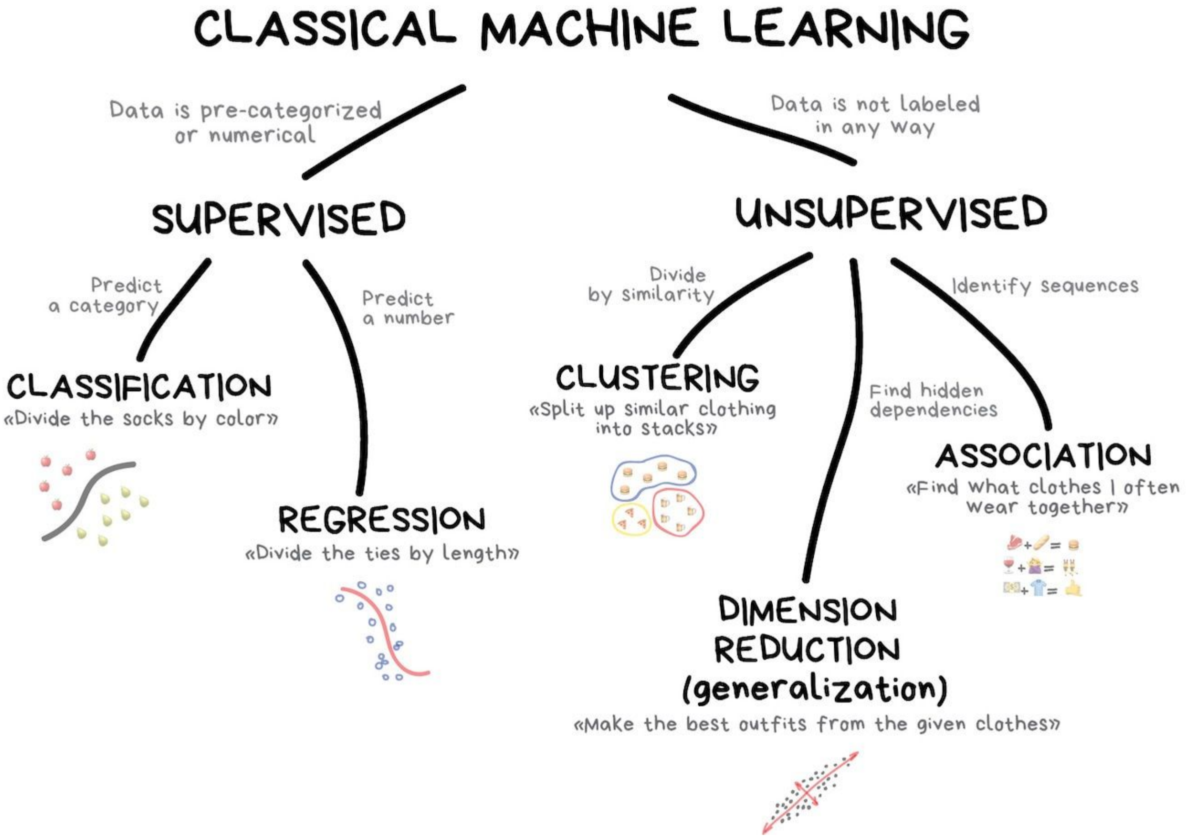
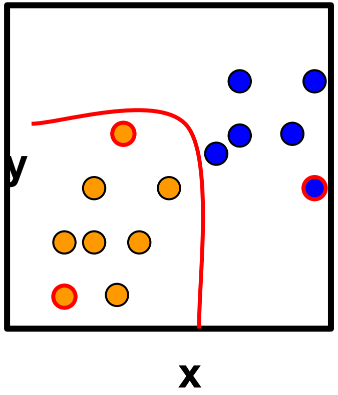
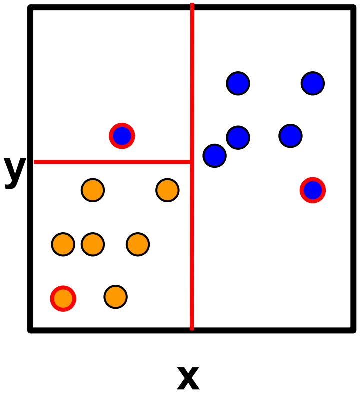
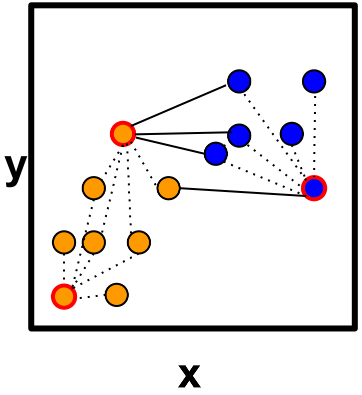


