.

For the gamma parameter on the other hand, if it is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting.

When gamma is very small, the model is too constrained and cannot capture the complexity or "shape" of the data.

As we can see from the heatmap diagonal C and gamma values gives some good models. It is possible to make smooth models (lower gamma values) more complex by increasing the importance of correctly classifying each point (large C values) which leads to the diagonal of good performing models.

For some intermediate gamma values larger C's gives same performances. In such cases a model that with the smaller C value could be chosen to favor models that use less memory and that are faster to predict.

There can be small differences in the in scores results from random splits of the cross validation. Increasing the number of splits can smooth these results but cost more time to compute[1]

3. Neural Networks: *Train a Multi-Layer perceptron using the cross entropy loss with l-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).*

Answer: Diabetes dataset is chosen. The shape is 768 x 8. Normalization on the data is done. Logarithmic range for alpha parameter is used. Cross entropy loss metric log_loss from sklearn library is used to evaluate the error.

Sklearn.metrics.log_loss is defined as the cross entropy loss. This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of the true labels given a probabilistic classifier's predictions[2]

Different hidden layer size, max iteration counts and different partitions of the training and test sets are chosen to experiment and all the curves for different parameter combinations are plotted as the training and validation error as a function of the penalty strength alpha.

Experimented values are:

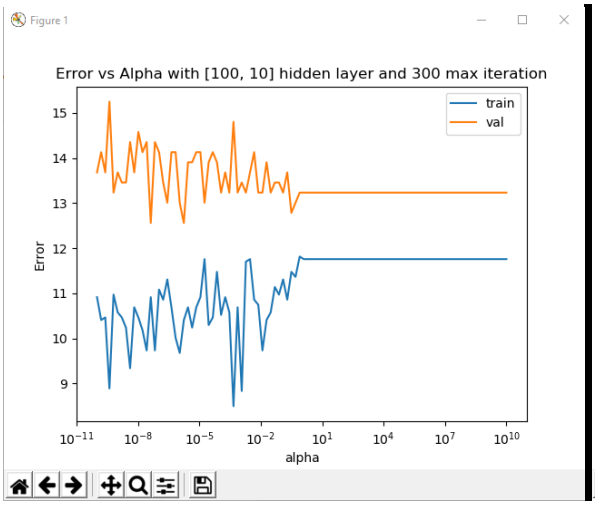
Number of hidden layer = 10, 50, 100

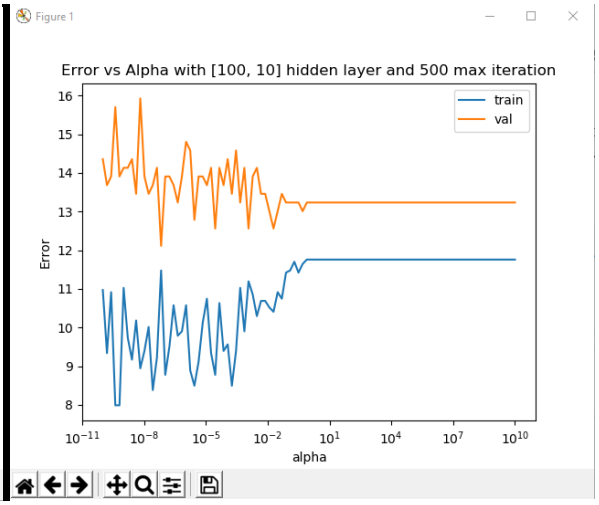
Maximum iteration count = 300, 500, 600

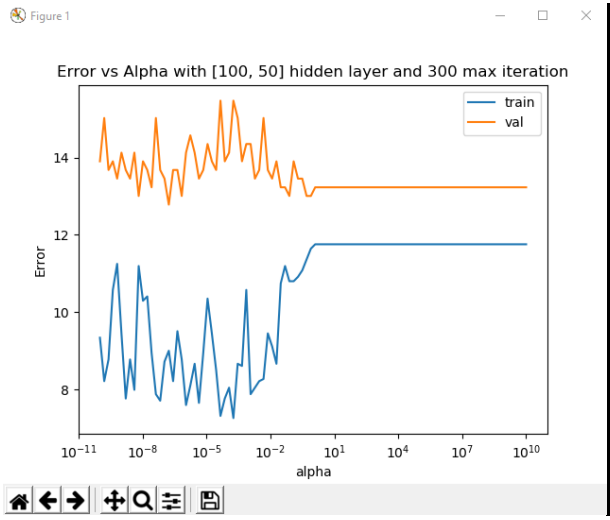
Test size = %20, %30, %40

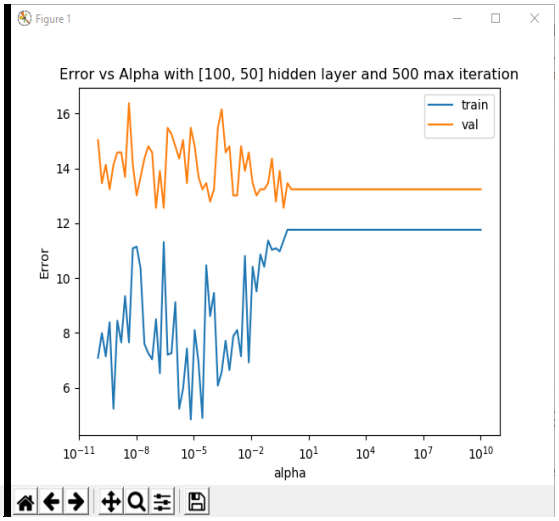
Below graphs are plotted.

