Data mining and machine learning

dsminingf17vm

Physics MSc course

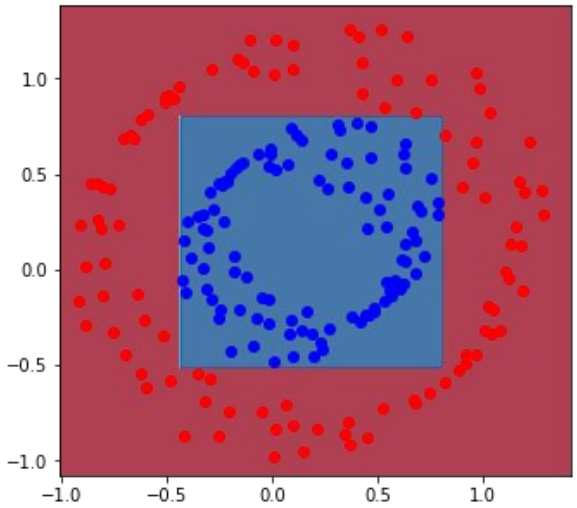
08 - Decision trees, random forest, ensemble, boosting

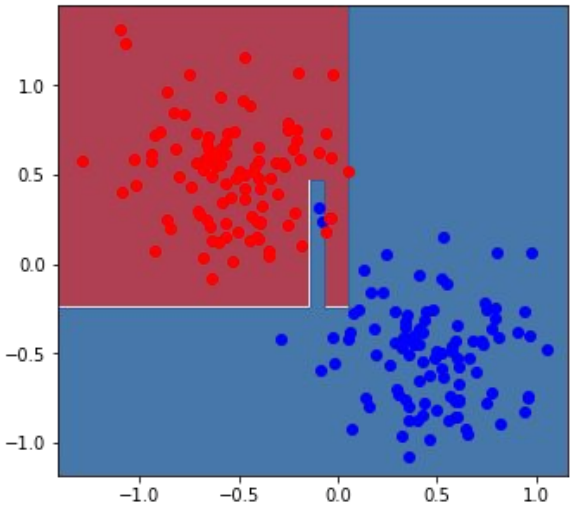
Pataki Bálint Ármin
ELTE, Physics of Complex Systems Department
2020.11.02.

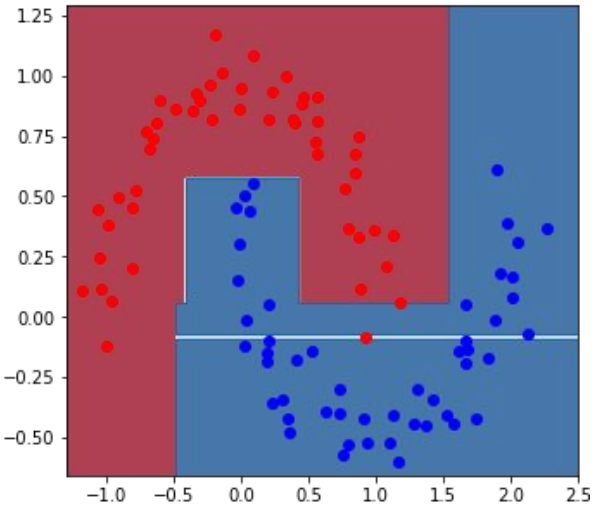
The idea behind trees

- KNN: using the y values of the closest neighbours
- Trees: divide the input space into regions with constant y prediction
  - Division along the axes (single parameter)
  - Nested divisions

