Introduction to Data Analysis and Machine Learning in Physics: 4. Decisions Trees

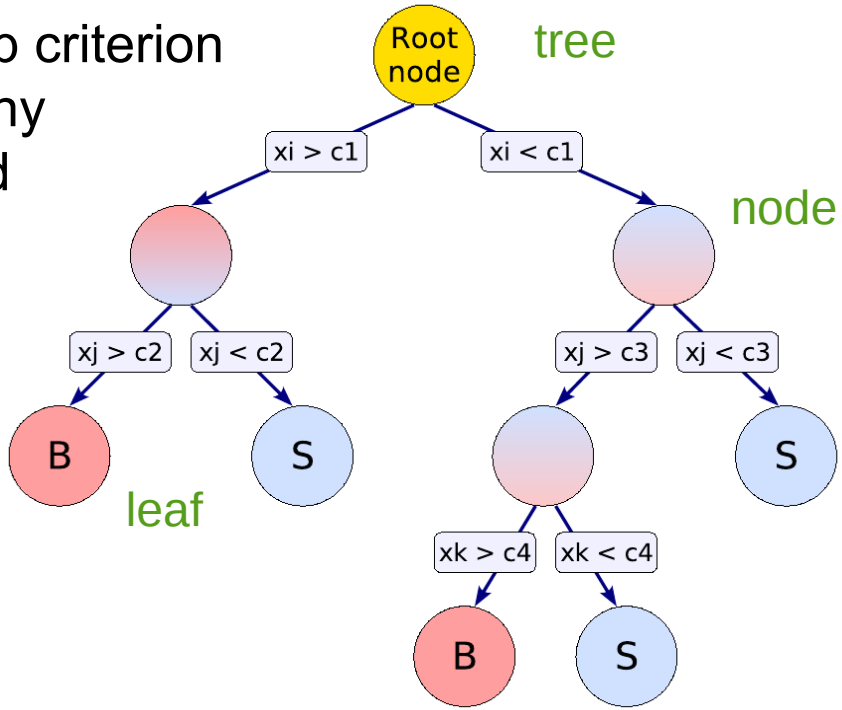
Jörg Marks

Studierendentage, 8-12 April 2024

Decision Trees

- Decision tree

  is a binary tree like decision structure for a set of variables. For each variable a decision is taken until a stop criterion is reached. The variable space is divided in many regions until a classification in S or B is reached



- Advantage

  - Easy to understand
  - Suitable for multivariate analyses
  - Simple and fast training

- Disadvantage

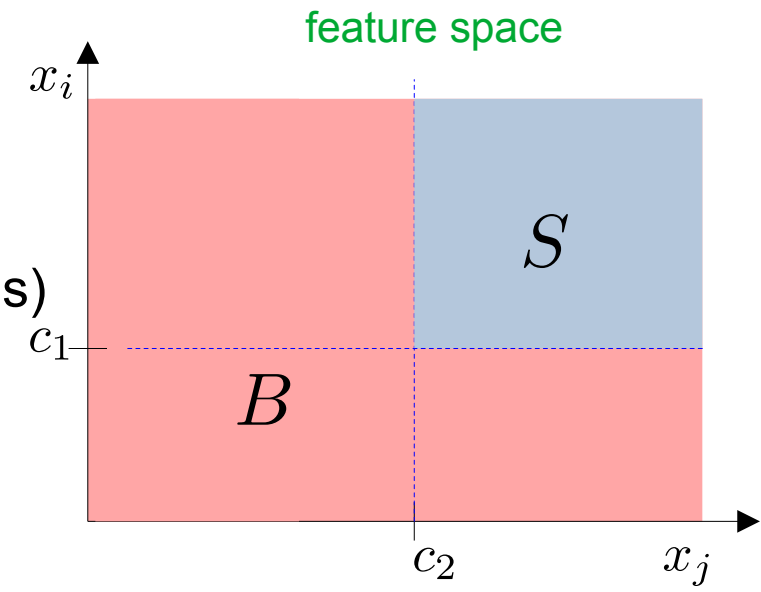
  - Sensitive to statistical fluctuations in the data
  - One tree has very limited (poor) classification results
  - Difficult to find a global minimum (best result)

→ other methods are needed, use many decision trees (forest)
Boosted Decision Tree (BDT)