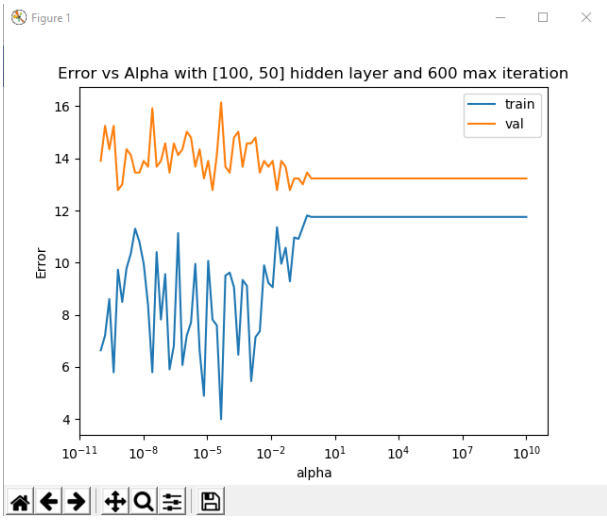
For test size %20

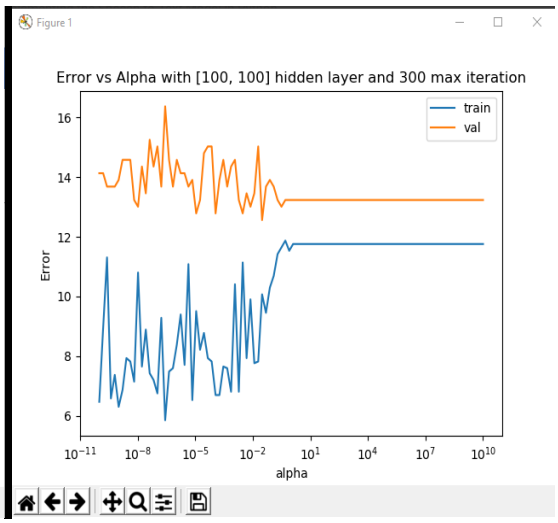


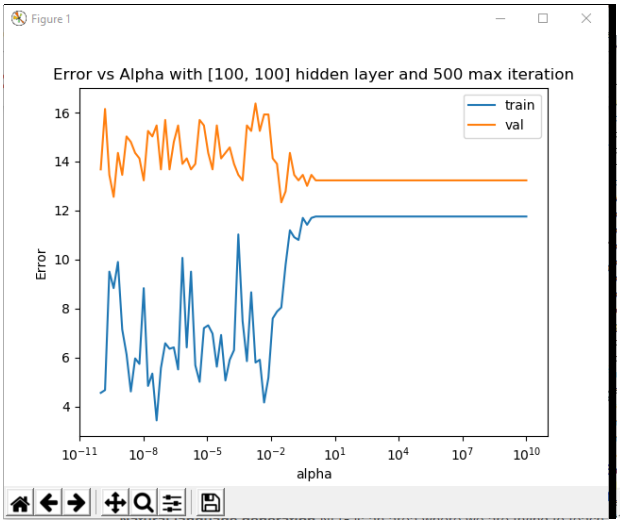


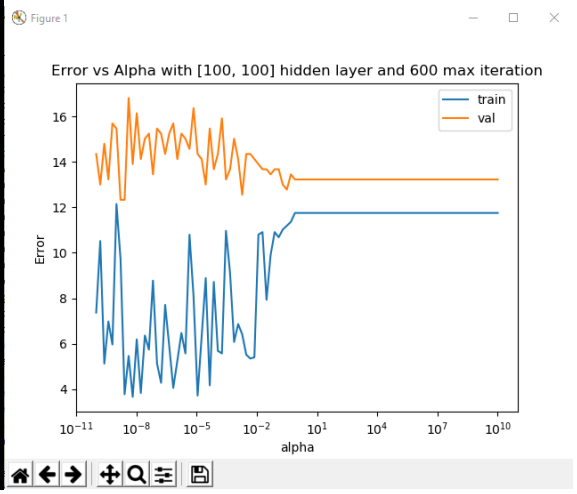




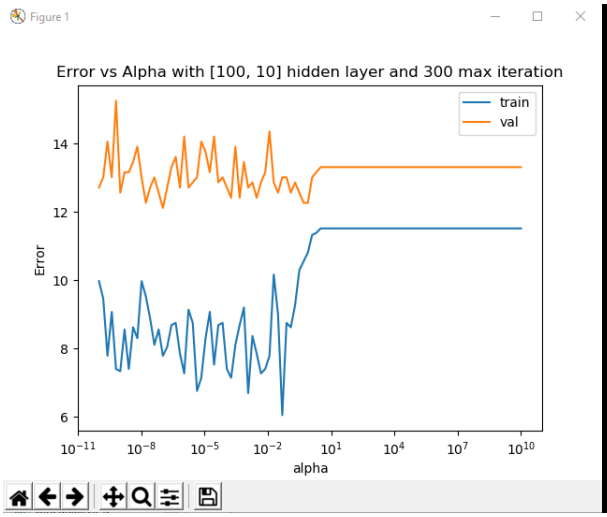


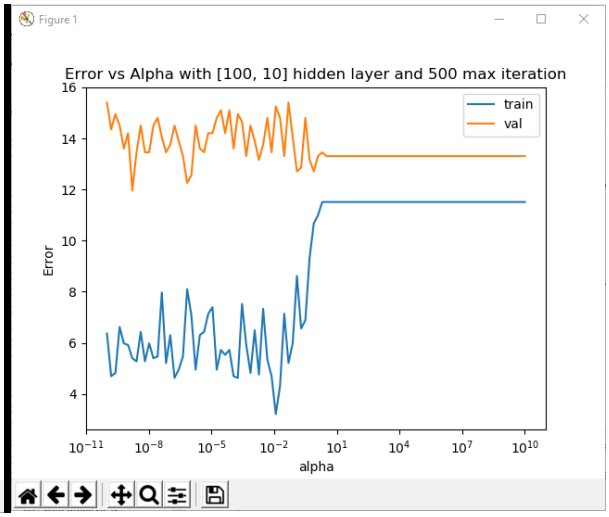