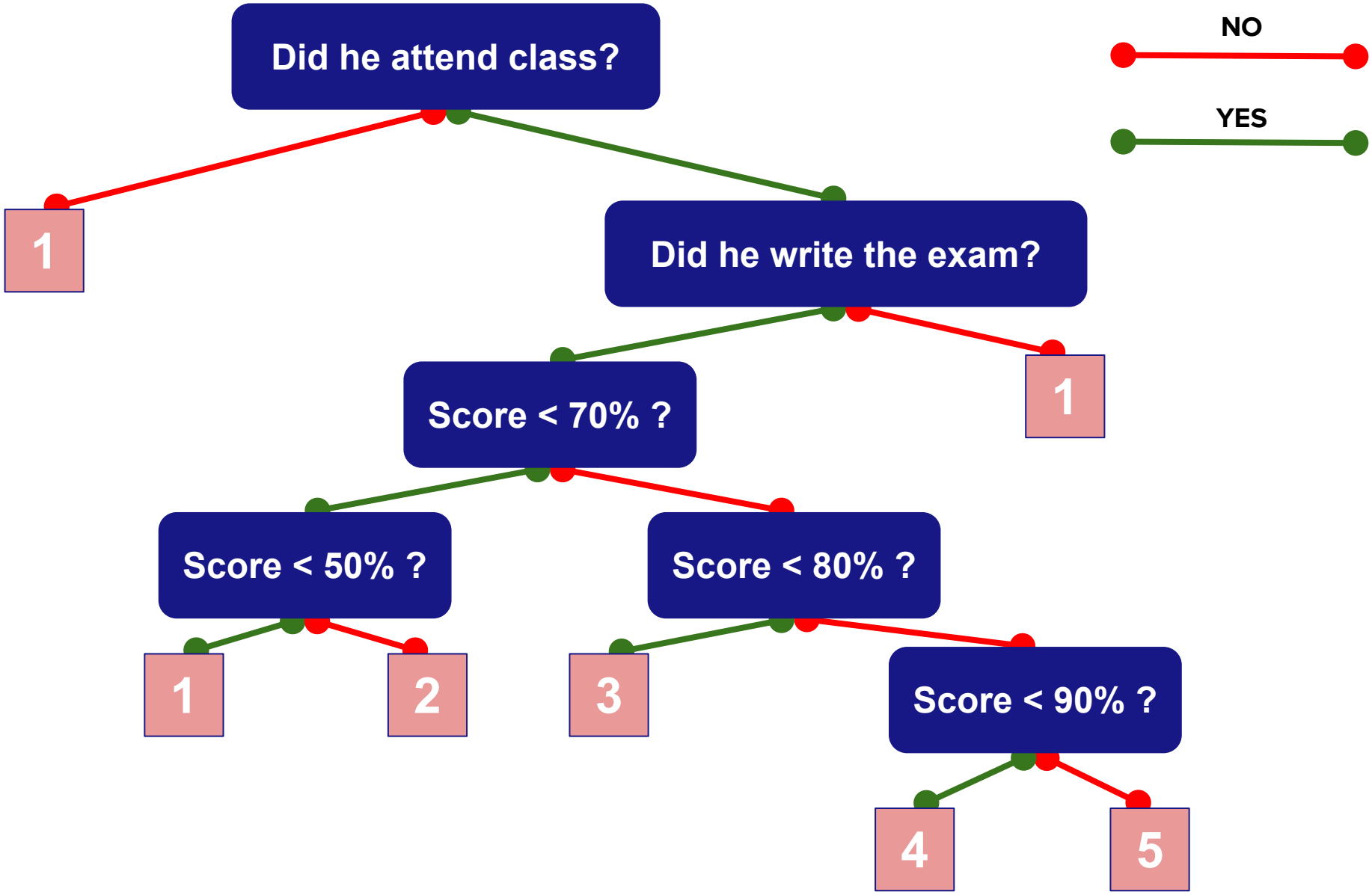
Intuitive, similar to human thinking. Easy to interpret!







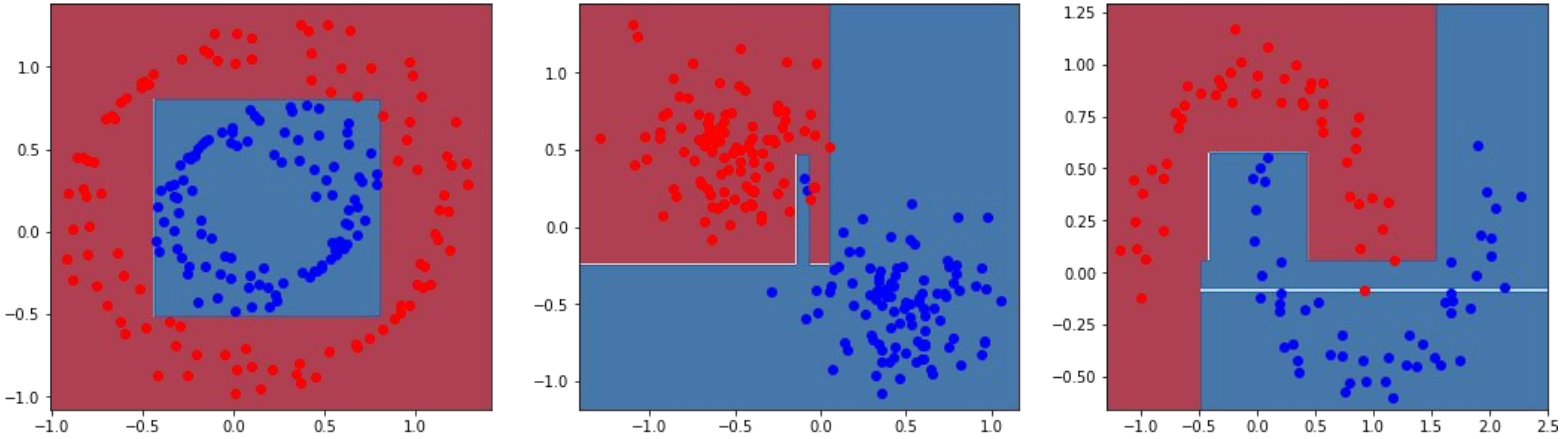
Generating prediction with a decision tree