Decision Trees

- Decision tree

  is a binary tree like decision structure for a set of variables. For each variable a decision is taken until a stop criterion is reached. The variable space is divided in many regions until a classification in S or B is reached



- Advantage

  - Easy to interpret and visualize (rectangular volumes)
  - Suitable for multivariate analyses
  - Simple and fast training

- Disadvantage

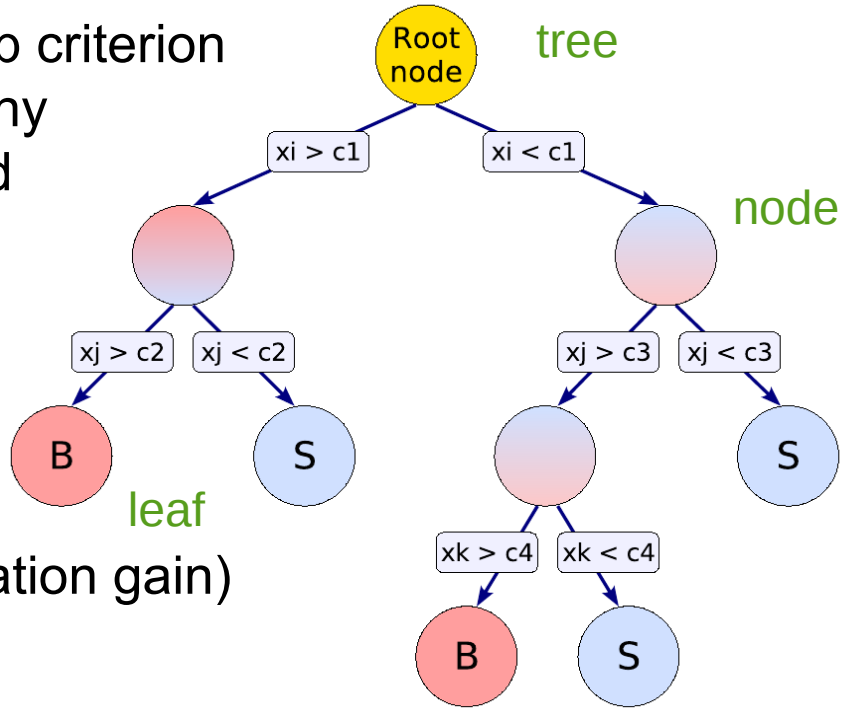
  - Sensitive to statistical fluctuations in the data
  - One tree has very limited (poor) classification results
  - Difficult to find a global minimum (best result)

  → other methods are needed, use many decision trees (forest)
  Boosted Decision Tree (BDT)

Decision Trees

- Decision tree

  is a binary tree like decision structure for a set of variables. For each variable a decision is taken until a stop criterion is reached. The variable space is divided in many regions until a classification in S or B is reached



- Stop criterion

  - Lower limit of events per node reached
  - Upper limit of the number of nodes reached
  - Maximum depth reached
  - Further division gives no improvement (separation gain)

- Separation gain

  - Define a measure for S and B in one node

    Gini index: $G = p \cdot (1 - p), \quad p \equiv$ purity (G small → good Separation)

$$p = \frac{\sum_{signal} S_i \cdot w_i}{\sum_{signal} S_i \cdot w_i + \sum_{background} B_i \cdot w_i}$$

- Separation gain: $N_{parent} \cdot G_{parent} - N_{left} \cdot G_{left} - N_{right} \cdot G_{right}$

  → Maximize

Boosted Decision Trees

- Combination of weak learning trees (algorithms)

  Describes an improvement in classification results of MVA algorithmen by combining (reweighted) training samples. The result is the weighted sum. This combination results typically in a big improvement compared to the single result.

