ML05: Decision trees and ensemble models - example outputs

Petr Pošík, Dept. of Cybernetics, FEE CTU in Prague

B(E)3M33UI - Umělá inteligence / Artificial Intelligence

- This document accompanies the specification of ML05 tasks and presents example of the outputs which are expected from the students.
- **Your results may be different from what we present here;** it depends on the random seed, on the order of the operations, etc. It does not necesarilly mean that your results are wrong. The results below are generated with random seed 1234.
- Your way of displaying information may be different.
- The actual code that produces these results is hidden inside function calls, and will not be made available. It is the task of the students to develop it.

Preparation

First, an initial setup, import of our module with the solution, and setup of the random seed for repeatable results (this wouldn't be done in practice).

In [1]:
```
%matplotlib inline
from ML05_solution import *
np.random.seed(1234)
```

And load the data.

In [2]:
```
X, y, Xtr, ytr, Xtst, ytst = load_and_prepare_data()
```

```
Complete data : X (392, 2) , y (392,) , sum(y == 0) = 147 , sum(y == 1) = 245
Training data : X (313, 2) , y (313,) , sum(y == 0) = 117 , sum(y == 1) = 196
Testing data : X (79, 2) , y (79,) , sum(y == 0) = 30 , sum(y == 1) = 49
```

The data contain 392 examples: 147 in class 0, 245 in class 1. This dataset is split 80:20 to training and testing data.

Task 1

Use the grid search to find optimal parameter settings for a decision tree model, and plot its validation curve. Here we will tune just the `max_depth` parameter based on 5-fold crossvalidation on the training dataset. Then we shall plot the validation curves, i.e. errors on training and testing data wrt the `max_depth` parameter.

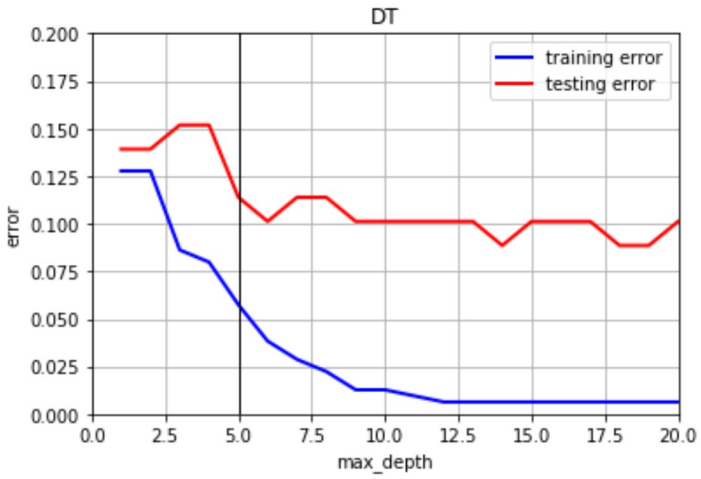
In [3]:
```
m1 = DecisionTreeClassifier()
m1_name = 'DT'
m1_param_grid = {'max_depth': range(1, 21)}
p1 = task_GS_val_curve(m1, m1_name, m1_param_grid, Xtr, ytr, Xtst, ytst)
```

```
==== Grid Search and Validation Curve for DT
Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.2s finished

Best parameters found: {'max_depth': 5}
CV score for the best parameter values: 0.9230382744495648
Computing validation curve for max_depth in range(1, 21)

<Figure size 432x288 with 0 Axes>
```



Note that using 5-fold cross-validation to assess 20 values of parameter requires us to fit the model 100 times.

In this particular run, the grid search with crossvalidation on training data found that the best value of `max_depth` is 5. You may wonder why this value does not correspond to the minimum of the red curve in the validation graph. The reason is that the grid search uses crossvalidated error estimate, while the red curve estimates the error using testing data, i.e. the errors are estimated using different data, and that's why they are not the same.

But even from the validation curves it seems that increasing the `max_depth` to values higher than 6 (this value may be different for other random seeds) will not bring much in terms of testing error, while it would cause decrease of the training error, i.e. the model would become overfitted.

Task 2

Now we shall repeat the analysis for the AdaBoost algorithm. Here we will tune only the number of weak models in the whole model, parameter `n_estimators` .

In [4]:
```
m2 = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1))
m2_name = 'AdaBoost'
m2_param_grid = {'n_estimators': range(1, 61)}
p2 = task_GS_val_curve(m2, m2_name, m2_param_grid, Xtr, ytr, Xtst, ytst)
```

```
==== Grid Search and Validation Curve for AdaBoost
Fitting 5 folds for each of 60 candidates, totalling 300 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done 300 out of 300 | elapsed:   23.3s finished

Best parameters found: {'n_estimators': 58}
CV score for the best parameter values: 0.9041378648233487
Computing validation curve for n_estimators in range(1, 61)

<Figure size 432x288 with 0 Axes>
```

Note that using 5-fold cross-validation to assess 60 values of parameter requires us to fit the model 300 times.