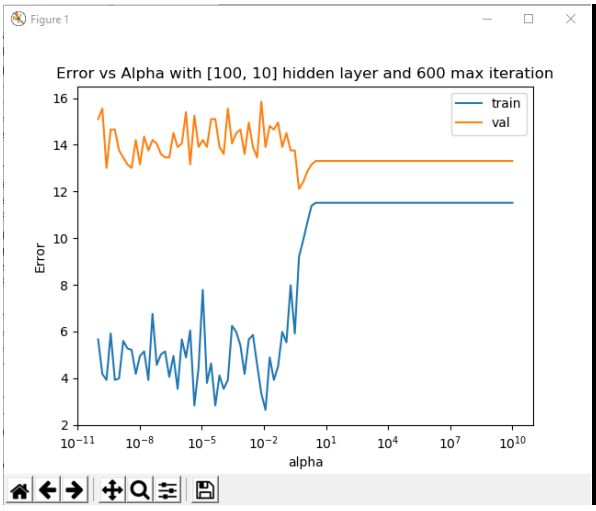


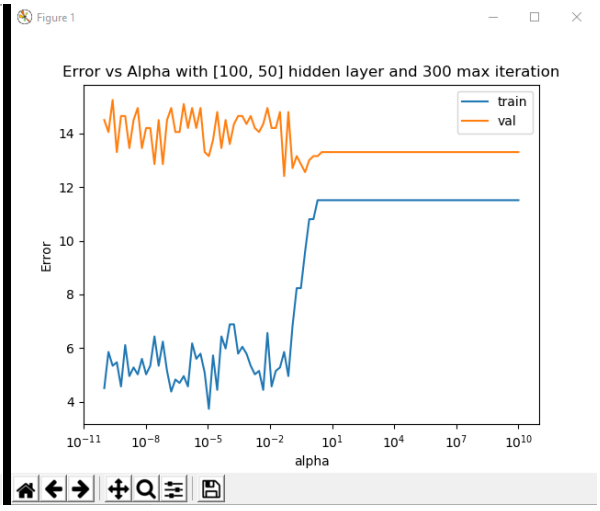


For test size %30









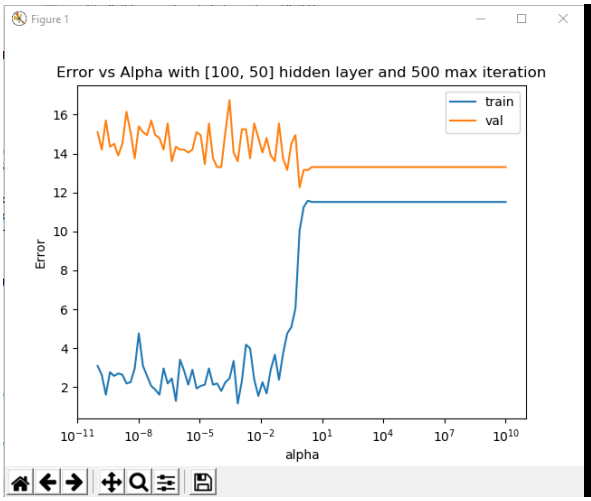


Figure 1
Error vs Alpha with [100, 50] hidden layer and 500 max iteration
train
val
Error
alpha