Boosted Decision Trees

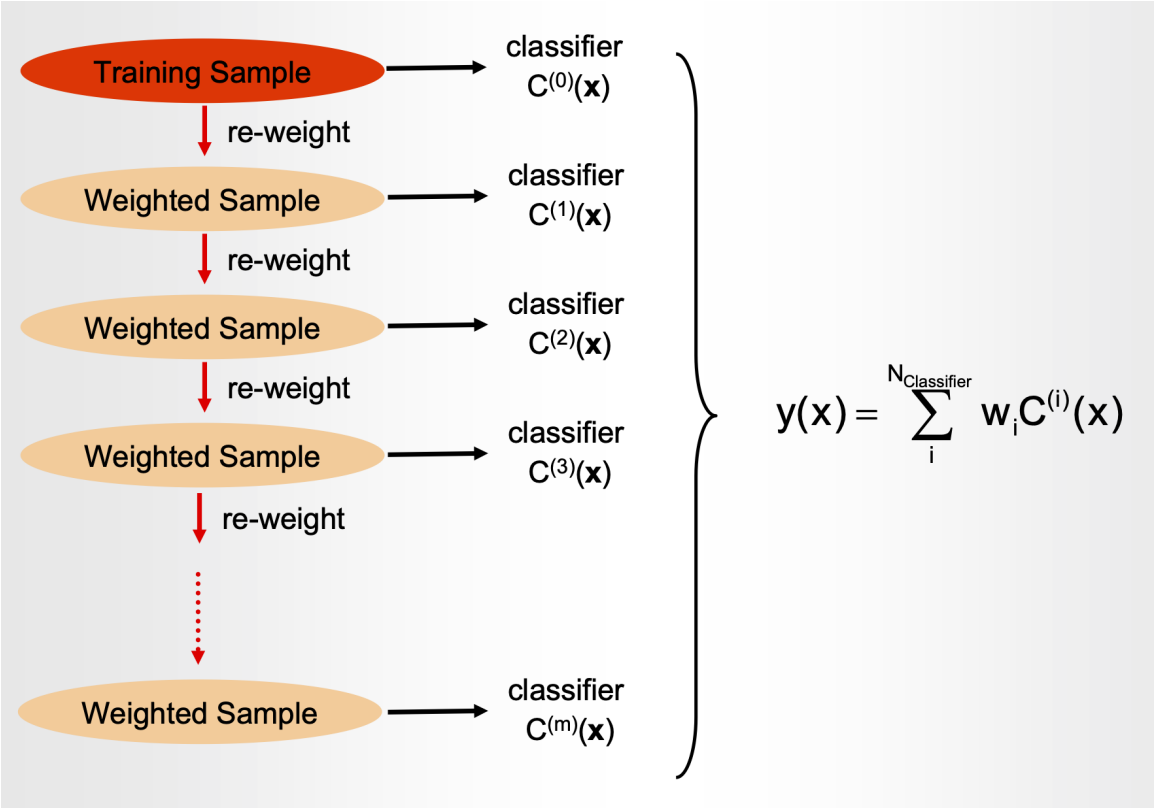
- Combination of weak learning trees (algorithms)

Describes an improvement in classification results of MVA algorithmen by combining (reweighted) training samples. The result is the weighted sum. This combination results typically in a big improvement compared to the single result.

- Bootstrap aggregation (Bagging)

Sample training data and train a separate model on each of the training sets.

$$y_{Bagging}(\vec{V}) = \frac{1}{N_{collection}} \sum_{i}^{N_{collection}} F_i(\vec{V})$$



Boosted Decision Trees

- Combination of weak learning trees (algorithms)

Describes an improvement in classification results of MVA algorithmen by combining (reweighted) training samples. The result is the weighted sum. This combination results typically in a big improvement compared to the single result.

- Bootstrap aggregation (Bagging)
  Sample training data and train a separate model on each of the training sets.

$$y_{Bagging}(\vec{V}) = \frac{1}{N_{collection}} \sum_{i}^{N_{collection}} F_i(\vec{V})$$

- Random forest
  Use bagging to select random subsets and then train a tree selecting a random subset of the features (variables)

  → reduces the correlation between different trees
  more robust against missing data

Boosted Decision Trees

- Combination of weak learning trees (algorithms)

Describes an improvement in classification results of MVA algorithmen by combining (reweighted) training samples. The result is the weighted sum. This combination results typically in a big improvement compared to the single result.

- Bootstrap aggregation (Bagging)
  Sample training data and train a separate model on each of the training sets.

$$y_{Bagging}(\vec{V}) = \frac{1}{N_{collection}} \sum_{i}^{N_{collection}} F_i(\vec{V})$$

- Random forest
  Use bagging to select random subsets and then train a tree selecting a random subset of the features (variables)

- AdaBoost (adaptive boosting) Freund & Schapire, 1996
  mis-classified data gets in the training of the following trees a larger weight α , which is determined via the mis-classification rate err

$$\alpha = \frac{1 - err}{err}$$

The weight are normalized such that the sum of the weights is constant

$$y_{Boost}(\vec{V}) = \frac{1}{N_{collection}} \sum_{i}^{N_{collection}} ln(\alpha_i) \cdot F_i(\vec{V})$$

y_{Boost} Total weighted classifier

$N_{collection}$ Number of the reweighted training samples

Boosted Decision Trees

- Combination of weak learning trees (algorithms)

Describes an improvement in classification results of MVA algorithmen by combining (reweighted) training samples. The result is the weighted sum. This combination results typically in a big improvement compared to the single result.