CLASSICAL MACHINE LEARNING
Data is pre-categorized or numerical
Data is not labeled in any way
SUPERVISED
UNSUPERVISED
Predict a category
Predict a number
CLASSIFICATION
«Divide the socks by color»
REGRESSION
«Divide the ties by length»
Divide by similarity
Identify sequences
Find hidden dependencies
CLUSTERING
«Split up similar clothing into stacks»
ASSOCIATION
«Find what clothes I often wear together»
DIMENSION REDUCTION (generalization)
«Make the best outfits from the given clothes»







Supervised classification methods

Supervised learning methods learn by example

Used to classify and predict (regression):

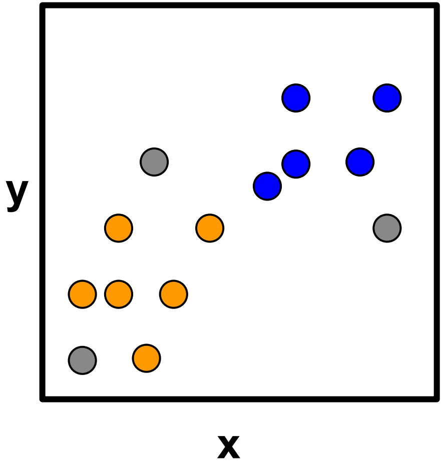
- Similarity can be used in conjunction to parametric or non-parametric methods
- Need labels, in some cases a lot of labels
- Dependent on the definition of similarity

Clustering vs classifying

unsupervised / *supervised*

The goal is to partition the space so that the **unobserved** variables are separated in groups consistently with an observed subset

target features: x and y



target features: color

models typically return a partition of the space

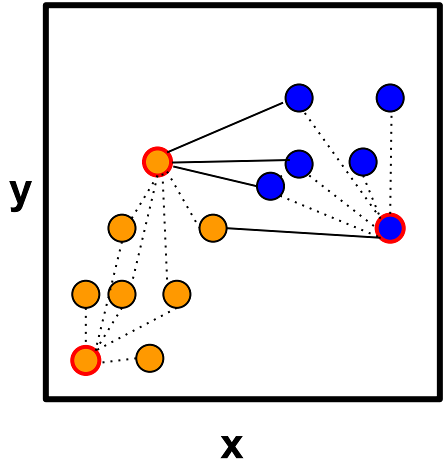
Supervised ML: classification

A subset of variables has class labels. Guess the label for the other variables

KNearest Neighbors

Assigns the class of closest neighbors

target features:
y and y



target features:
color

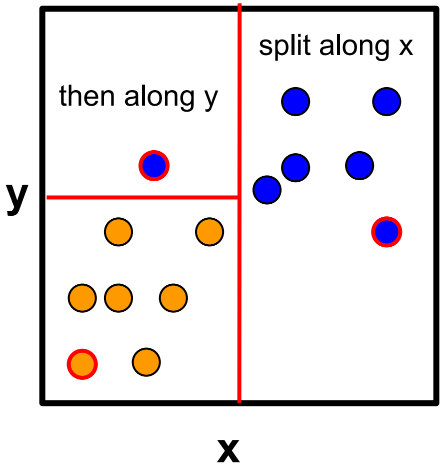
Supervised ML: classification

A subset of variables has class labels. Guess the label for the other variables

Tree methods

Split spaces along each axis separately

target features:
x and y



target features:
color

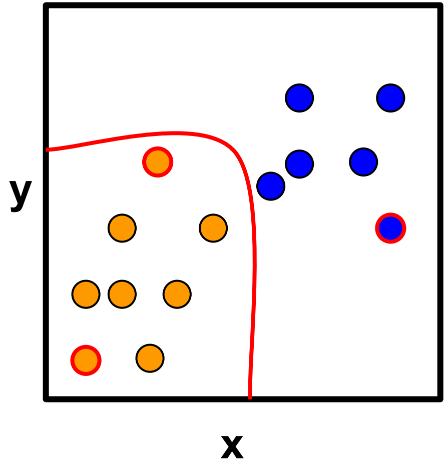
Supervised ML: classification

A subset of variables has class labels. Guess the label for the other variables

SVM (support-vector machine)