Again, the found number of estimators (58 here) may be different in your case due to random effects. But you can see from the validation curves that in case of AdaBoost increasing the number of weak models does not harm the model much. Based on the validation curve, one could argue that we may have chosen an ensemble of 18-20 models, i.e. simpler model with more or less the same quality. But if we changed the chosen model to one with say 18 estimators, we would actually use the testing data for training which is not desirable. We would then have to evaluate the chosen model again on a completely different dataset to get an unbiased error estimate.

Task 3

And, as the last competitor, we shall repeat the process for the random forest model. Here, we will tune not only the number of trees in the ensemble, but also the maximal depth of the trees. We will also plot the validation curves wrt to both these parameters. This part may take a while due to testing all pairs of parameter values.

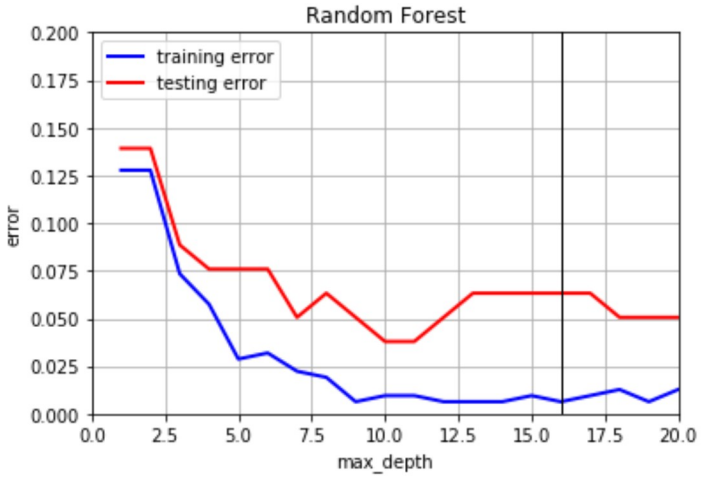
In [5]:
```
m3 = RandomForestClassifier()
m3_name = 'Random Forest'
m3_param_grid = {'max_depth': range(1, 21),
                 'n_estimators': range(1, 31)}
p3 = task_GS_val_curve(m3, m3_name, m3_param_grid, Xtr, ytr, Xtst, ytst)
```

```
==== Grid Search and Validation Curve for Random Forest
Fitting 5 folds for each of 600 candidates, totalling 3000 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done 3000 out of 3000 | elapsed:  1.7min finished

Best parameters found: {'max_depth': 16, 'n_estimators': 23}
CV score for the best parameter values: 0.9296402969790065
Computing validation curve for max_depth in range(1, 21)
Computing validation curve for n_estimators in range(1, 31)

<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

Using 5-fold cross-validation to assess 20x30 combinations of parameter values requires us to fit the model 3000 times.

Again, you can see that the best parameter values found by cross-validation do not coincide with the parameter values for which we measured minimal testing error.

Task 4

In this task we shall compare the test set errors of the three types of models with the found parameter values.