- Gradient boosting

Basic idea:

- Train a first decision tree
- Then train a second one on the residual errors made by the first tree
- And so on

In slightly more detail:

- Consider labeled training data: $\{\boldsymbol{x}_i, y_i\}$
- Model prediction at iteration m: $F_m(\boldsymbol{x}_i)$
- New model: $F_{m+1}(\boldsymbol{x}) = F_m(\boldsymbol{x}) + h_m(\boldsymbol{x})$
- Find $h_m(\boldsymbol{x})$ by fitting it to $\{(\boldsymbol{x}_1, y_1 - F_m(\boldsymbol{x}_1)),\ (\boldsymbol{x}_2, y_2 - F_m(\boldsymbol{x}_2)),\ \ldots (\boldsymbol{x}_n, y_n - F_m(\boldsymbol{x}_n))\}$

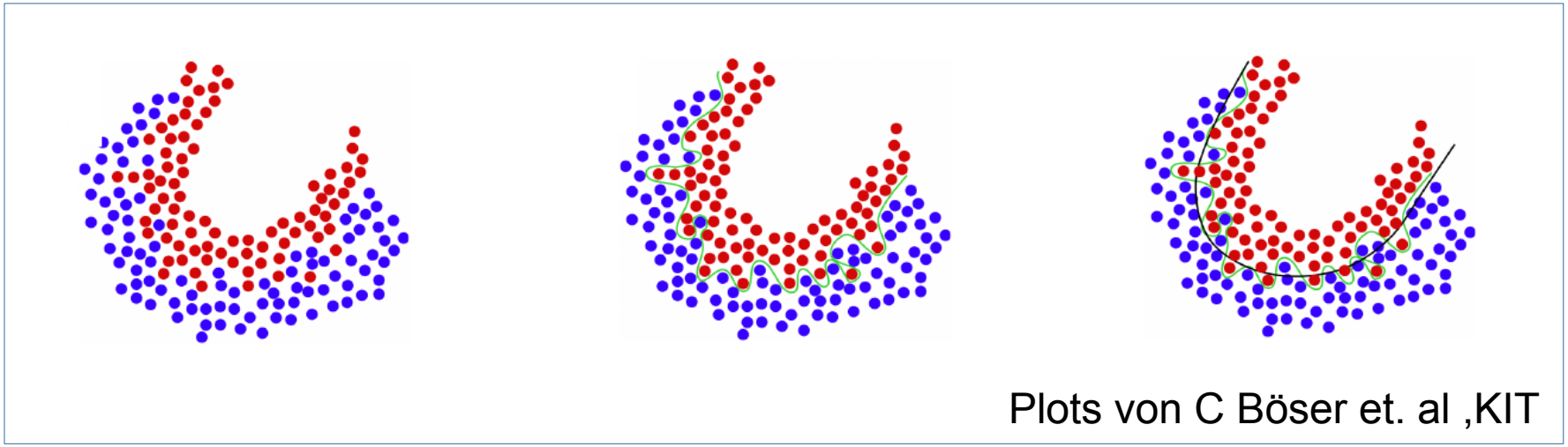
Boosted Decision Trees

- Overtraining

A perfect boosted decision tree would have in all nodes B about zero, but that means also the statistical fluctuations were learned as properties of the training sample

- splitting the data set into a training and a test set and testing only on the test set helps to control this. Control plots of the output variables are compared for both sets, if they are similar there is no overtraining.