Decision tree

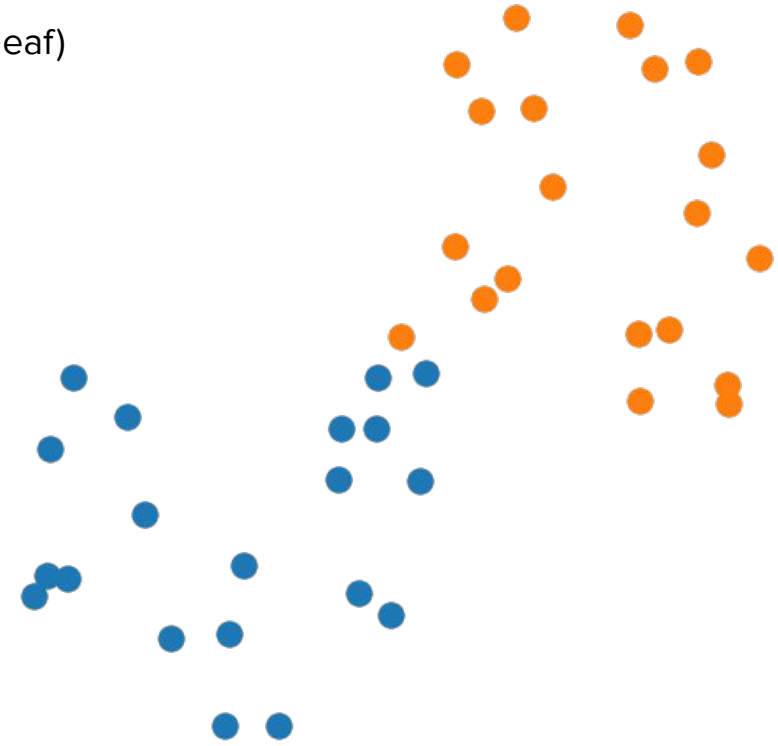
- Splitting the input space into regions with constant predicted value
  - Division along the axes (single parameter)
  - Easy to interpret

- Prediction
  - Regression
    - Mean of the train y values in the region of the test point
  - Classification
    - Majority voting
    - Probability: empirical frequency of the classes



What is the best split for a feature?

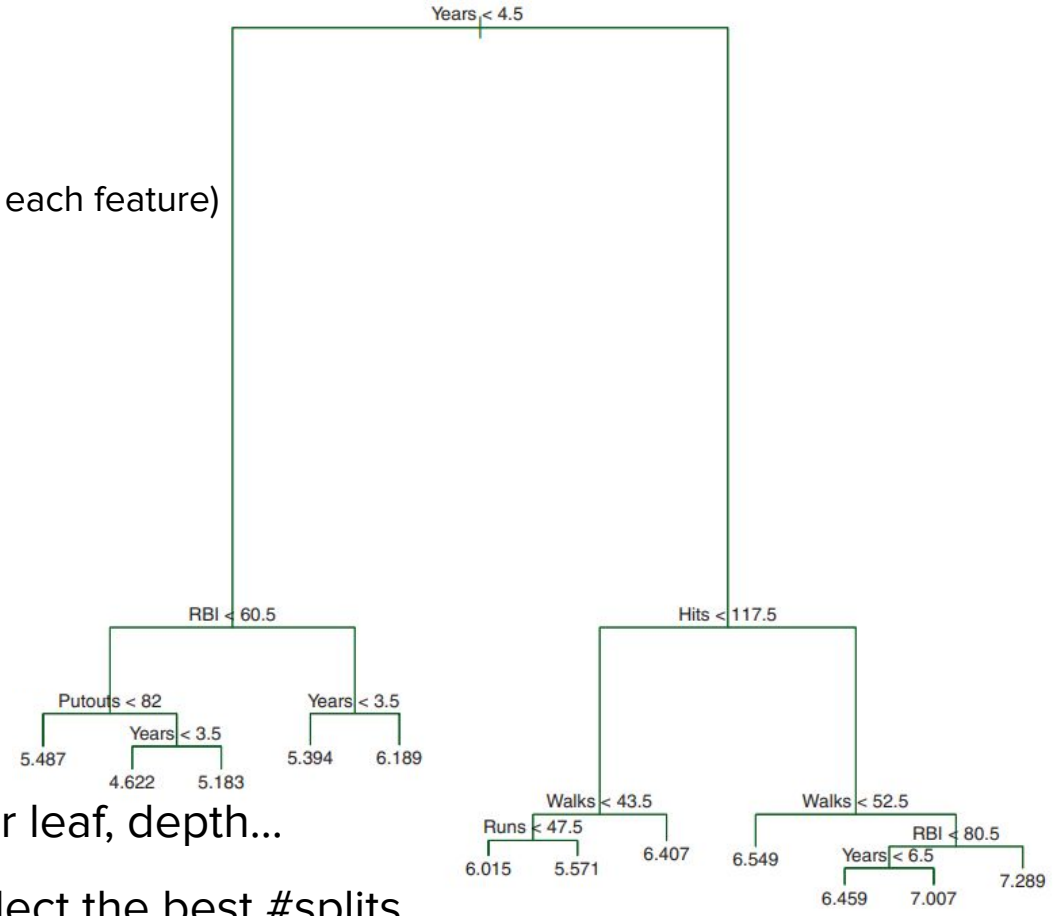
- Regression
  - Minimize MSE of predictions (using the mean of y on a leaf)
  - Minimize MAE of predictions (using the median)