In [6]:
```
models = [m1, m2, m3]
names = [m1_name, m2_name, m3_name]
task_model_accuracies(models,
                          names,
                          [p1, p2, p3],
                          Xtr, ytr, Xtst, ytst)
```

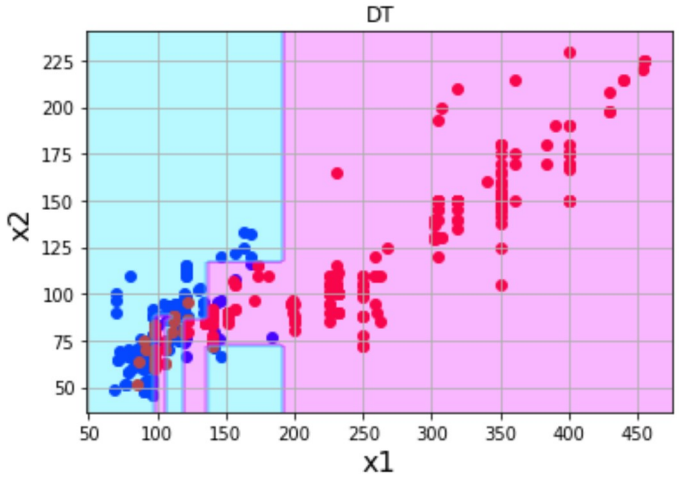
```
==== Accuracy on test data:
Model: DT Accuracy: 0.8860759493670886
Model: AdaBoost Accuracy: 0.9240506329113924
Model: Random Forest Accuracy: 0.9367088607594937
```

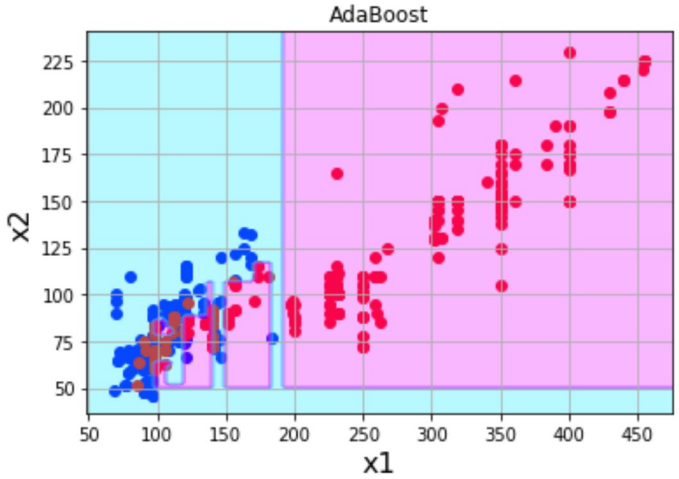
Comparing the three models on testing data shows that Random Forest seems to be best, followed by AdaBoost and DT. Again, it may easily happen on these data that the ranking will be different in your case.

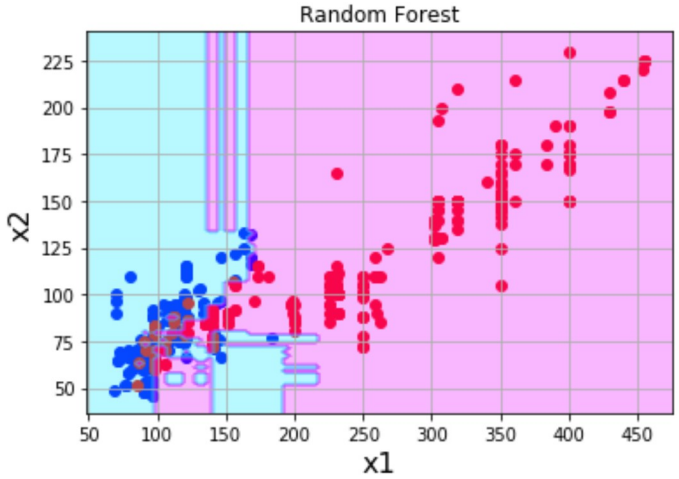
Task 5

Plot the decision boundaries for all three models.

In [7]:
```
task_decision_boundaries(models, names, X, y)
```







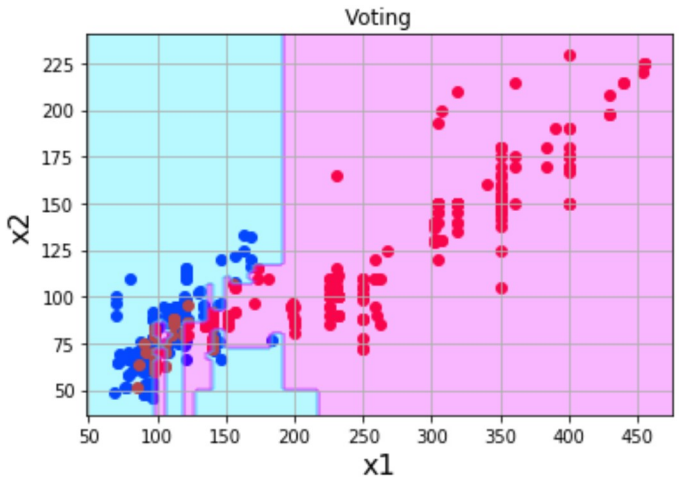
Comparing the decision boundaries, you can see that the models differ a lot in areas not covered by data points. This may play a role if we obtained new data. This situation was actually simulated with the available data by splitting them into training and testing.

Task 6

Here we should combine the models into an simple ensemble using a VotingClassifier. For the three models, we shall use the best parameters found by cross-validation above. We also print the accuracies of the constituent models to show that they are the same as above.

In [8]:
```
task_voting(models, names, X, y, Xtr, ytr, Xtst, ytst)
```

```
Model: DT , Accuracy: 0.8860759493670886
Model: AdaBoost , Accuracy: 0.9240506329113924
Model: Random Forest , Accuracy: 0.9367088607594937
Model: Voting Accuracy: 0.9240506329113924
```



Above, we reported once more the testing accuracy of the constituent models to check that they are the same as before. The voting applied to these three classifiers on the current dataset actually does not improve the testing accuracy. But if you compare the decision boundary of voting with that of the original models, we can see that the resulting model is more regular, less rugged. This should be again beneficial for generalization to new, unseen data.