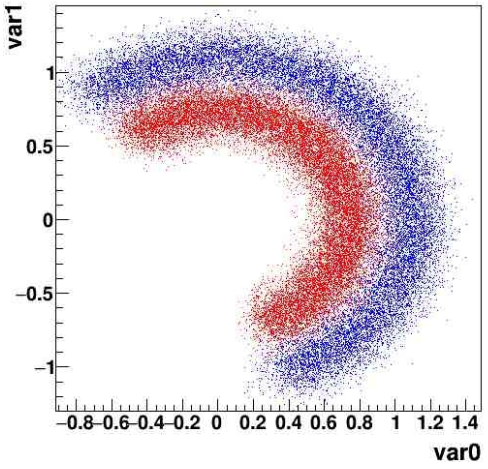
- Cutting back insufficient nodes avoid overtraining, it is called pruning

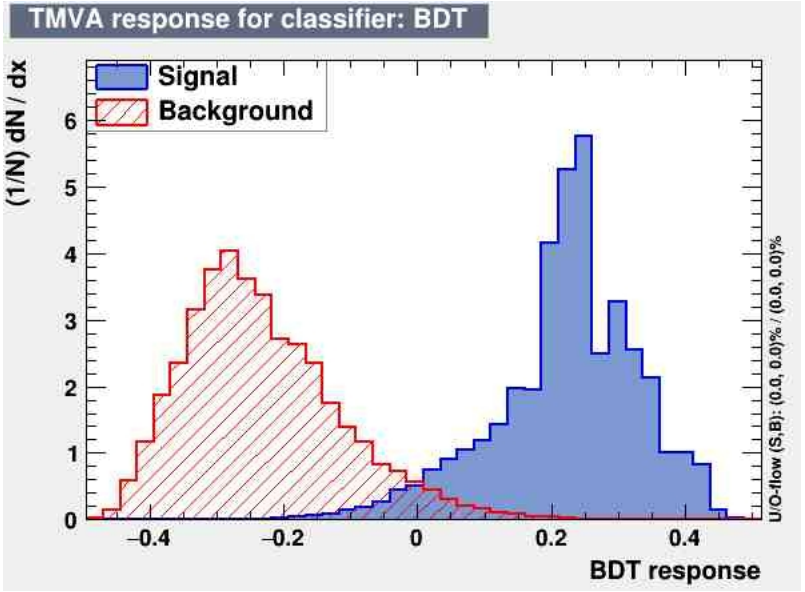


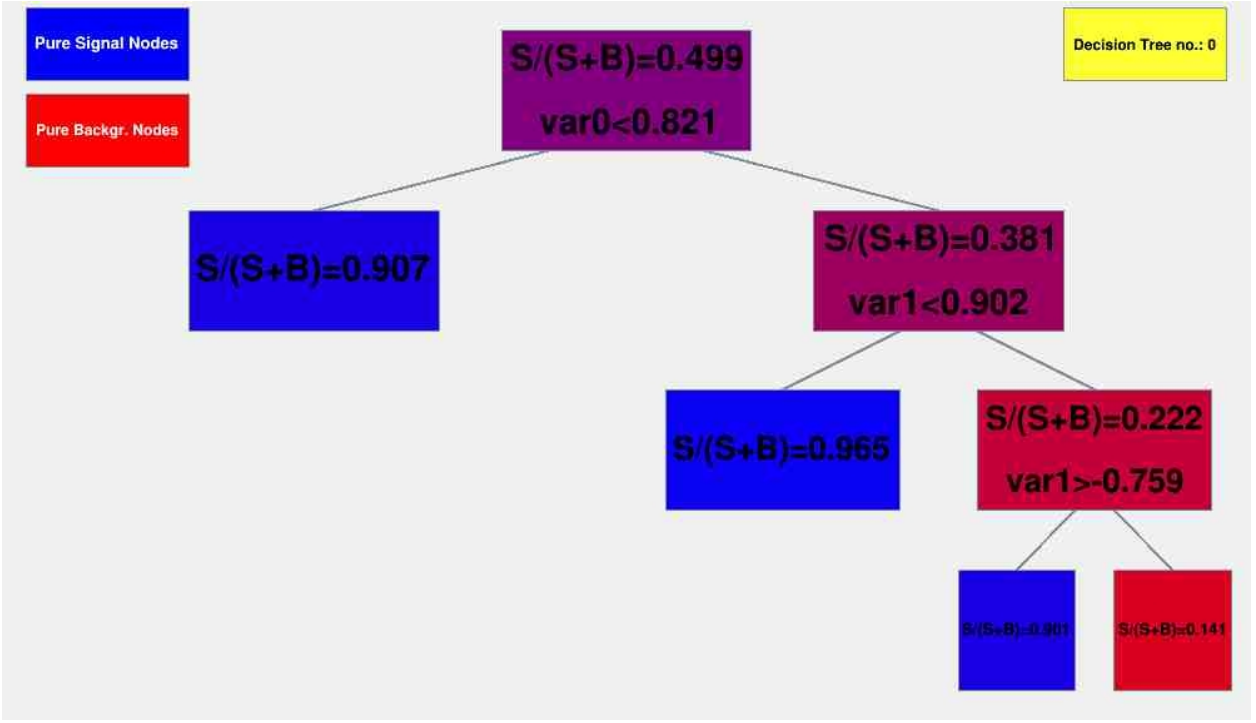
Plots von C Böser et. al ,KIT

Boosted Decision Trees

- Example: semicircles







XGBoost

- Example with a famous library

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. (from the XGBoost documentation https://xgboost.readthedocs.io/en/stable/)

Predict critical temperature for superconductivty (Regression with XGBoost)

- Superconductivty data set:
  https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data

Predict the critical temperature based on 81 material features.

From the abstract:

We estimate a statistical model to predict the superconducting critical temperature based on the features extracted from the superconductor's chemical formula. The statistical model gives reasonable out-of-sample predictions: ±9.5 K based on root-mean-squared-error. Features extracted based on thermal conductivity, atomic radius, valence, electron affinity, and atomic mass contribute the most to the model's predictive accuracy.

https://doi.org/10.1016/j.commatsci.2018.07.052

04_decision_trees_critical_temp_regression.ipynb

XGBoost

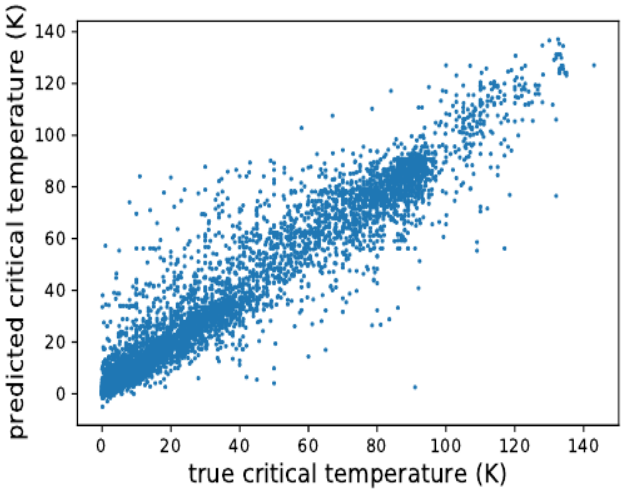
- Example with a famous library

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. (from the XGBoost documentation https://xgboost.readthedocs.io/en/stable/)

```
import xgboost as xgb

XGBreg = xgb.sklearn.XGBRegressor()

XGBreg.fit(X_train, y_train)

y_pred = XGBreg.predict(X_test)

from sklearn.metrics import mean_squared_error
