**Prof. Dr. Peer Küppers**

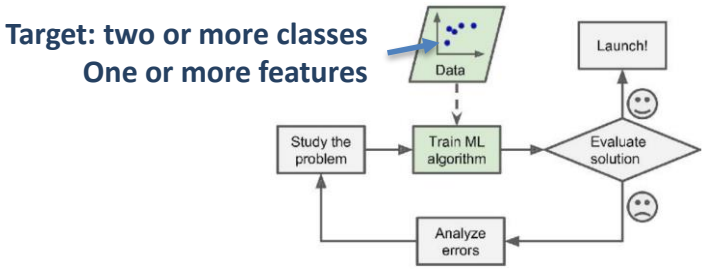
Summer School

# Supervised Learning: Classification

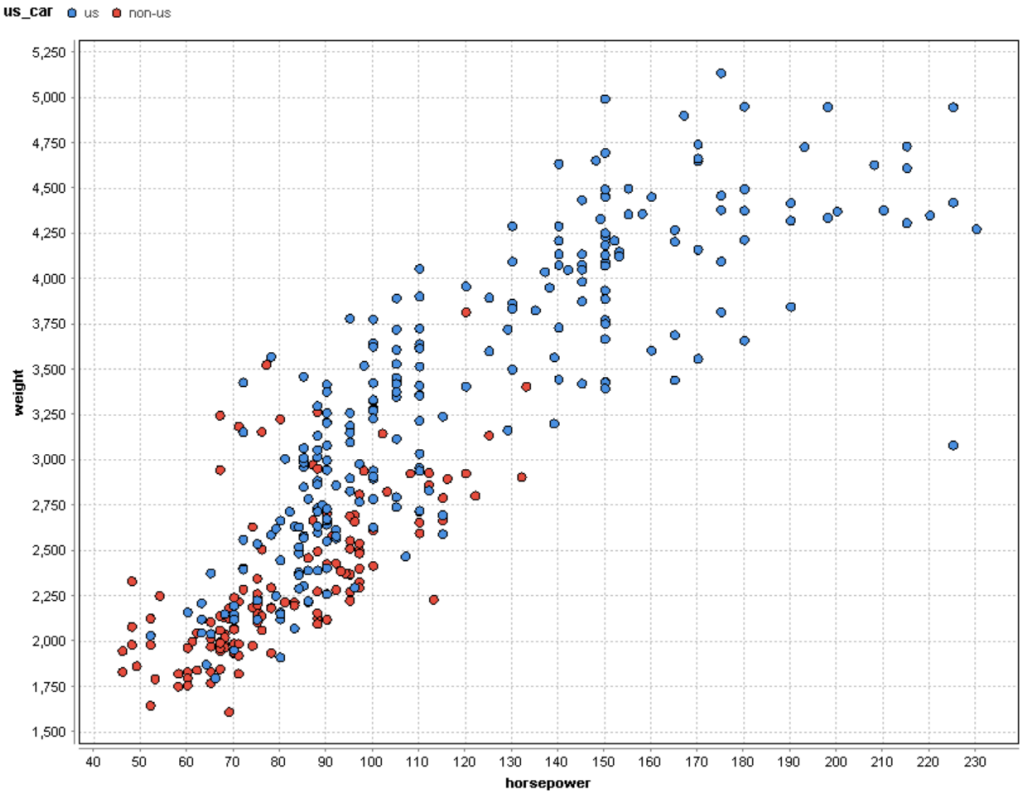
# Agenda – Part 3

- [x] **Classification Goals**
- [ ] **k-Nearest Neighbors (kNN)**
- [ ] **Decision Trees**
- [ ] **Ensemble Methods**
- [ ] **Evaluating Classification Models**
- [ ] **Summary & Outlook**

# Classification: Goal

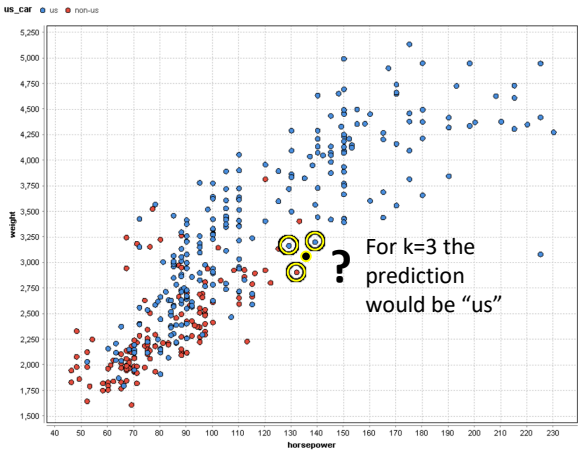



- Classification aims at predicting a nominal target feature based on one or multiple other (numerical) features
  - Use training data set of already labeled records to „learn“ which features influence in how far an example belongs to a specific class.
- Example: Predict origin (US vs. non-US) of a car based on its horsepowers and weight:



# kNN: Introduction

- k-Nearest-Neighbors (kNN) is a simple-yet-popular classification method that often serves as a baseline.
- kNN is a so-called lazy learning method, which reflects that it does not actually learn the parameters of a model, but always looks at the training data.
  - no cost for training a model, i.e., learning parameters
  - cost at runtime (i.e., when classifying a data point) depends on the amount of training data available
- To classify a previously unseen data point, kNN
  - identifies the k closest data points in the training data according to a suitable **distance measure**
  - predicts the nominal target feature (class) as the value that is most frequent among the k closest data points