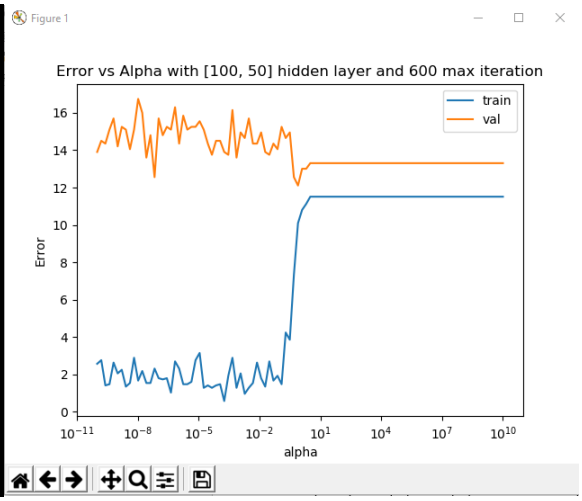


Figure 1
Error vs Alpha with [100, 50] hidden layer and 600 max iteration
train
val
Error
alpha

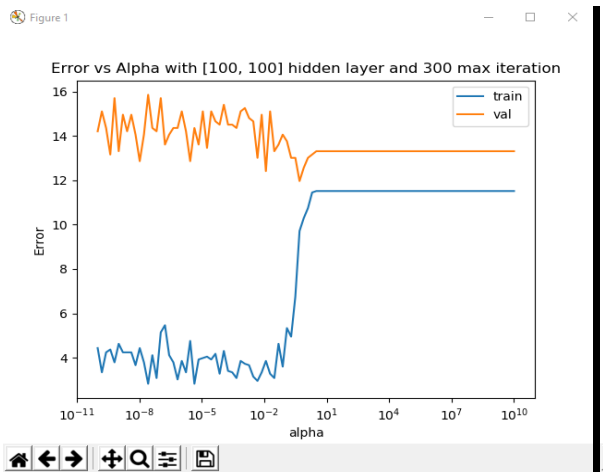


Figure 1
Error vs Alpha with [100, 100] hidden layer and 300 max iteration
train
val
Error
alpha