rms = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"root mean square error {rms:.2f}")
```

This gives:

```
root mean square error 9.68
```



Exercise 1

- Compare different decision tree classifiers

  Use the heart desease dataset we already discused and compare different classifiers from scikit-learn: `AdaBoostClassifier, RandomForestClassifier, GradientBoostingClassifier`

  a) Plot the ROC curves and decide which algorithm is best.

  04_decision_trees_ex_1_compare_tree_classifiers.ipynb

  Solution:
  04_decision_trees_ex_1_sol_compare_tree_classifiers.ipynb

Exercise 2

- Apply the XGBoost classifier to the MAGIC data set

```
# train XGBoost boosted decision tree
import xgboost as xgb
XGBclassifier = xgb.sklearn.XGBClassifier(nthread=-1,seed=1,
                                          n_estimators=1000)
```

04_decision_trees_ex_2_magic_xgboost_and_random_forest.ipynb

a) Plot predicted probabilities for the test sample for signal and background events (plt.hist)

b) Which is the most important feature for discriminating signal and background according to XGBoost? Hint: use plot_impartance from XGBoost (see XGBoost plotting API). Do you get the same answer for all three performance measures provided by XGBoost ("weight", "gain", or "cover")?

c) Visualize one decision tree from the ensemble (let's say tree number 10). For this you need the the graphviz package (`conda install graphviz`)

d) Compare the performance of XGBoost with the random forest classifier from scikit-learn. Plot signal and background efficiency for both classifiers in one plot. Which classifier performs better?

04_decision_trees_ex_2_sol_magic_xgboost_and_random_forest.ipynb