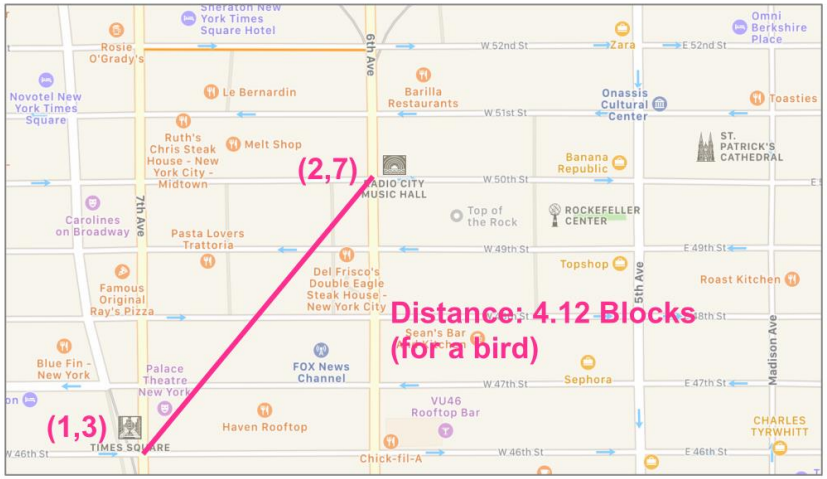


# kNN: Introduction

- Distance measures

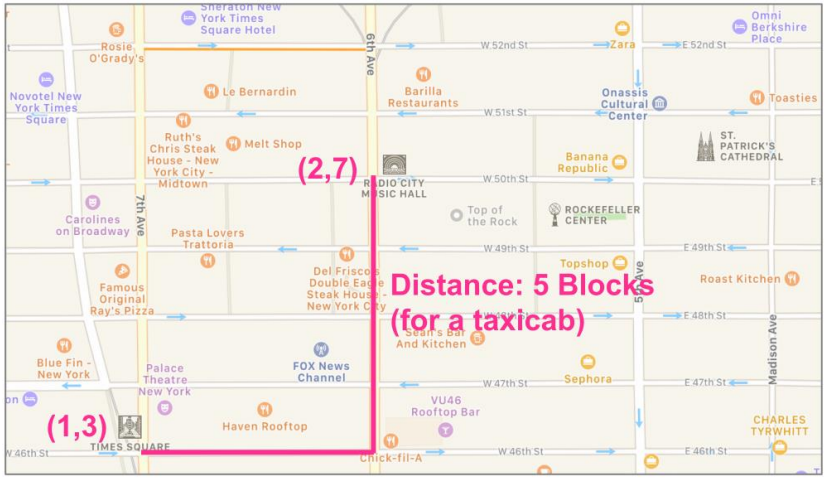
### Euclidian distance

$$d(x, x') = \sqrt{\sum_{i=1}^{m}(x_i - x'_i)^2}$$



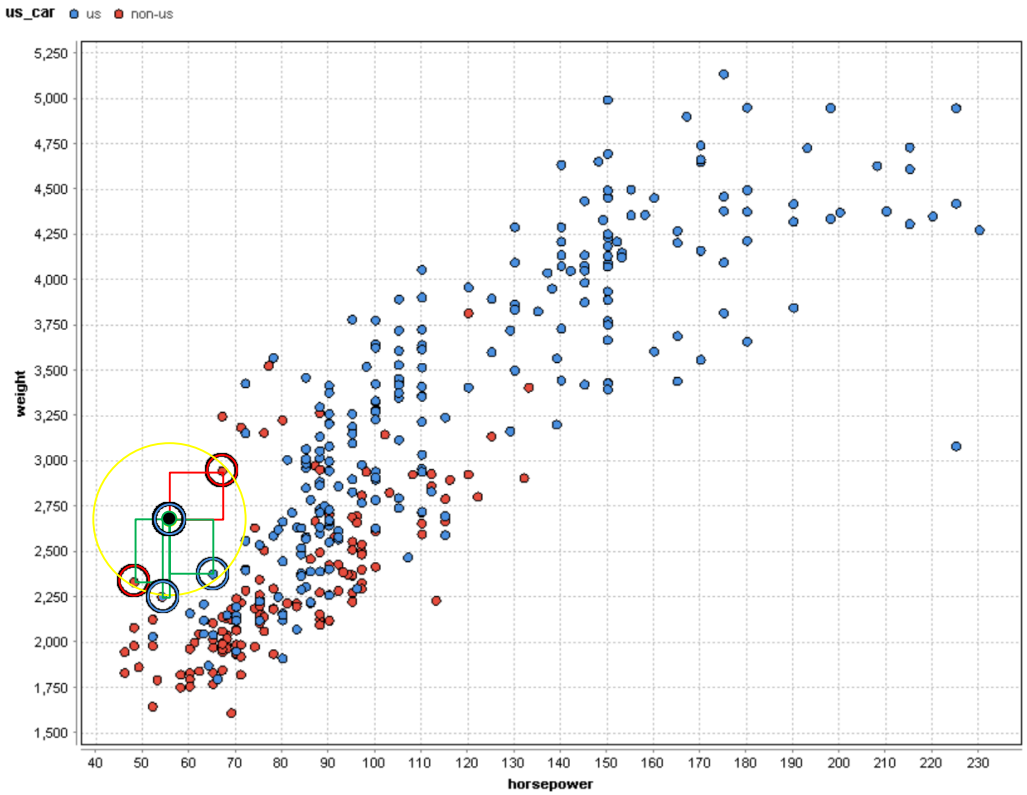

### Manhattan distance

$$d(x, x') = \sum_{i=1}^{m}|x_i - x'_i|$$



# Notes on Distance Measures

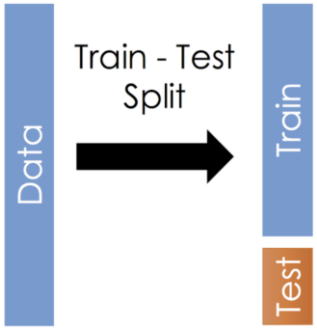
- Example, k=3
  - Euclidian distance
  - Manhattan distance



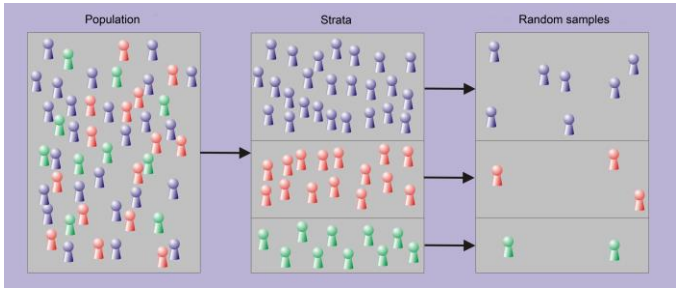

- Picking the right distance measure depends on the data and underlying assumptions
  - Euclidian distance prefers a “homogeneous” of the data point to be classified among all features (here: horsepower and weight).
  - Manhattan distance may lead to identifying close data points based on few features (possibly ignoring some completely).

# Train-Test-Split in Classification

- As always in machine learning, we have to hold back **test data** that can be used to evaluate our model (e.g. 20%).
  - Avoid overfitting!



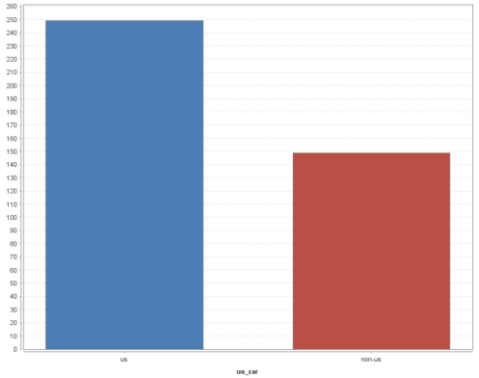

- There are different methods for train-test-splits.
  - Linear (as shown in the figure on the right)
  - Randomly choose x% as test set.
  - Stratified sampling
    - Divide the entire dataset (population) into homogeneous, non-overlapping subgroups (strata)
    - Select from these strata randomly, but keep relation between strata consistent within entire dataset and train / test sets.




*Stratified sampling is a good choice for classification since we have class labels as the target.*

Source: http://www.fernuni-hagen.de/statliteracy/chapter2/2.2(5).html

# Stratified Sampling



- In the mpg-dataset for example:
  - 249 us cars
  - 149 non-us cars

- When applying a 70%/30% stratified sampling train-test-split
  - Training set (278 samples):

```
non-us    0.374372
us        0.625628
```

  - Test set (120 samples):

```
non-us    0.375
us        0.625
```

- We will use this sampling method in all our labs!

Source: http://www.fernuni-hagen.de/statliteracy/chapter2/2.2(5).html

# Agenda – Part 3

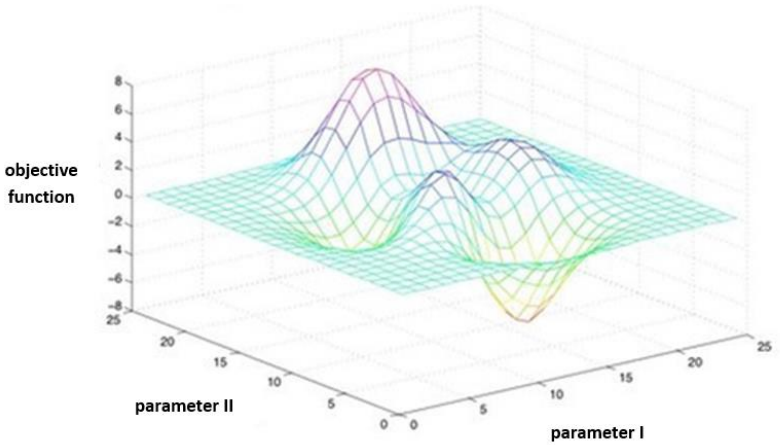
- [ ] **Classification Goals**
- [x] **k-Nearest Neighbors (kNN)**
- [ ] **Decision Trees**
- [ ] **Ensemble Methods**
- [ ] **Evaluating Classification Models**
- [ ] **Summary & Outlook**

# Lab

- Predicting the origin of a car using kNN

# kNN: Optimization

- Finding the right hyperparameter value(s) is the challenge.
- Popular method: **grid-search**
  - Vary the hyperparameter(s) of a machine learning model and evaluate each combination's outcome.
  - The objective function could for example be classification accuracy.