- Classification
  - Minimize Gini impurity
    - $I_G = 1 - \sum_{c=1}^{C} p_c^2$, p_c is the empirical prob.
  - Maximize information gain (entropy)

How to grow a tree?

- Too many possible trees to consider
- Recursive greedy binary splitting
  - Start with the whole dataset
  - Consider each possible binary split (for each feature)
  - Calculate split criterion for each
  - Select the best
  - Repeat



When to stop?

a.) grow a full tree (1 data point per leaf)

b.) grow until a pre-defined condition

Number of leafs, number of points per leaf, depth...

c.) cross-validate after each split and select the best #splits

d.) complexity pruning ➜ aim at best tree, but growing larger tree is penalized (regularization)

Different feature types

Numerical value

- easy, already discussed

Categorical value

- one-hot encoding (just like for linear regression)
- or you can split to two categorical groups (some implementation does not support it)
  - group1 : physicist, mathematician, engineer
  - group2 : truck driver, painter, bricklayer

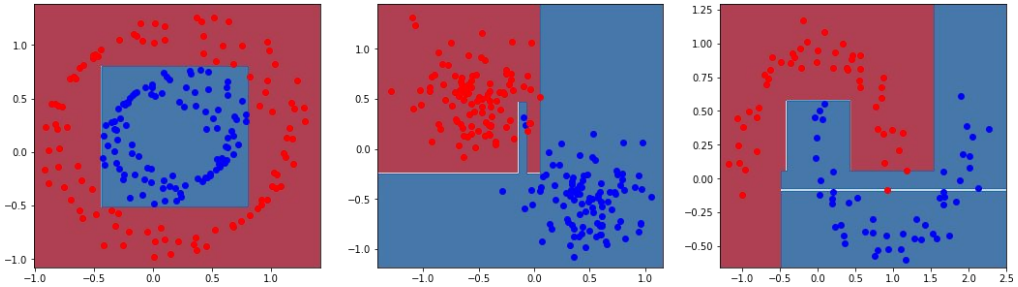
Missing value

- decision trees handle missing values pretty well
- is random missingness ➜ often good to impute with the mean value
- non-random missingness ➜ assign -9999 or some extreme value
  - ➜ the decision tree can split along this feature and handle the two groups, if it is useful
  - ➜ this would be a terrible idea for many models (eg for linear regression)

Decision tree - summary

- Splitting the input space into regions with constant predicted value
  - Division along the axes (single parameter)
  - Easy to interpret

- Training
  - Split data based on one variable to two parts
    - But which variable and which threshold/selection criteria?
      - To maximize ‘cleanness’ in each two new groups (considering all possible splits)
      - Gini impurity, entropy
- Prediction
  - Go through the tree and fall on a leaf.
    - If the leaf is ‘pure’ (only one class belonged to that during training)
      - that class is your prediction (for classification)
      - or that value is your prediction (for regression)
    - If the leaf is not pure ➔ majority voting (for classification) or average of samples (for regression)
      - Probability: frequency of class

