Technische Universität München
Fakultät für Informatik
Prof. Nassir Navab

WS 15/16
Exercise Sheet 7
December 1, 2015

Exercises in Machine learning in Medical Imaging

Exercise 1 Decision Tree

The task is to implement a decision tree which was discussed during the lecture. You will fill in the appropriate missing code in the functions `treeTrain.m`, `weakTrain.m`, `weakTest.m` and `treeTest.m`.

a) Understand the variables and data structure used for defining the tree. Specifically understand *dataix*, *leafdist*, *weakModels* and *model* and *opts*.

b) Start with function `weakTrain.m`. In this function, you have been asked to implement two weak learners - decision stump dec $= (x_r < t)$ and 2D linear decision functions dec $= (w_1 x_{r_1} + w_2 x_{r_2} + w_3 < 0)$. Further, evaluate the information gain for this decision (dec) using function `evalDecision`. The `evalDecision` function has a dependency on the function `classEntropy` which calculate the class entropy. The information gain ($\mathrm{I}_{gain}(Y)$) is given by:

$$\mathrm{I}_{gain}(Y) = \mathcal{E}(Y) - \frac{N_L}{N}\mathcal{E}(Y_L) - \frac{N_R}{N}\mathcal{E}(Y_R)$$

where N is the number of samples reaching a particular split node and N_L and N_R are the number of samples that are assigned to the left and right children nodes respectively. The class entropy $\mathcal{E}(Y)$ is evaluated as:

$$\mathcal{E}(Y) = -\sum_{\forall c} \frac{N_c}{N} log\left(\frac{N_c}{N}\right)$$

where N_c is the number of samples in Y belonging to a particular class c. Take care handle singularity when $N_c = 0$ for a particular class.

c) Having implemented `weakTrain.m`, suitably modify `weakTest.m`. These functions will be used to train and test node-level weak models while training a tree.

d) To train a tree, suitably fill in the code in the function `treeTrain.m`. The variable *dataix* will help you keep track of which samples reached what nodes. The tree data structure will be explained in the class. Fill in the appropriate arguments for `weakTrain` and `weakTest`. Upon reaching the leaf nodes, the class labels are used to define posterior predictive models at each leaf node. Generate class histogram and class posterior probability distribution function and save in the appropriate index in the variable *leafDist*.

e) Test `treeTrain` and `treeTest` functions for synthetic data by running `runDecisionTree.m`. For the first result, set `prettyspiral` variable to `true`. This generates synthetic three-class spiral data. For the second result, set `prettyspiral` to `false`. This generates synthetic twirl data. Save each of the plots. A sample result is shown in Figure 1 and Figure 2.