  - For kNN, we will only use one parameter: the number of neighbors (k)

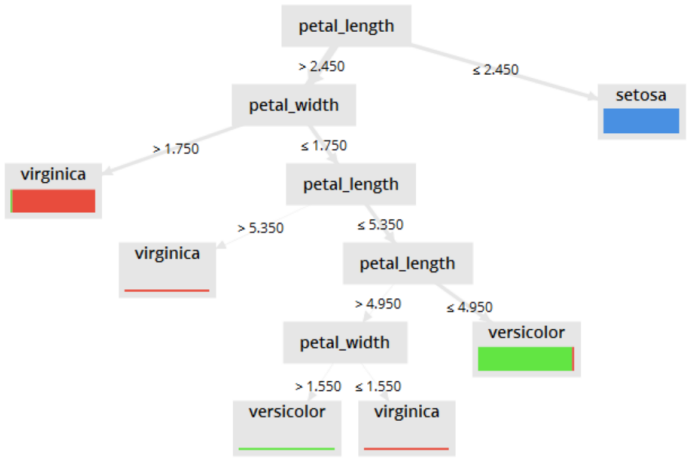
Source: https://www.oreilly.com/library/view/statistics-for-machine/9781788295758/e4dac91e-0078-4c8a-a44f-78a28abb649b.xhtml

# Agenda – Part 3

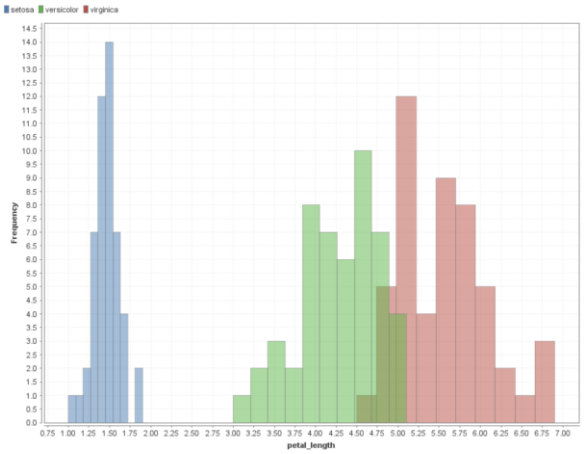
- [ ] **Classification Goals**
- [ ] **k-Nearest Neighbors (kNN)**
- [x] **Decision Trees**
- [ ] **Ensemble Methods**
- [ ] **Evaluating Classification Models**
- [ ] **Summary & Outlook**

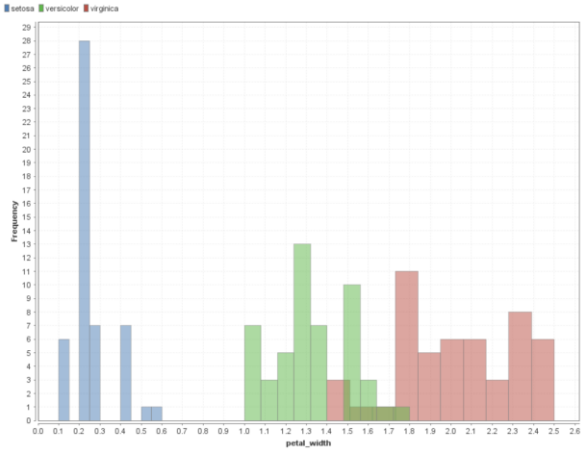
# Decision Trees: Introduction

- Decision trees are a family of classification methods that support an arbitrary number of target classes.
  - A decision tree provides a sequence of (binary) decisions which have to be made in order to decide which class a data point belongs to.
  - Good interpretability: we can see why a data point ended up in a specific class, and the classification can be performed by a human







# Decision Trees: Introduction

- Application examples
  - Churn Analysis
    - Detect customers who are likely to cancel a subscription, product or service
    - Example features: usage frequency, service hotline calls, sentiment analyses, competitors' activities, etc.
  - Fraud detection
    - Regular transactions vs. fraud transactions in credit cards
    - Challenge: much larger number of non-fraud transactions in training set
    - Example features: usage location, amount, frequency of usage, time of usage, etc.
  - Filter email spam
    - Spam emails vs. no-spam
    - Example features: words, sender, kind of formulation, etc. (challenge: unstructured data)
  - Predictive maintenance
    - Predict machine failures
    - Example features: sensor data (temperature, vibration, optical, etc.)

# Decision Trees: Introduction

Training set

| species | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| virginica | 6.200 | 2.800 | 4.800 | 1.800 |
| virginica | 6.300 | 2.900 | 5.600 | 1.800 |
| setosa | 5.100 | 3.500 | 1.400 | 0.300 |
| setosa | 5.200 | 3.500 | 1.500 | 0.200 |
| versicolor | 5.900 | 3 | 4.200 | 1.500 |
| versicolor | 5.700 | 3 | 4.200 | 1.200 |
| versicolor | 5.500 | 2.600 | 4.400 | 1.200 |
| virginica | 6.400 | 2.800 | 5.600 | 2.200 |
| versicolor | 5.100 | 2.500 | 3 | 1.100 |
| virginica | 6.700 | 3.300 | 5.700 | 2.500 |
| virginica | 7.700 | 3.800 | 6.700 | 2.200 |
| setosa | 4.800 | 3.400 | 1.600 | 0.200 |
| versicolor | 7 | 3.200 | 4.700 | 1.400 |
| versicolor | 6.300 | 2.500 | 4.900 | 1.500 |

target features

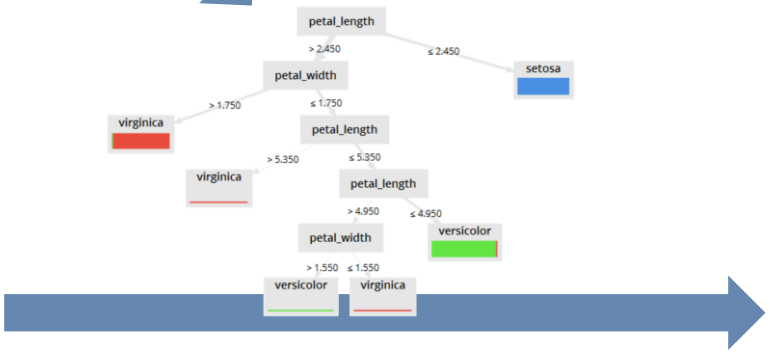
Test set / new samples

| species | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| ? | 4.600 | 3.100 | 1.500 | 0.200 |
| ? | 6.400 | 3.200 | 5.300 | 2.300 |
| ? | 5.500 | 2.400 | 3.700 | 1 |
| ? | 7.900 | 3.800 | 6.400 | 2 |
| ? | 6.400 | 3.100 | 5.500 | 1.800 |

Questions:

- Which features will be used as nodes?
- In which sequence will they be used?
- Which split values should be used?

Train model

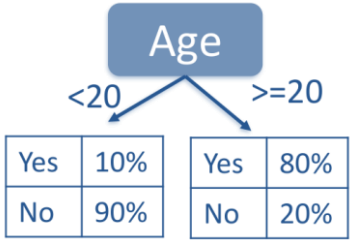



| species |
|---|
| setosa |
| virginica |
| versicolor |
| virginica |
| virginica |

Apply model

# Generation of Decision Trees: Impurity

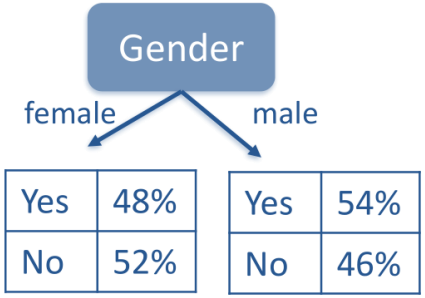
- Selection of nodes: example gym-membership
  - Who will sign-up for a gym membership after a sample training session?
    - Target: membership (Yes/No)
    - Features: Age and Gender
  - The historical data show these distributions
    - Age feature




Which feature is better suited for a node-split?