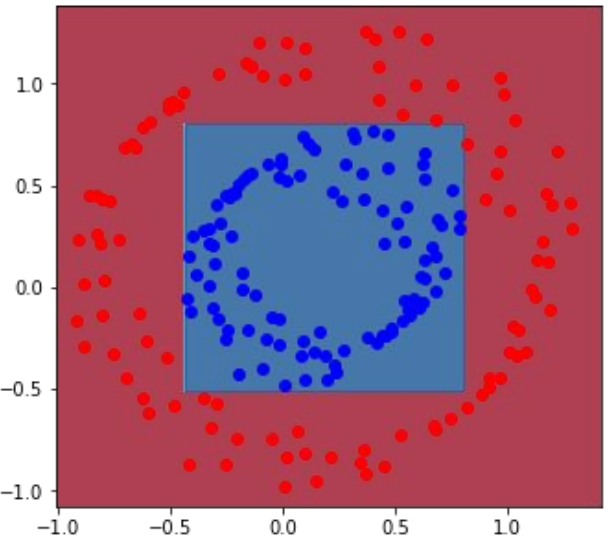


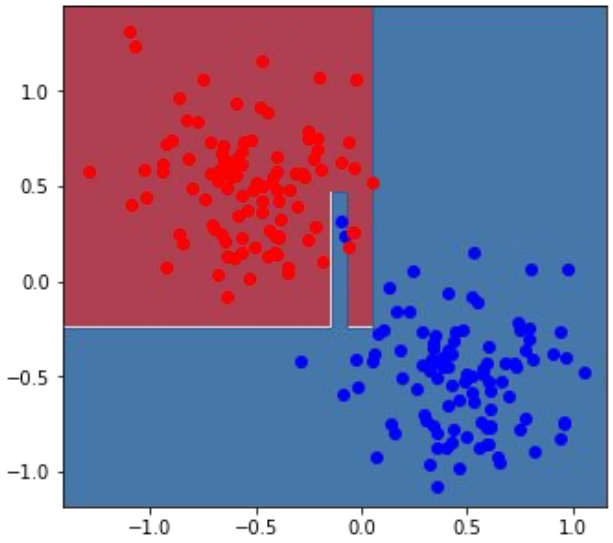
We can split only horizontally/vertically.
What if we need a diagonal/curved split?
Like middle and right.

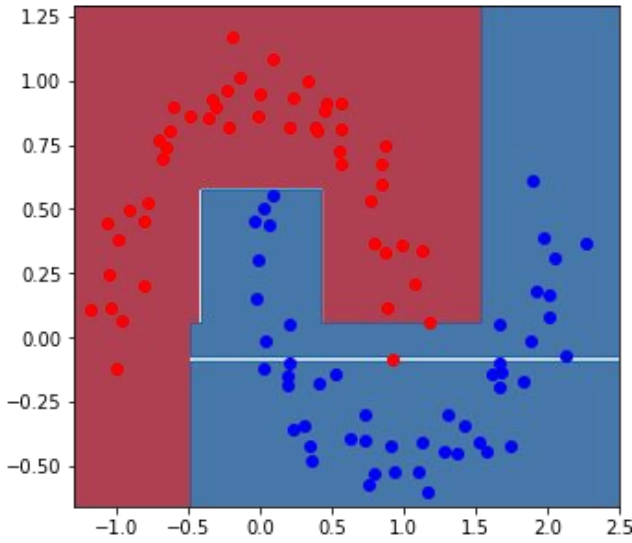
Random forest

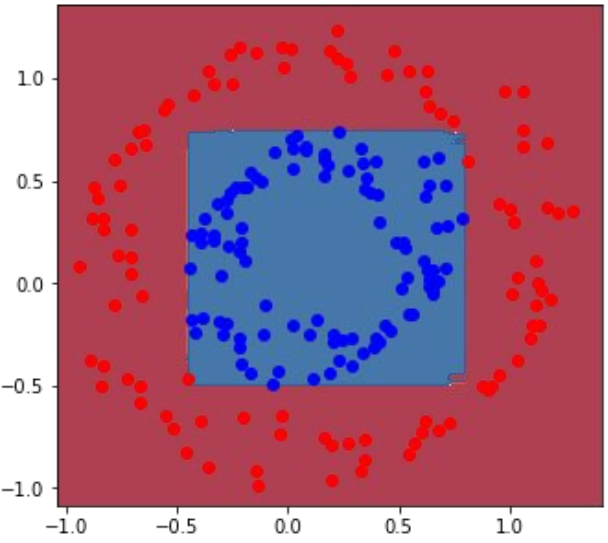
- A single decision tree is not accurate enough
- Train many of them!
  - N samples: select N samples out with replacement! (random subset)
  - At each split consider only a subset of features
- Result:
  - Many different decision tree
  - Average them! ➔ robust prediction
  - Idea: the errors cancel out each other if uncorrelated
- Interpretability
  - Feature importance
    i. Which feature increased the most the ‘cleanness’ ➔ each feature has its importance
    ii. Randomly shuffle a feature
      - Calculate how much the predictions got worse on the training set
      - The one that ruins the prediction the most is the most important feature