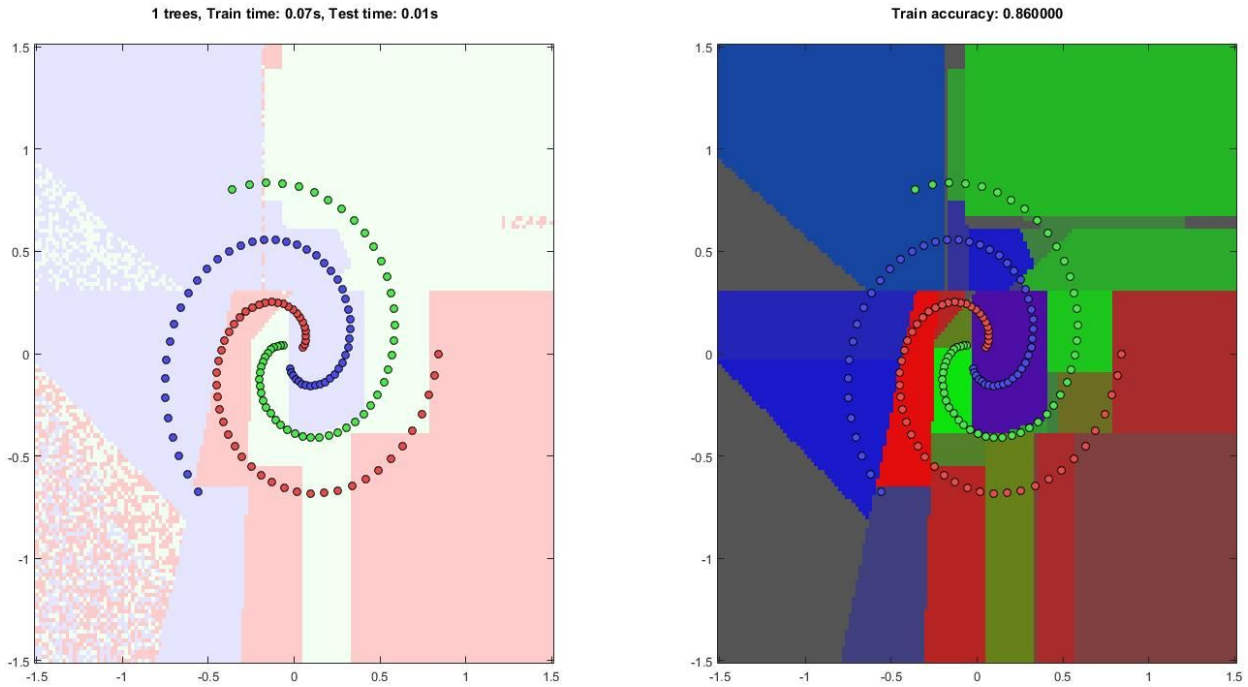

Figure 1: Result of a single tree with spiral data.

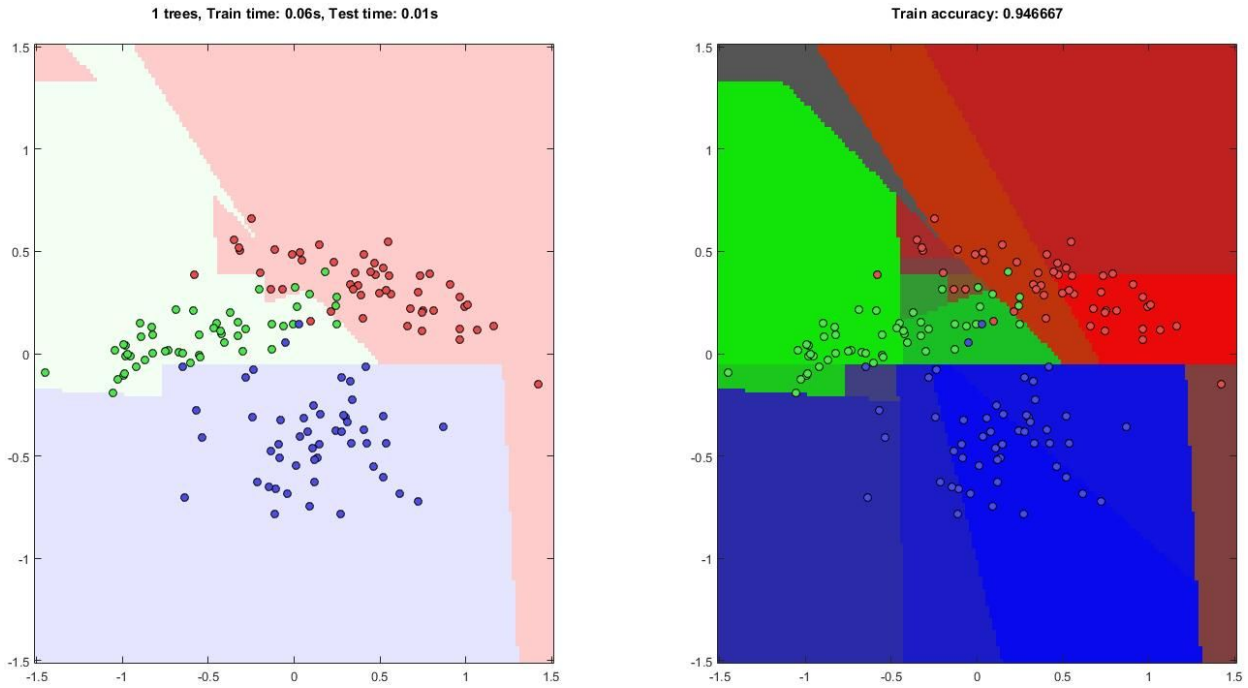

Figure 2: Result of a single tree with twirl data.