*Homogeneous, low impurity*
→ well-suited

    - Gender feature

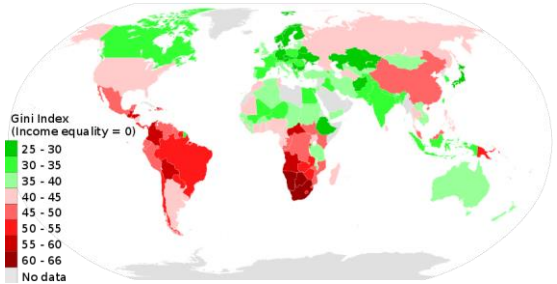



*Inhomogeneous, high impurity*

# Generation of Decision Trees: Impurity

- There are several methods to calculate the impurity. We will take a deeper look at the **Gini impurity**

- Original context: measure income distributions
  - **Gini index** that shows the range between the extrema
    - one person earns everything
    - every person earns the same amount




- Gini impurity
  - Similar concept, calculate the impurity of nodes in decision trees

Source: https://en.wikipedia.org/wiki/List_of_countries_by_income_equality

# Generation of Decision Trees: Impurity

- Gini impurity

$$Gini = 1 - \sum_{i=1}^{k} p_i^2$$

$p_i$ : fraction of items labeled with class i in the training data

$k$ : number of target classes in the training data

- Example: spam detection

| ID | Subject contains "lottery" | Subject is all CAPITALS | Mail contains link | Spam |
|---|---|---|---|---|
| 1 | Yes | Yes | No | Yes |
| 2 | No | No | No | No |
| 3 | No | Yes | Yes | Yes |
| 4 | Yes | No | No | Yes |
| 5 | No | No | No | No |
| 6 | No | Yes | No | No |
| 7 | Yes | No | Yes | Yes |
| 8 | No | No | No | No |
| 9 | No | Yes | No | Yes |
| 10 | No | No | Yes | No |

No of samples: 10
Samples in class "Yes" (i=1): 5
Samples in class "No" (i=2): 5

$$Gini = 1 - \sum_{i=1}^{2} p_i^2 = 1 - \left(\frac{5}{10}\right)^2 - \left(\frac{5}{10}\right)^2$$
$$= 1 - 0.25 - 0.25$$
$$= 0.5$$

This is the impurity of the data regarding the target "Spam".
A Gini impurity of 0.5 is the maximum possible, i.e. we have a very inhomogeneous set currently.

# Generation of Decision Trees: Impurity


- Next, we want to determine the split quality of all features based on the Gini impurity. We will start with the feature “Subject contains ‘lottery’ ”:
  - **Step 1**: calculate the Gini impurity for all subsets of the feature’s values:

| ID | Subject contains “lottery” | Subject is all CAPITALS | Mail contains link | Spam |
|---|---|---|---|---|
| 1 | Yes | Yes | No | Yes |
| 2 | No | No | No | No |
| 3 | No | Yes | Yes | Yes |
| 4 | Yes | No | No | Yes |
| 5 | No | No | No | No |
| 6 | No | Yes | No | No |
| 7 | Yes | No | Yes | Yes |
| 8 | No | No | No | No |
| 9 | No | Yes | No | Yes |
| 10 | No | No | Yes | No |

Two subsets:

- S1: samples that contain “lottery” (=Yes)
- S2: samples that do not contain “lottery” (=No)

| | S1 (=Yes) | S2 (=No) |
|---|---|---|
| #Samples | 3 | 7 |
| # class “Spam” | 3 | 2 |
| # class “No Spam” | 0 | 5 |
| Gini impurity | $1-\left(\frac{3}{3}\right)^2-\left(\frac{0}{3}\right)^2=0$ | $1-\left(\frac{2}{7}\right)^2-\left(\frac{5}{7}\right)^2\approx 0{,}41$ |

$$Gini = 1 - \sum_{i=1}^{k} p_i^2$$

# Generation of Decision Trees: SplitQuality


- **Step 2**: calculate the Gini SplitQuality of this feature working as a node by weighting both subsets' gini impurity values:

$$SplitQuality = \sum_{i=1}^{k} \frac{n_i}{n} Gini_i$$

| ID | Subject contains "lottery" | Subject is all CAPITALS | Mail contains link | Spam |
|---|---|---|---|---|
| 1 | Yes | Yes | No | Yes |
| 2 | No | No | No | No |
| 3 | No | Yes | Yes | Yes |
| 4 | Yes | No | No | Yes |
| 5 | No | No | No | No |
| 6 | No | Yes | No | No |
| 7 | Yes | No | Yes | Yes |
| 8 | No | No | No | No |
| 9 | No | Yes | No | Yes |
| 10 | No | No | Yes | No |

$k$ : number of target classes at current node

$n$ : total number of samples at current node

$n_i$ : number of samples at child-node *i*

| | S1 (=Yes) | S2 (=No) |
|---|---|---|
| #Samples | 3 | 7 |
| # class "Spam" | 3 | 2 |
| # class "No Spam" | 0 | 5 |
| Gini impurity | $1-\left(\frac{3}{3}\right)^2-\left(\frac{0}{3}\right)^2=0$ | $1-\left(\frac{2}{7}\right)^2-\left(\frac{5}{7}\right)^2\approx 0.41$ |
| SplitQuality of Node "Lottery" | $SplitQuality=\frac{3}{10}\cdot 0+\frac{7}{10}\cdot 0.41=\mathbf{0.29}$ | |

# Generation of Decision Trees: Selection

| ID | Subject contains "lottery" | Subject is all CAPITALS | Mail contains link | Spam |
|---|---|---|---|---|
| 1 | Yes | Yes | No | Yes |
| 2 | No | No | No | No |
| 3 | No | Yes | Yes | Yes |
| 4 | Yes | No | No | Yes |
| 5 | No | No | No | No |
| 6 | No | Yes | No | No |
| 7 | Yes | No | Yes | Yes |
| 8 | No | No | No | No |
| 9 | No | Yes | No | Yes |
| 10 | No | No | Yes | No |

- We need to calculate the SplitQuality for all features


| | S1 (=Yes) | S2 (=No) |
|---|---|---|
| Gini impurity | $1-\left(\frac{3}{3}\right)^2-\left(\frac{0}{3}\right)^2=0$ | $1-\left(\frac{2}{7}\right)^2-\left(\frac{5}{7}\right)^2\approx 0.41$ |
| SplitQuality | $SplitQuality=\frac{3}{10}\cdot 0+\frac{7}{10}\cdot 0.41=\mathbf{0.29}$ | |

Select the feature with the lowest value as split node!


| | S1 (=Yes) | S2 (=No) |
|---|---|---|
| #Samples | 4 | 6 |
| # class "Spam" | 3 | 2 |
| # class "No Spam" | 1 | 4 |
| Gini impurity | $1-\left(\frac{3}{4}\right)^2-\left(\frac{1}{4}\right)^2\approx 0.38$ | $1-\left(\frac{2}{6}\right)^2-\left(\frac{4}{6}\right)^2\approx 0.44$ |
| SplitQuality | $SplitQuality=\frac{4}{10}\cdot 0.38+\frac{6}{10}\cdot 0.44=\mathbf{0.42}$ | |


| | S1 (=Yes) | S2 (=No) |
|---|---|---|
| #Samples | 3 | 7 |
| # class "Spam" | 2 | 3 |
| # class "No Spam" | 1 | 4 |
| Gini impurity | $1-\left(\frac{2}{3}\right)^2-\left(\frac{1}{3}\right)^2=0.44$ | $1-\left(\frac{3}{7}\right)^2-\left(\frac{4}{7}\right)^2\approx 0.49$ |
| SplitQuality | $SplitQuality=\frac{3}{10}\cdot 0.44+\frac{7}{10}\cdot 0.49=\mathbf{0.48}$ | |

# Generation of Decision Trees: Growing the Tree


We identified the top-most feature to be used as split-node.

The class “Lottery=Yes” has an **impurity of 0**
→ we reached the leaf in this branch of the tree.

The class “Lottery=No” has an **impurity > 0**
→ we need to grow the tree further and find out which of the two remaining features has the better SplitQuality on the remaining data

| ID | Subject contains “lottery” | Subject is all CAPITALS | Mail contains link | Spam |
|---|---|---|---|---|
| ~~1~~ | ~~Yes~~ | ~~Yes~~ | ~~No~~ | ~~Yes~~ |
| 2 | No | No | No | No |
| 3 | No | Yes | Yes | Yes |
| ~~4~~ | ~~Yes~~ | ~~No~~ | ~~No~~ | ~~Yes~~ |
| 5 | No | No | No | No |
| 6 | No | Yes | No | No |
| ~~7~~ | ~~Yes~~ | ~~No~~ | ~~Yes~~ | ~~Yes~~ |
| 8 | No | No | No | No |
| 9 | No | Yes | No | Yes |
| 10 | No | No | Yes | No |

## Remaining dataset in the branch:

| ID | Subject contains “lottery” | Subject is all CAPITALS | Mail contains link | Spam |
|---|---|---|---|---|
| 2 | No | No | No | No |
| 3 | No | Yes | Yes | Yes |
| 5 | No | No | No | No |
| 6 | No | Yes | No | No |
| 8 | No | No | No | No |
| 9 | No | Yes | No | Yes |
| 10 | No | No | Yes | No |