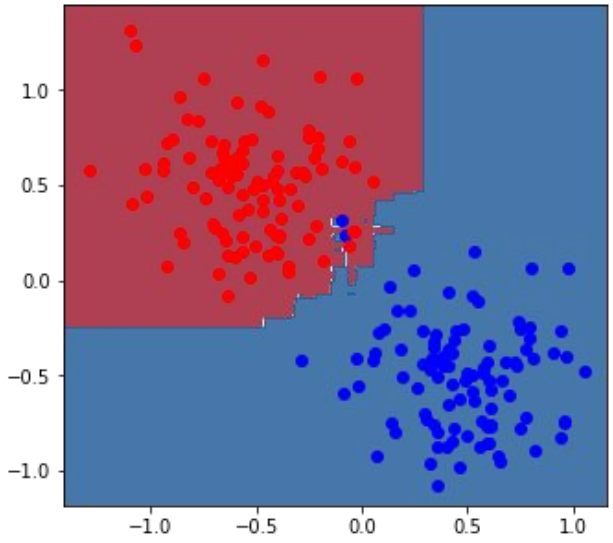
Decision tree vs Random forest

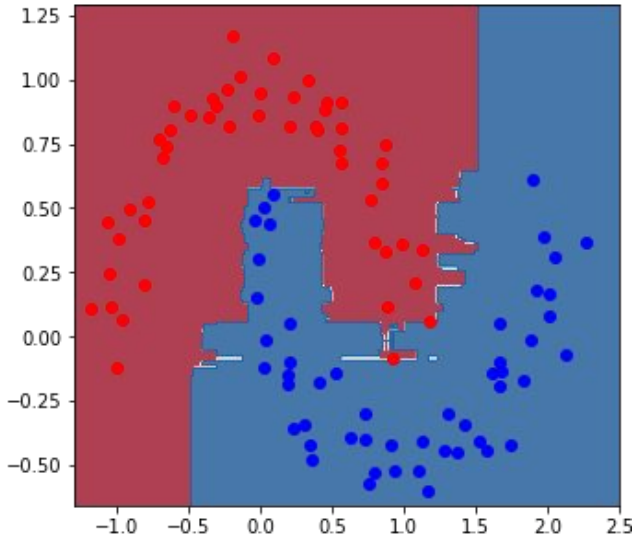








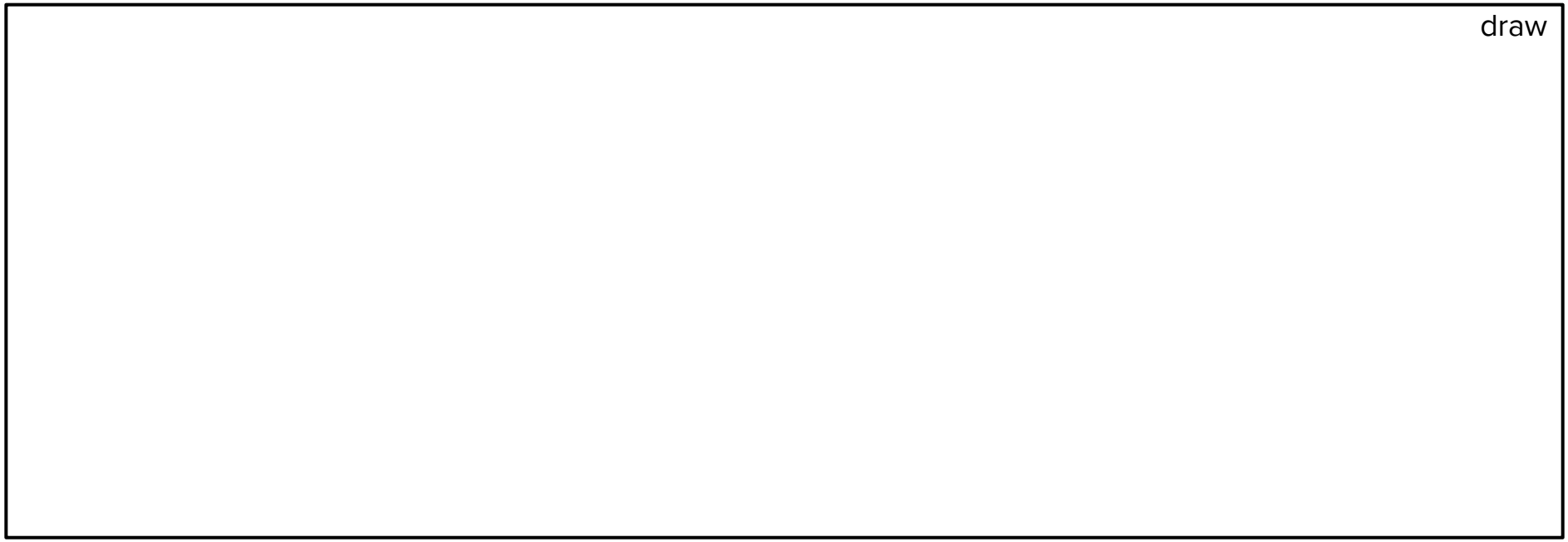




Meta algorithms

Ensemble

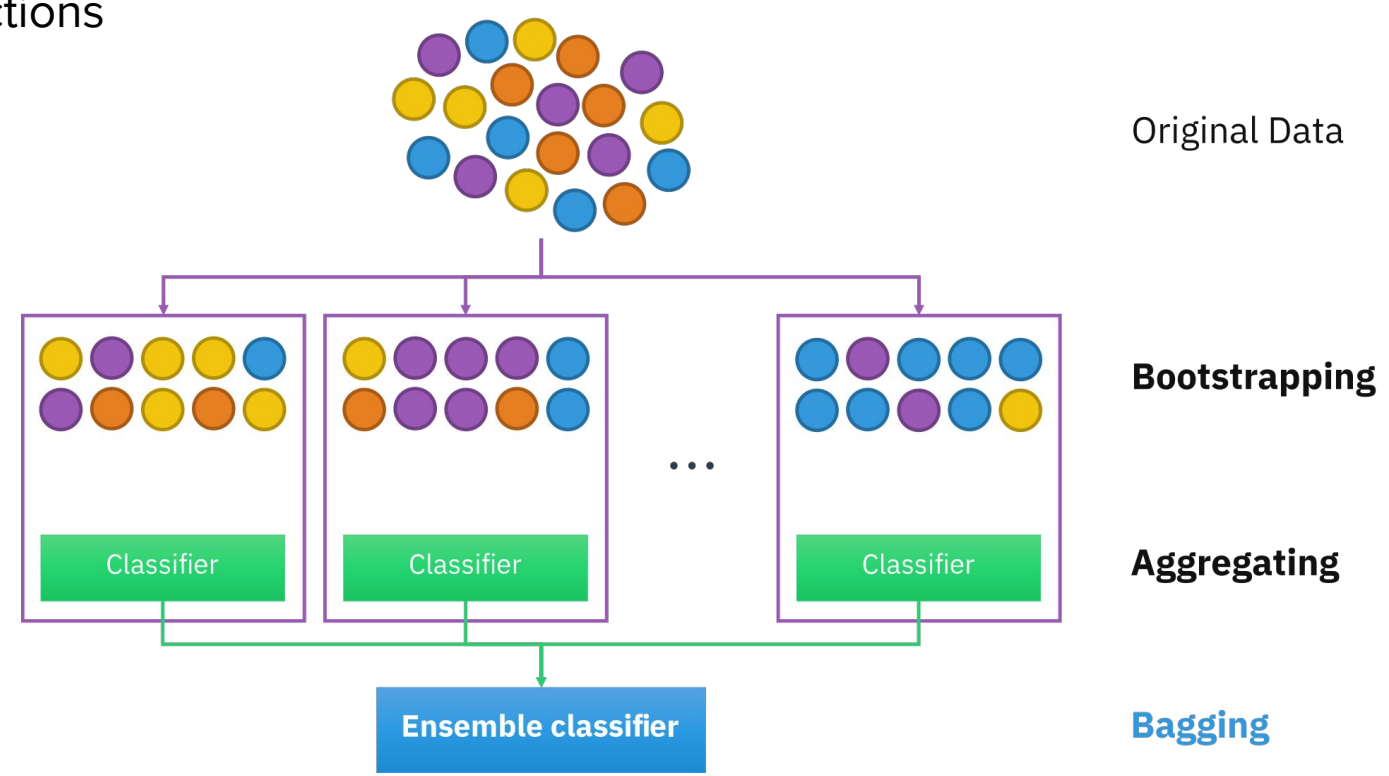
- train many (N) algorithms & generate predictions
- combine the predictions!
  - average
  - Feed the N predictions as an input for an other machine learning model (meta model)!
  - The meta model can learn which input area which model is the most reliable
    (like a boss)



Meta algorithms

Bagging (**b**ootstrap **agg**regat**ing**) ➔ just like for the random forest!

- Generate N new dataset from the original with random replacement sampling
- Train a model for each new dataset
- Average the predictions



By Sirakorn - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=85888768

Gradient boosting (for squared error loss)

Model #1

- Input: X
- Objective: y
- Output: y^1

Model #2

- Input: X
- Objective: $(y - y^1)$ = error of the first model (the residuals)
- Output: y^2