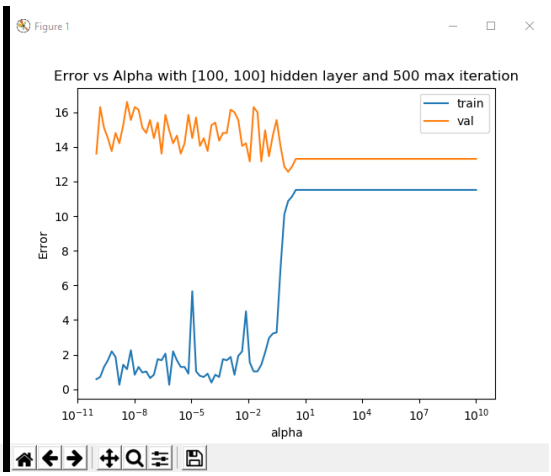


Figure 1
Error vs Alpha with [100, 100] hidden layer and 500 max iteration
train
val
Error
alpha

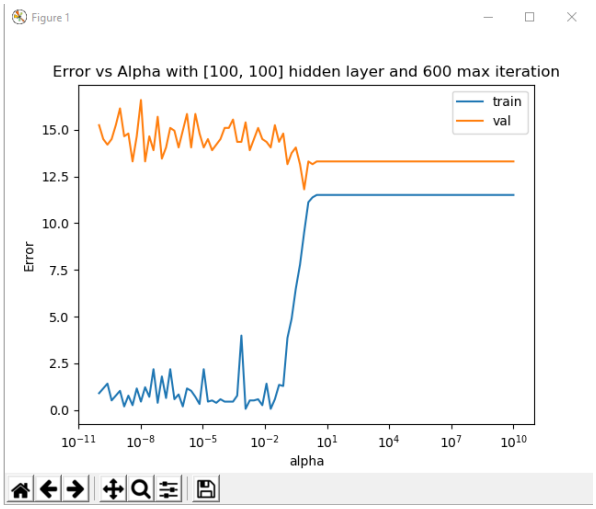
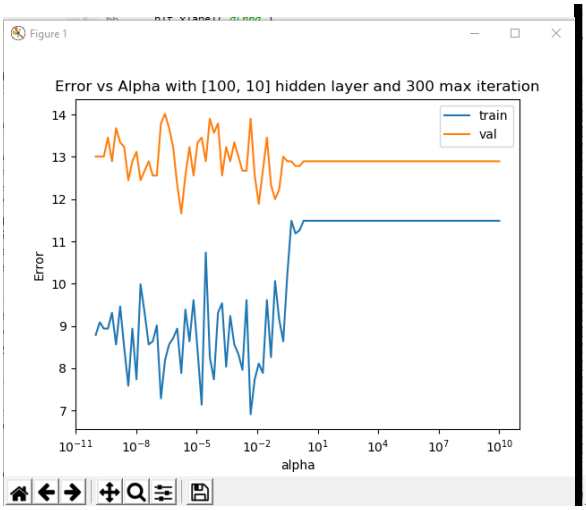
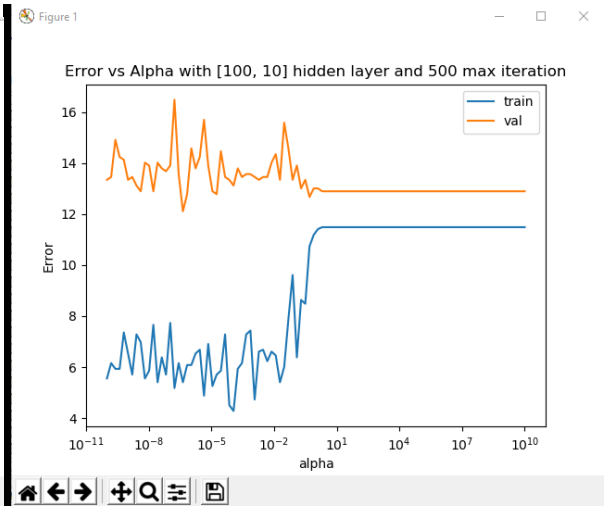
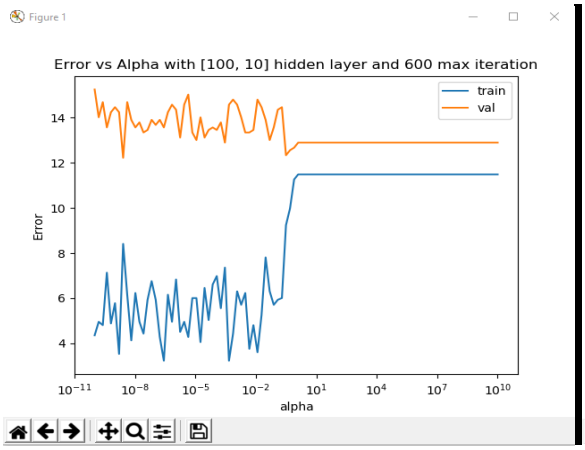