# Growing the Tree

| ID | Subject contains "lottery" | Subject is all CAPITALS | Mail contains link | Spam |
|---|---|---|---|---|
| 2 | No | No | No | No |
| 3 | No | Yes | Yes | Yes |
| 5 | No | No | No | No |
| 6 | No | Yes | No | No |
| 8 | No | No | No | No |
| 9 | No | Yes | No | Yes |
| 10 | No | No | Yes | No |


Spam

*We need to grow the tree in this branch*


$$Gini = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 \approx 0.44$$

$$Gini = 1 - \left(\frac{0}{4}\right)^2 - \left(\frac{4}{4}\right)^2 = 0$$

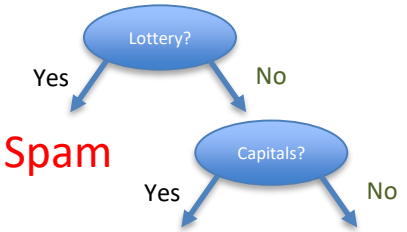
$$SplitQuality = \frac{3}{7} \cdot 0.44 + \frac{4}{7} \cdot 0 = \mathbf{0.19}$$

**vs.**


$$Gini = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$$

$$Gini = 1 - \left(\frac{1}{5}\right)^2 - \left(\frac{4}{5}\right)^2 = 0.32$$

$$SplitQuality = \frac{2}{7} \cdot 0.5 + \frac{5}{7} \cdot 0.32 \approx \mathbf{0.37}$$




Spam

*We need to grow the tree in this branch*

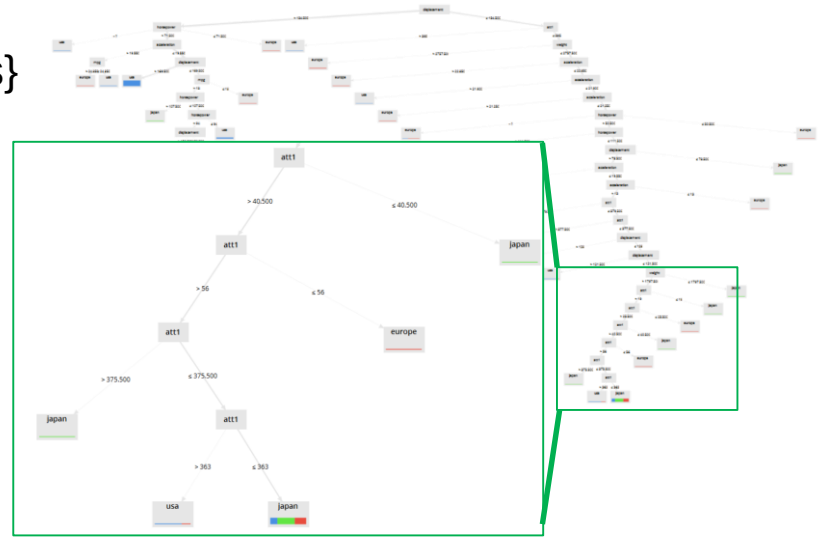
No Spam

The class "Capitals=No" has an **impurity of 0**
→ we reached the leaf in this branch of the tree.

The class "Capitals=Yes" has an **impurity > 0**
→ we need to grow the tree further

# Generation of Decision Trees: ID3 algorithm

- The ID3 algorithm defines the split rules and growth of the tree:
- Split (node, {samples})
  1. A ← the best attribute for splitting {samples}
  2. Decision attribute for this node ← A
  3. For each value of A, create new child node
  4. Split training {samples} to child nodes
  5. For each child node / subset:
     if subset is pure: STOP
     else: Split (child_node, {subset})




The tree even memorizes the values of the ID column “att1”

- Problem
  - The algorithm yields a tree that contains only leafs that are “pure” (see step 5).
  - The tree growths a lot.
  - It is very prone to overfitting, i.e. memorizing the training data instead of finding generalizable relationships between features and target.
    - Model performs well on training data, but poorly on test / new data.

# Generation of Decision Trees: Pruning

- In order to avoid overfitting, the growth of decision trees is usually restricted by a method called **pruning**.

- Pre-pruning
  - Is done during growth of the tree
  - Stop when, e.g.
    - a purity threshold cannot be reached,
    - a maximum depth is reached, or
    - a minimum number of samples in subsets is not achieved.

- Post-pruning
  - Grow tree without restrictions
  - Trim afterwards branches that do not effectively change the classification error rates
  - Sometimes a better option since small but potentially significant relationships between features and target are not missed (as with pre-pruning)
  - Requires additional computations

# Agenda – Part 3

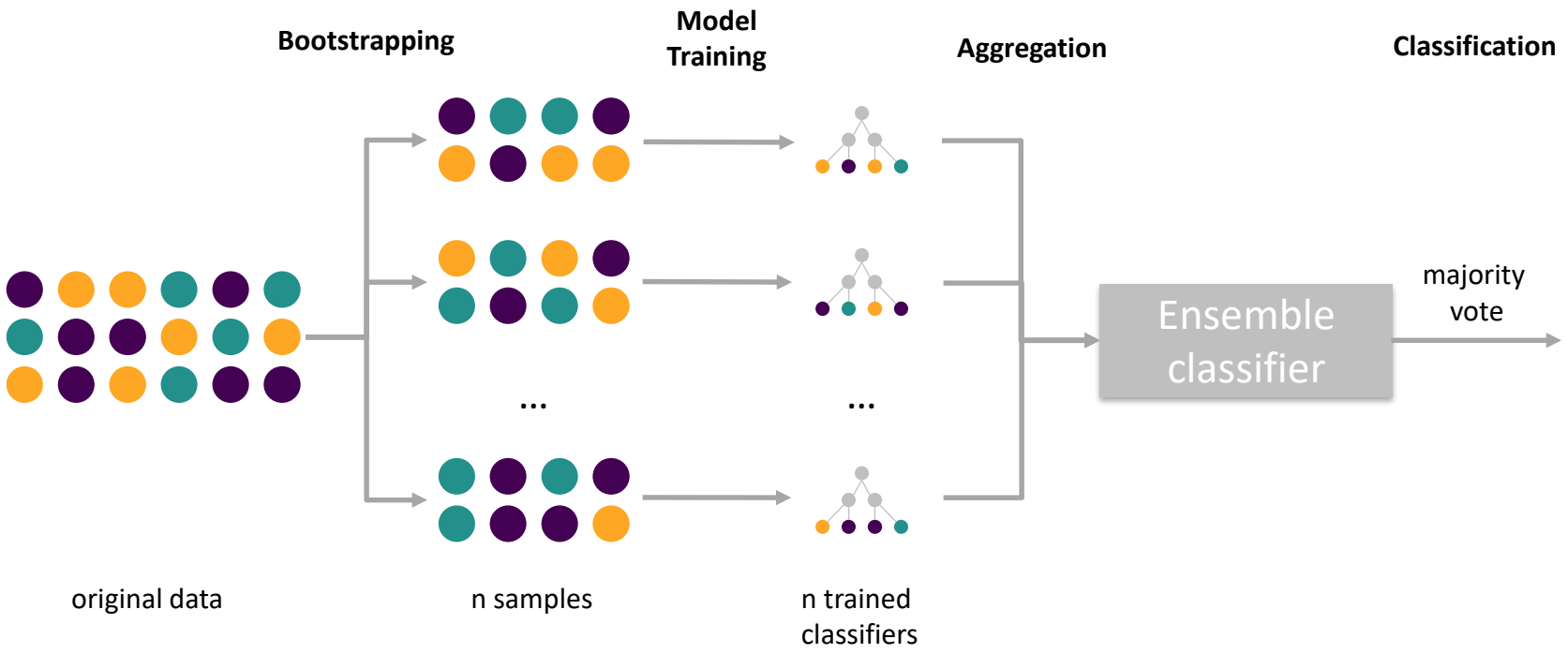
- [ ] Classification Goals
- [ ] k-Nearest Neighbors (kNN)
- [ ] Decision Trees
- [x] Ensemble Methods
- [ ] Evaluating Classification Models
- [ ] Summary & Outlook

# Ensemble Methods: Introduction

- A single decision tree (or other simple classifier) does not perform well.
  - Especially: high variance and prone to overfitting
  - However, it is very fast
- Idea
  - Train multiple simple classifiers (e.g. trees) to improve the performance
    - → Ensembles of Classifiers
  - Combine the classification results from different classifiers to produce the final output, e.g. by
    - Weighted average, weighted majority vote, logistic regression
  - With insufficient data (almost always), the single classifier can find many equally good solutions. → combination of multiple classifiers → reduce risks
  - Reduce bias or variance
    - Minimize variance, e.g. with Bagging or Random Forests
    - Minimize bias (increase predictive force), e.g. with Boosting
    - Combination, e.g. with Stacking