By completing these functions you have learnt how a decision tree is built and tested.

Exercise 2 Decision Forest

Having successfully implemented and tested a decision tree, you are now progressing to generate an ensemble of decorrelated decision trees to constitute a decision forest. You have to complete the code in the functions: `forestTrain.m` and `forestTest.m`. Note, these functions use `treeTrain.m` and `treeTest.m` and their implementations must be completed prior to taking up this part of the assignment.

a) Understand the datastructure used for the forest and how it interfaces with the decision

tree's datastructure. Understand hyperparameters passed through the *opts* variable.

b) Begin with `forestTrain.m`. Using $< X, Y >$, you have to generated a bootstrapped dataset X_{bag}, Y_{bag} using Monte Carlo Resampling with Replacement algorithm. Randomly select a small subset of datasamples and features and upsample it with replacement to the size of the original dataset. Then train a tree with the bagged dataset.

c) For `forestTest.m`, you pass the trained forest as variable *model* in the testing function. Fill in the code to call the `treeTest` function and get tree-wise soft predictions. Aggregate these predictions. Generate Ysoft and Yhard.

d) Test `forestTrain` and `forestTest` functions for synthetic data by running `runDecisionForest.m`. For the first result, set `prettyspiral` variable to `true`. This generates synthetic three-class spiral data. For the second result, set `prettyspiral` to `false`. This generates synthetic twirl data. Save each of the plots. A sample result is shown in Figure 3 and Figure 4.

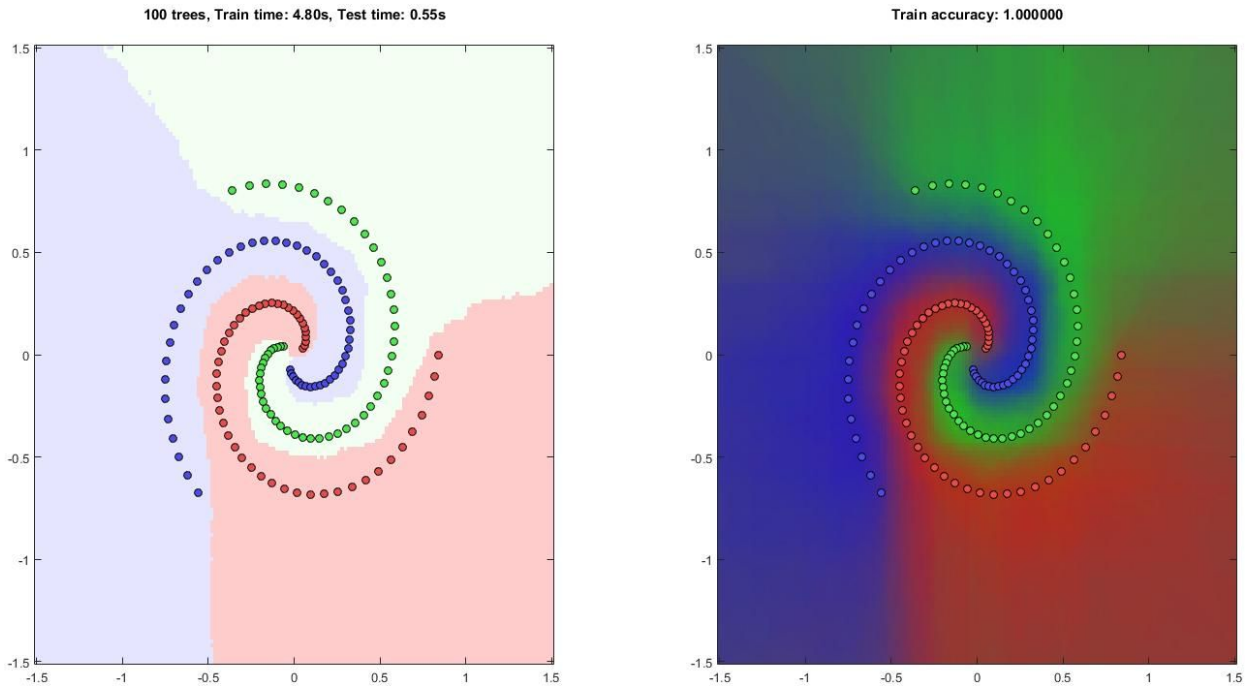

Figure 3: Result of a forest with 100 trees with spiral data.