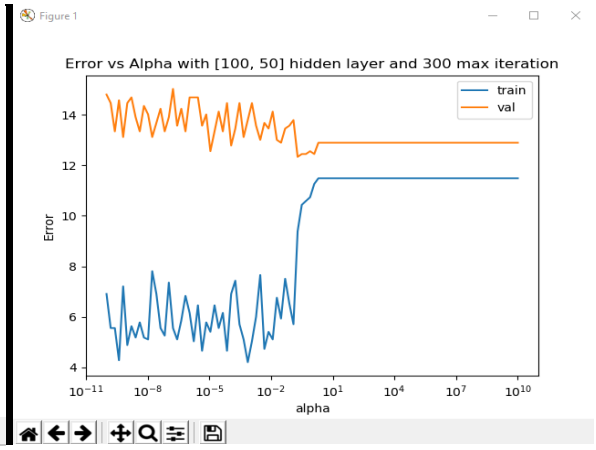
Figure 1
Error vs Alpha with [100, 100] hidden layer and 600 max iteration
train
val
Error
alpha

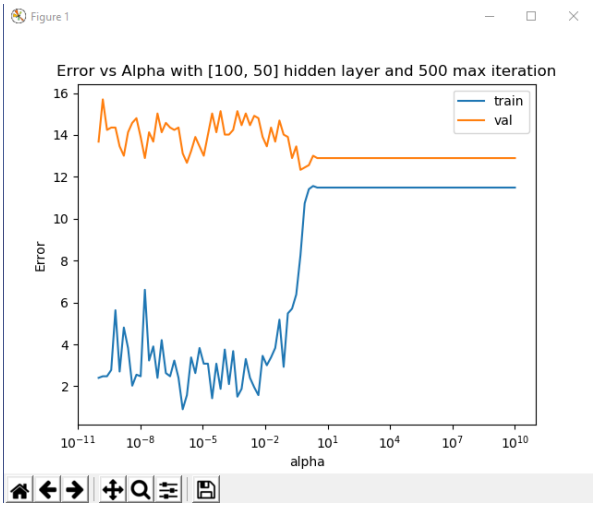
For test size %40

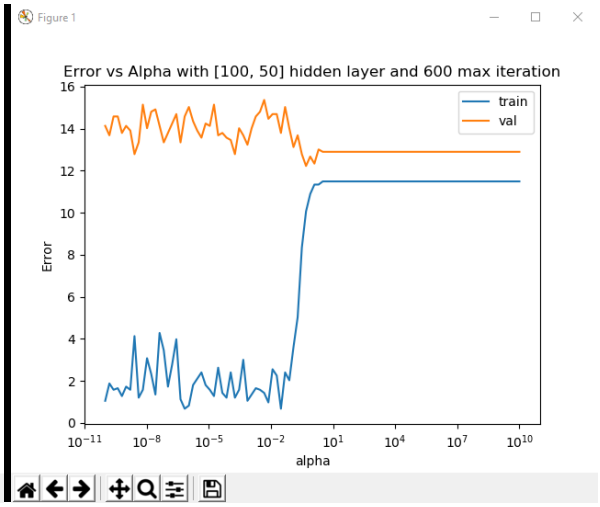


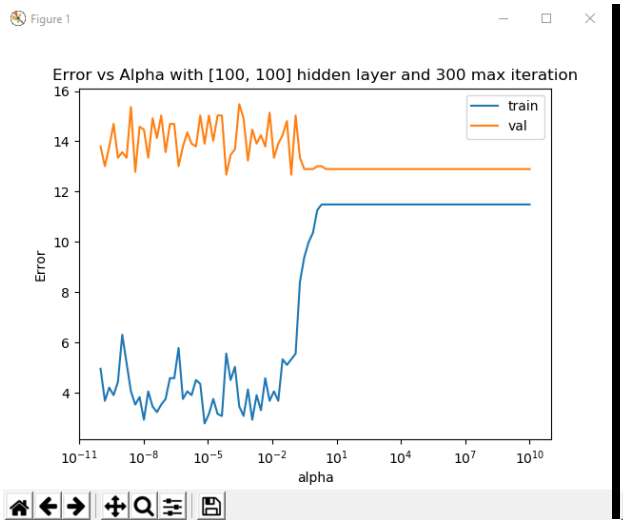


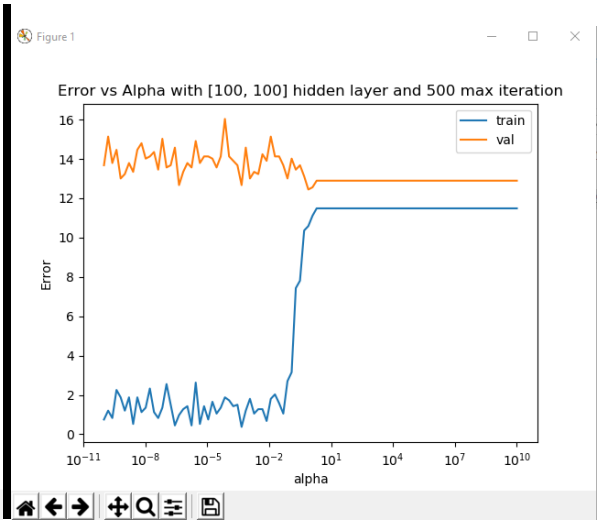


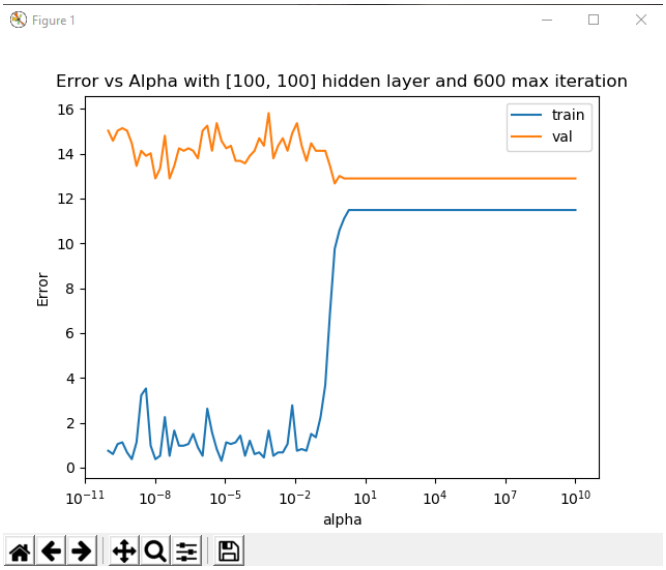












Multi-Layer Perceptron is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Alpha is L2 penalty (regularization term) parameter. Regularization term added to the loss function that shrinks model parameters to prevent overfitting[3] Which means with increasing the alpha we help reducing the overfitting.

As we can see from all the graphs, errors of both training and validation set creates waves on the small alpha values but for some alpha value the error stabilizes. In general we see a decrease in the validation error and an increase in the training error before stabilizing.

I only give the model 70% of the data and increased the size of the hidden layers, in hopes that this will cause overfitting on the training data. Therefore, with the increase of alpha a decrease in overfitting is expected.

From the graphs, changing the hidden layer size and maximum iteration does not change the behavior of the curves just makes small changes on the error values. Maximum iteration count sometimes could not be enough for convergence.

Ideally, as we increase alpha we would to see the validation accuracy increase and the training accuracy decrease somewhat (which we can see it on the graphs as validation error decreases and training error increases) until there is a point where they meet, hence reducing any overfitting.

References

[1] https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html

[3] https://scikit-learn.org/stable/modules/neural_networks_supervised.html