Model #3

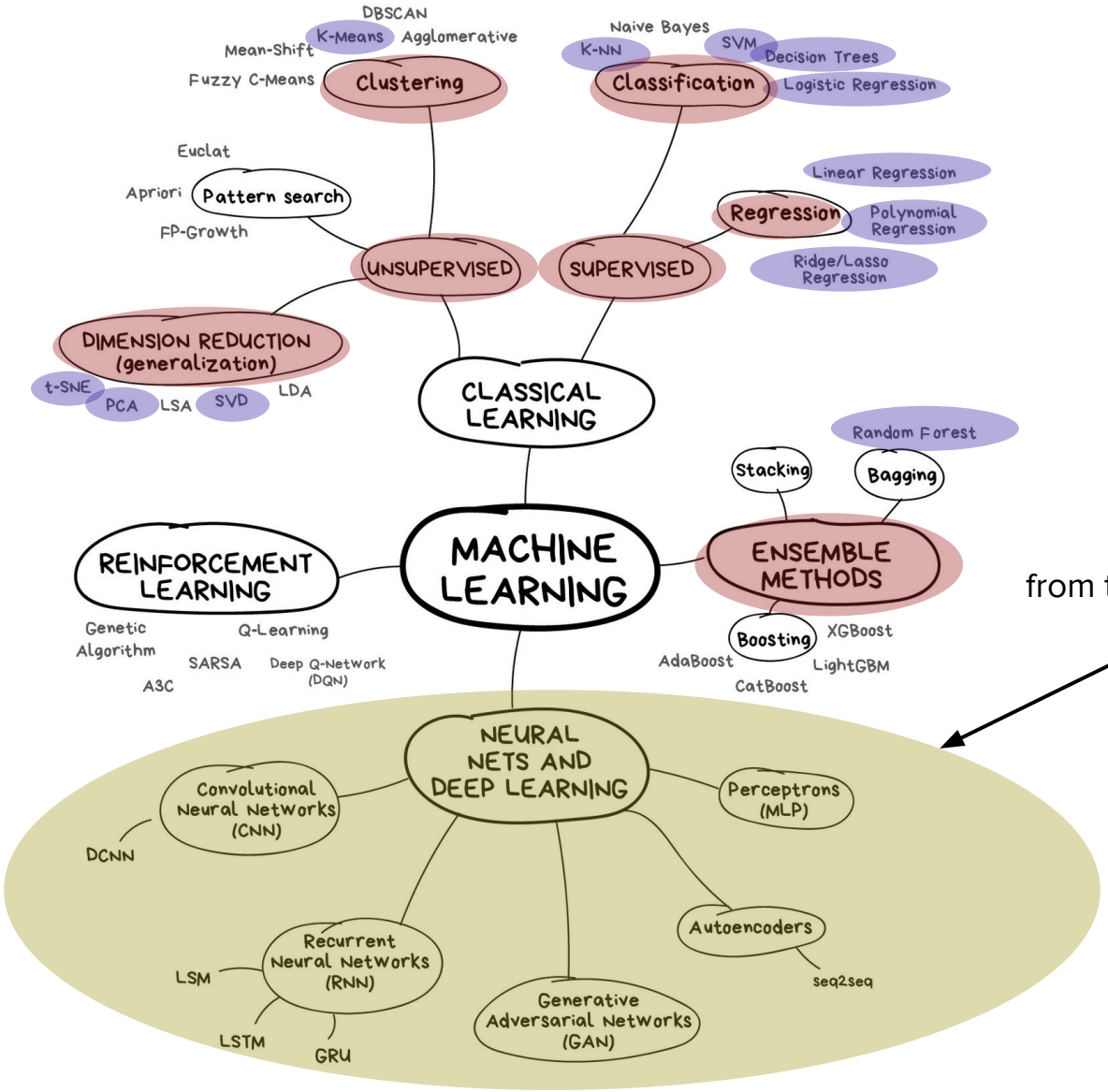
- Input: X
- Objective: $(y - y^1 - y^2)$ = error of the previous models
- Output: y^3

Final prediction: $y^1 + y^2 + y^3 + \ldots$

Many Kaggle competitions are won by:
- gradient boosting decision trees (XGBoost implementation)
- as en ensemble of 10s-100s of different models.

Each model tries to fix the previous models' errors!
XGBoost is a great library for gradient boosting
Most often used with decision trees!

Model zoo



from the next lecture on

http://valyrics.vas3k.com/blog/machine_learning/

Reference

ISLR chapter 8. https://www-bcf.usc.edu/~gareth/ISL/

Assignment 08.

1. Prepare dataset

- load the diabetes_data_upload.csv dataset
- search for missing values and if needed, handle them!
- encode the non numeric variables into numeric ones! For the binary features simply encode them as (0/1), do not create two separate columns for them!

2. Train & visualize decision tree classifier

- train a decision tree classifier using the sklearn API
- use its default parameters
- for training use all the data, this is only and exploratory task now
- visualize the decision tree (the `plot_tree` function in sklearn will be helpful)
- manually check for two cases if the returned Gini impurities are correct
- in a few sentences discuss the results

3. Random forest feature importance

- train a random forest classifier on all the data using the sklearn API
- use default values again, but fix the random_state to 42!
- plot the 10 most important features' importances
  - create a bar plot where the height of the bar is the feature importance
  - show the 10 features where the feature importance is the highest
  - `feature_importance` attribute is helpful

4. Evaluation

- generate prediction probabilities with a decision tree and with a random forest model
- use 5 fold cross validation for both time (so you should get 520 predictions)
- use default parameters for both models
- compare the two models with ROC curves
  - why does the decision tree's ROC curve looks different?

5. Tuning model

- using 80/20% train/test split generate predictions for a random forest model
- plot the AUC vs number of trees in the forest for both the traing and the test data
- do we experience overfitting if we use too many trees?