Exercise 3 Parameter Sensitivity

Take a closer look at the code in the files `runDecisionTree.m` and `runDecisionForest.m`. There are a few important hyperparameters associated with forest that need to critically analyzed. These include:

- **Weak model at the internal nodes**: You have implemented two decision models at the split nodes : decision stumps and 2D linear decision boundaries. This is selected with variable *opts.classifierID*, which is set to '1' for decision stump and '2' for 2D linear decision model. Critically analyze the impact of this keeping the other hyperparameters fixed.

- **Tree Depth**: Change the hyperparameter (*opts.depth*) from 1 to 9 and critically analyze its impact on both a single decision tree and a forest.

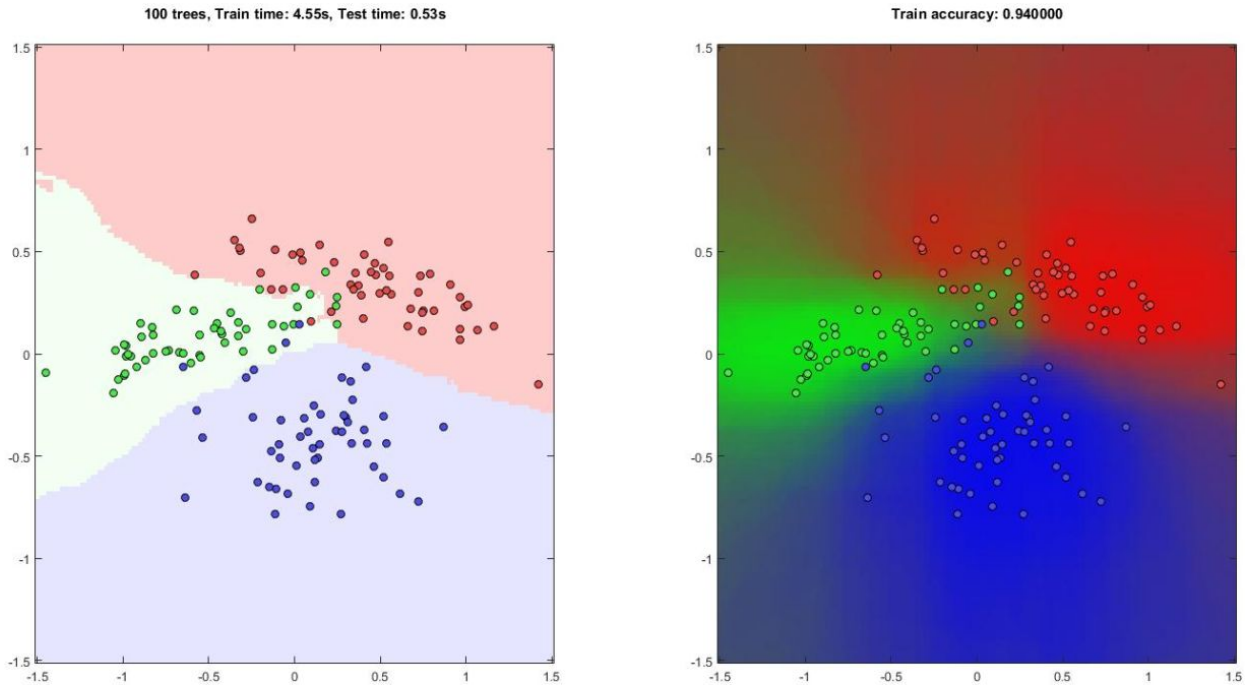

Figure 4: Result of a forest with 100 trees with twirl data.

- **Number of Trees**: Change the hyperparameter (*opts.numTrees*) from 5 to 100 in increments of 5 and critically analyze its impact on both a single decision tree and a forest (keep other hyperparameters fixed).

Plot performance characteristics as training accuracy *vs.* changing hyperparameters for both tree depth and number of trees in the ensemble.