# Reduce Variance with Ensemble Methods

- **Example: Bagging**





Cf. Zhou (2012) "Ensemble Methods. Foundations and Algorithms"

# Ensemble Methods: Overview

- Aggregating a group of classifiers (“base classifiers”) as an ensemble committee and making the prediction by consensus.
- Weak learner ensembles

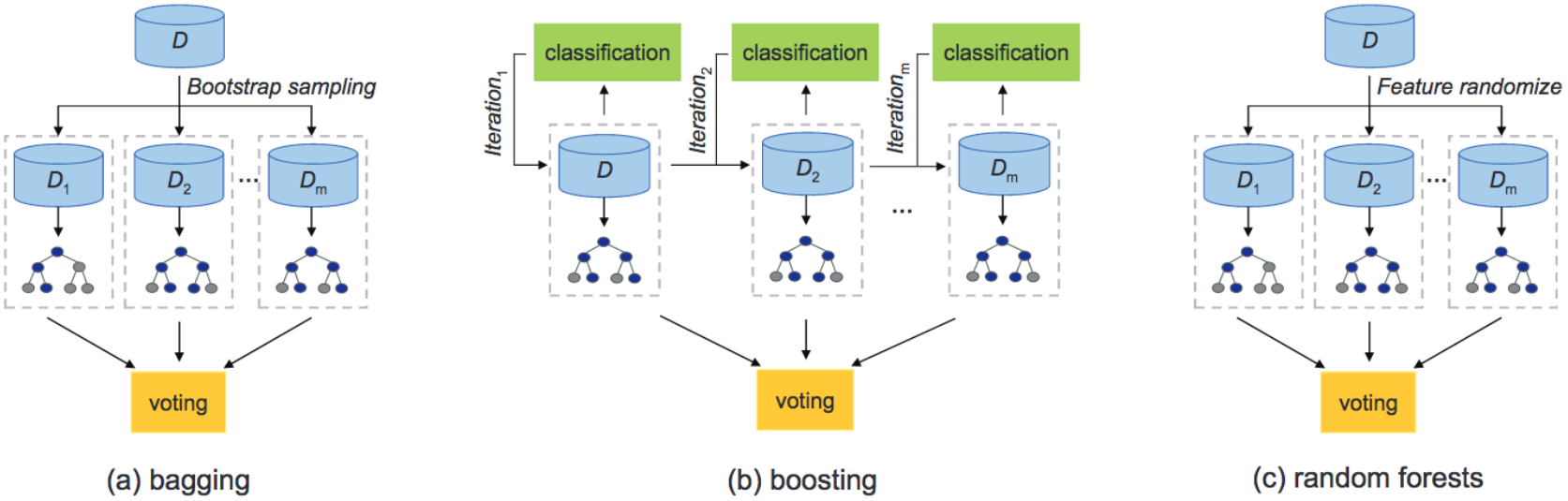
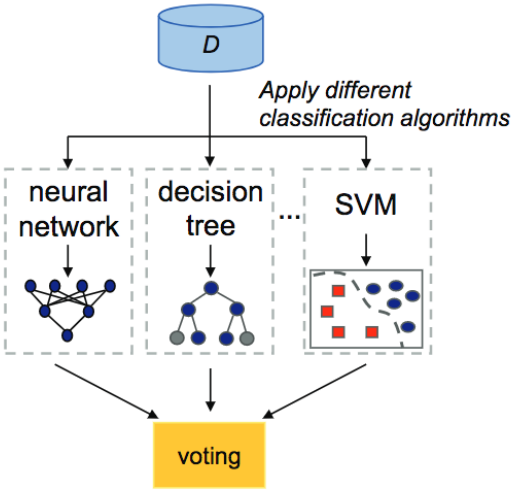



Fig. 1: Schematic illustration of the three popular ensemble methods.

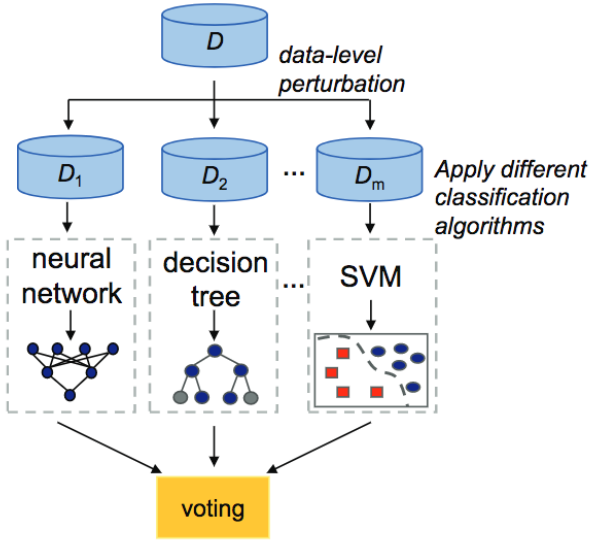
Source: Current Bioinformatics, 5, (4):296-308, 2010

# Ensemble Methods: Overview

- Strong learner ensembles ("stacking")




(a) without data perturbation



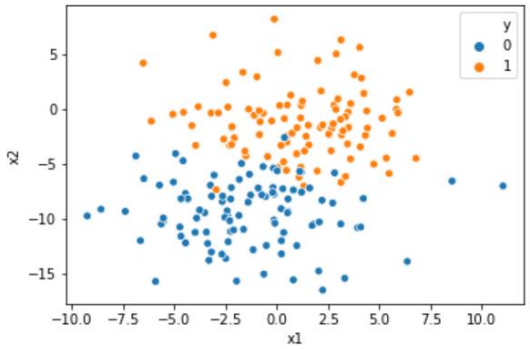

(b) with data perturbation

Fig. 8: Schematic illustration of the ensemble using different classification algorithms. (a) classification algorithms are trained using the same training set. (b) classification algorithms are trained using different perturbations of the training set.

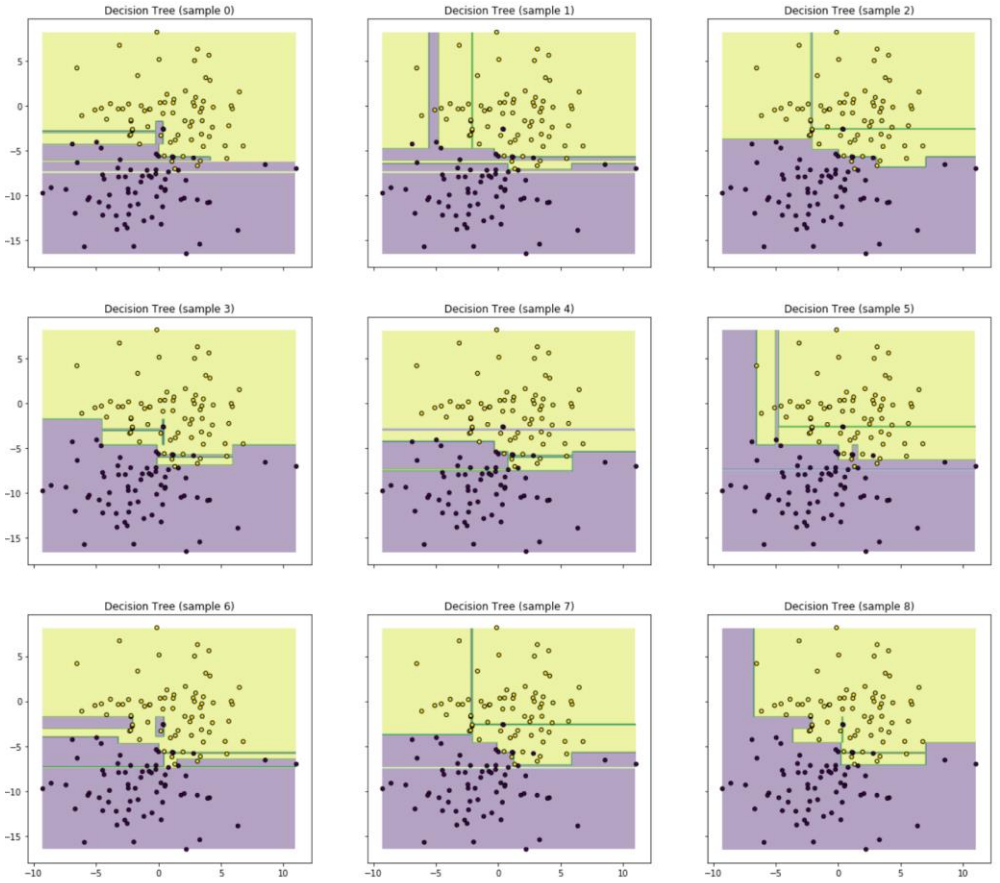
Source: Current Bioinformatics, 5, (4):296-308, 2010

# Bagging: Reduce Variance

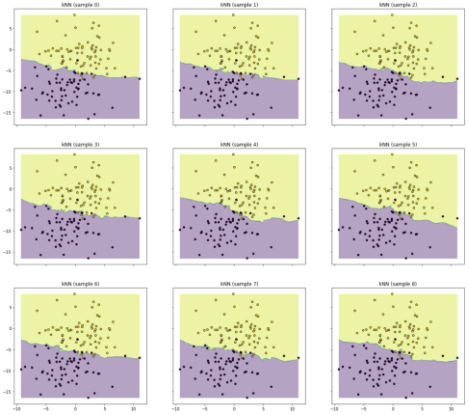
**Original dataset:**




- Unstable learner: example decision tree
  - For 9 different samples (70% randomly chosen) from the original dataset, we get these trained decision trees:
    - *max_depth = 10*
  - → Relatively high variance
    - → The model changes significantly with relatively small changes in the input dataset.

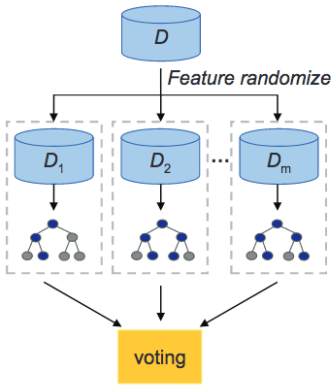



**kNN is more stable:**

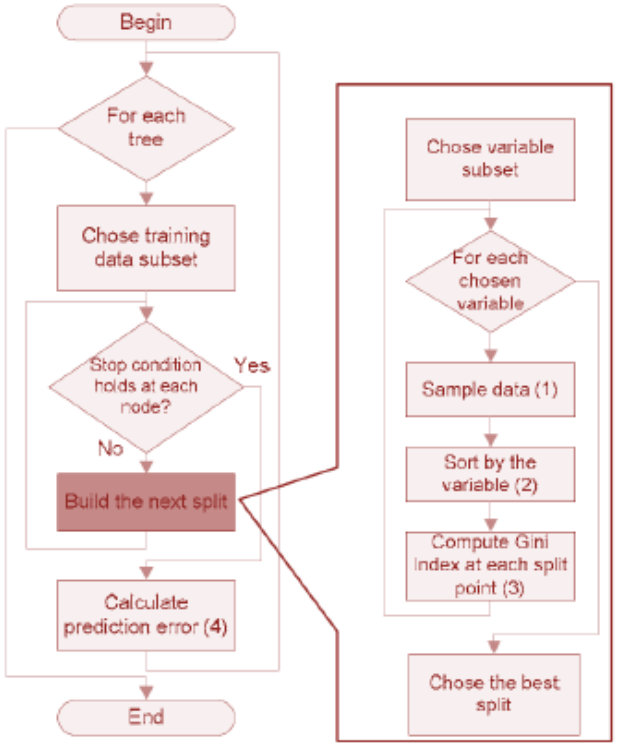


# Random Forests: Introduction

- **Random forest** is an ensemble classifier that consists of many decision trees and outputs the class that is voted by the majority of the individual trees (=bagging so far).
- The method combines the "bagging" idea with a **random selection of features**.
- Proceeding:




(c) random forests



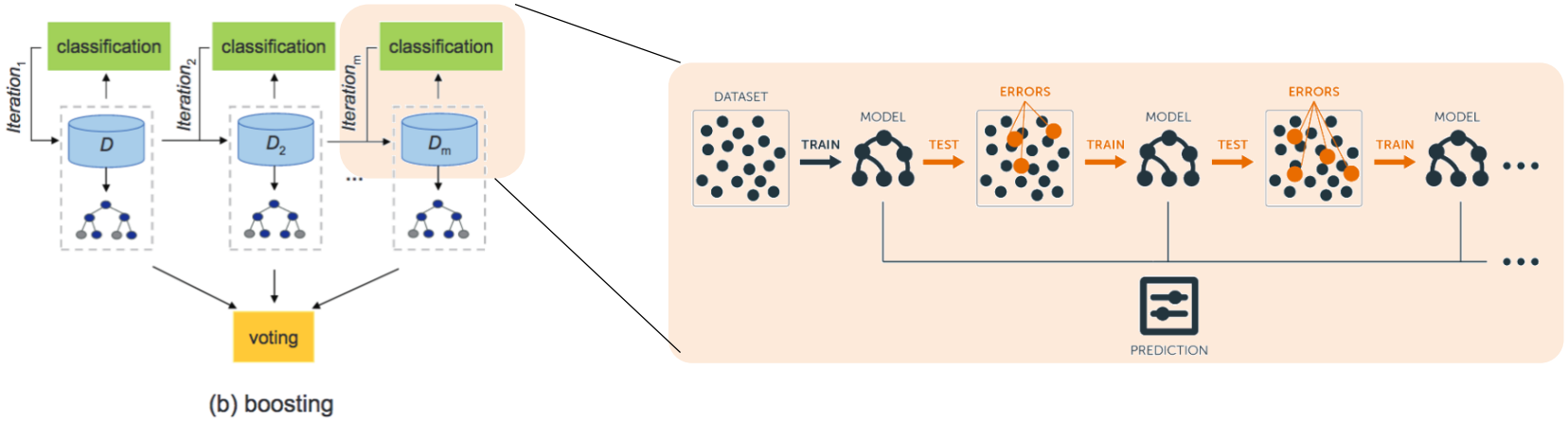

Source: http://home.etf.rs/~vm/os/dmsw

# Random Forests: Evaluation

- Advantages
  - One of the most accurate learning algorithms available.
    - For many data sets, produces a highly accurate classifier
  - Runs efficiently on large datasets
    - Can handle thousands of input variables without variable deletion.
    - Full parallelization possible!
  - Gives estimates of what variables are important in the classification.
  - Can handle missing data, no feature scaling required.
  - Robust to outliers
- Challenges
  - Random forests have been observed to overfit in case of noisy data.
  - Complexity (but parallelizable!)

# Boosting: Introduction

- Construct a sequence of weak classifiers, and combine them into a strong classifier by a weighted majority vote.



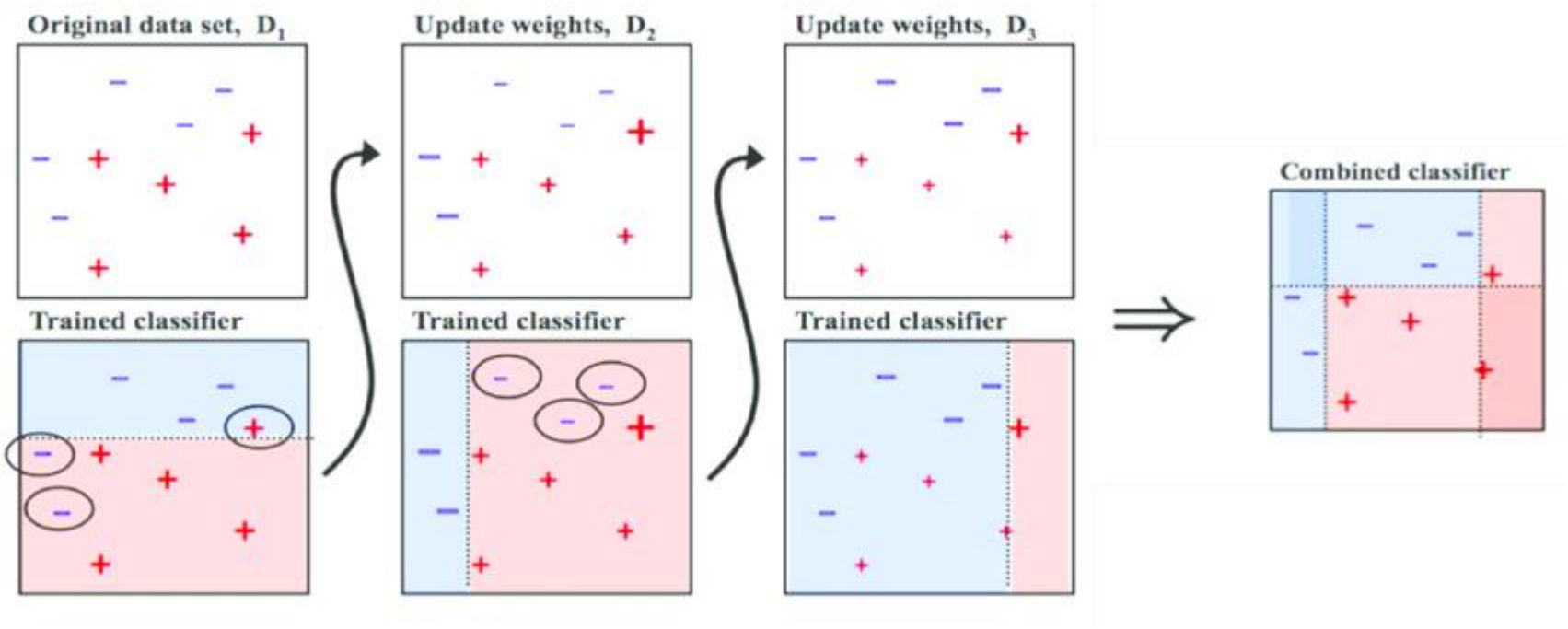

(b) boosting

- Example: Adaboost: classifiers are trained in iterations on weighted versions of the dataset.

Source: https://medium.com/diogo-menezes-borges/boosting-with-adaboost-and-gradient-boosting-9cbab2a1af81

# Boosting: Introduction

- Example: AdaBoost
  - At the end of every model prediction: “boost” the weights of the misclassified instances so that the next model does a better job on them.




Source: https://medium.com/diogo-menezes-borges/boosting-with-adaboost-and-gradient-boosting-9cbab2a1af81

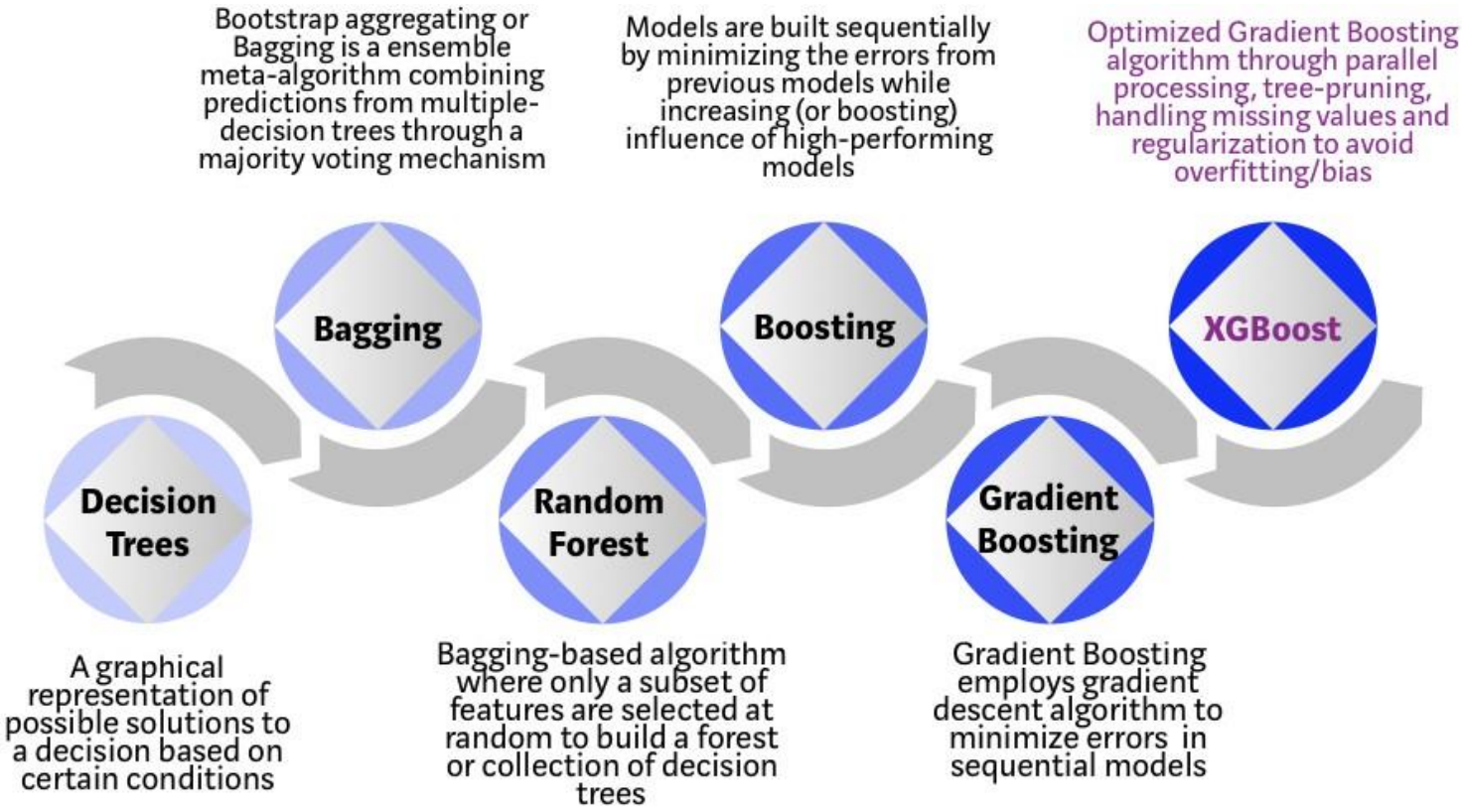
# Boosting: Introduction

- There are many boosting techniques
  - Gradient Boosting
    - Sequentially add the previous predictors' underfitted predictions to the ensemble, ensuring the errors made previously are corrected.
  - XGBoost
    - Extreme Gradient Boosting
    - Advanced implementation of Gradient Boosting
    - High predictive power while being much faster than any other gradient boosting technique
    - Includes a variety of regularization techniques (will be discussed later) in order to reduce overfitting and improve overall performance.
  - Light GB
    - Suitable for extremely large datasets

Source: https://medium.com/diogo-menezes-borges/boosting-with-adaboost-and-gradient-boosting-9cbab2a1af81

# Ensemble Learning: Summary

- Evolution of Classifiers




Source: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

# Agenda – Part 3

- ☐ Classification Goals
- ☐ k-Nearest Neighbors (kNN)
- ☐ Decision Trees
- ☐ Ensemble Methods
- ■ Evaluating Classification Models
- ☐ Summary & Outlook

# Evaluation of Classification Models: Introduction

- Models are evaluated by measuring the correctly and incorrectly classified instances

- For a two-class problem, four cases can occur:
  - true positive (TP): actual and predicted classes are both positive
  - false positive (FP): actual class is negative, predicted class is positive
  - true negative (TN): actual and predicted classes are both negative
  - false negative (FN): actual class is positive, predicted class is negative

- We use a confusion matrix to show these values:

| | | Actual Class | |
|---|---|---|---|
| | | Class = Yes | Class = No |
| Predicted Class | Class = Yes | TP | FP |
| | Class = No | FN | TN |

# Evaluation of Classification Models: Accuracy

- We already used the **accuracy** measure:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Example:

| | | Actual Class | |
|---|---|---|---|
| | | Class = Yes | Class = No |
| Predicted Class | Class = Yes | 107 (TP) | 12 (FP) |
| | Class = No | 8 (FN) | 73 (TN) |

$$Accuracy = \frac{107 + 73}{107 + 73 + 12 + 8} = 0.9$$

# Agenda – Part 3

- [ ] Classification Goals
- [ ] k-Nearest Neighbors (kNN)
- [ ] Decision Trees
- [ ] Ensemble Methods
- [ ] Evaluating Classification Models
- [x] Summary & Outlook

# Thank you!