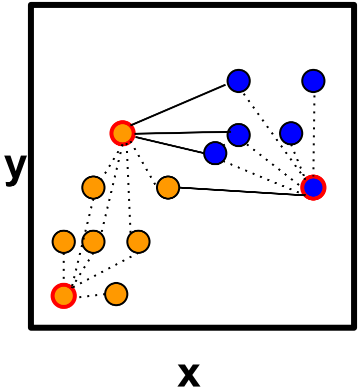
finds a hyperplane that optimally separates observations

target features: x and y



target features: color

models typically return a partition of the space



K-Nearest Neighbors

Lazy learner: k-Nearest Neighbors

1. Calculate the distance d to all known objects
2. Select the k closest objects

Classification:

Assign the most common among the k classes

Regression:

Predict the average (median) of the k target values

k-Nearest Neighbors

Good

non parametric

very good with large training sets

Not so good

it is only as good as the distance metric

If the similarity in feature space reflect similarity in label then it is perfect!

poor if training sample is sparse

poor with outliers

Lazy learning

Evaluation on demand, no global optimization - doesn’t learn a discriminative function from the training data but “memorizes” the training dataset instead.

Pros

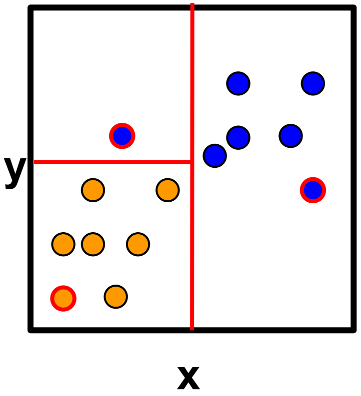
Because the model does not need to provide a global optimization the classification is "on-demand".

This is ideal for recommendation systems: think of Netflix and how it provides recommendations based on programs you have watched in the past.

Cons

Need to store the entire training dataset (cannot model data to reduce dimensionality).

Training==evaluation => there is no possibility to frontload computational costs



CART: Classification and Regression trees

Single tree

Application:

a robot to predict surviving the Titanic
(https://www.kaggle.com/c/titanic)

features:

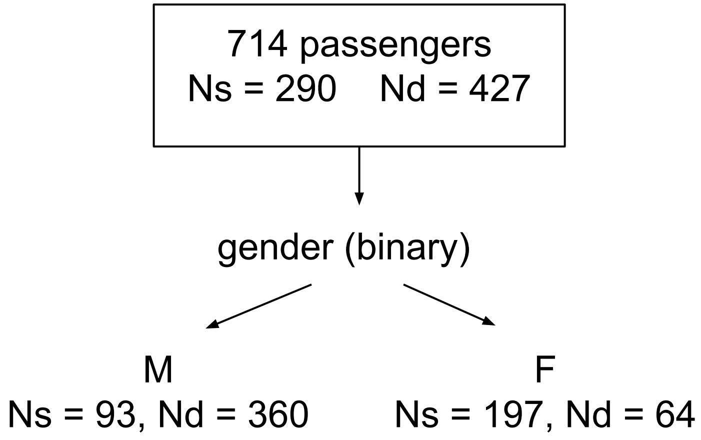
gender

ticket class

age

target variable:

-> survival (y/n)



Single tree

Application:

a robot to predict surviving the Titanic
(https://www.kaggle.com/c/titanic)

features:

gender 79% | 75%

ticket class

age

target variable:

-> survival (y/n)

714 passengers
Ns = 290 Nd = 427

gender (binary)

M
Ns = 93, Nd = 360

p = 360/ (360 + 93)
79%

F
Ns = 197, Nd = 64

p = 197/ (197 + 64)
75%

Optimize over purity:
p = N largest class / N totalset

Single tree

Application:

a robot to predict surviving the Titanic
(https://www.kaggle.com/c/titanic)

features:

gender 79% | 75%

ticket class 66% | 54 %

age

target variable:

-> survival (y/n)

714 passengers
Ns = 290 Nd = 427

Class (ordinal)

1st
Ns = 120, Nd = 80

p = 66%

2nd+3rd
Ns = 234, Nd = 298

p = 54%

Optimize over purity:
p = N largest class / N totalset

Single tree

Application:

a robot to predict surviving the Titanic
(https://www.kaggle.com/c/titanic)

features:

gender 79% | 75%

ticket class 66% | 54 %

age 66% | 61%

target variable:

-> survival (y/n)

714 passengers
Ns = 290 Nd = 427

age (continuous)

>6.5
Ns = 250, Nd = 107

p = 66%

<= 6.5
Ns = 139, Nd = 217

p = 61%

Optimize over purity:
p = N largest class / N totalset

Dataset splits performed by from the features with highest purity

Single tree

Application:

a robot to predict surviving the Titanic
(https://www.kaggle.com/c/titanic)

features:

gender 79% | 75%

ticket class M 60|85% F 96|65%

age M 74|67% F 66|60%

target variable:

-> survival (y/n)

714 passengers
Ns = 290 Nd = 427

gender

M
Ns = 93, Nd = 360

F
Ns = 197, Nd = 64

Single tree

Application:

a robot to predict surviving the Titanic
(https://www.kaggle.com/c/titanic)

features:

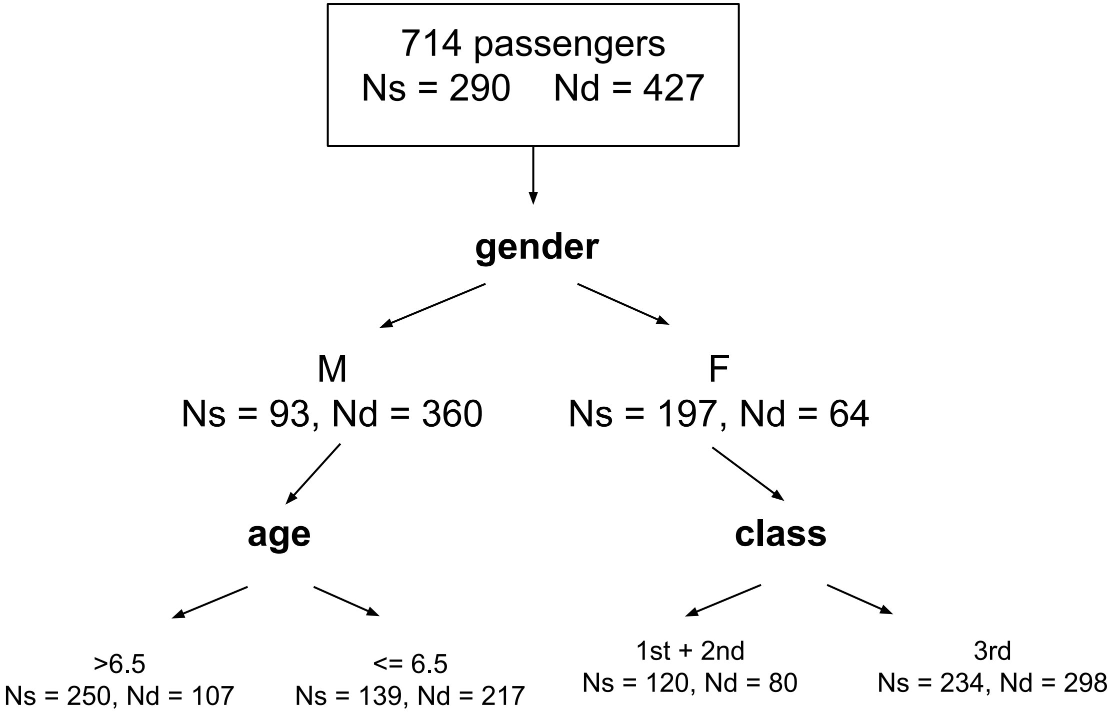
gender 79% | 75%

ticket class M 60|85% **F 96|65%**

age **M 74|67%** F 66|60%

target variable:

-> survival (y/n)



Single tree

Application:

a robot to predict surviving the Titanic
(https://www.kaggle.com/c/titanic)

features:

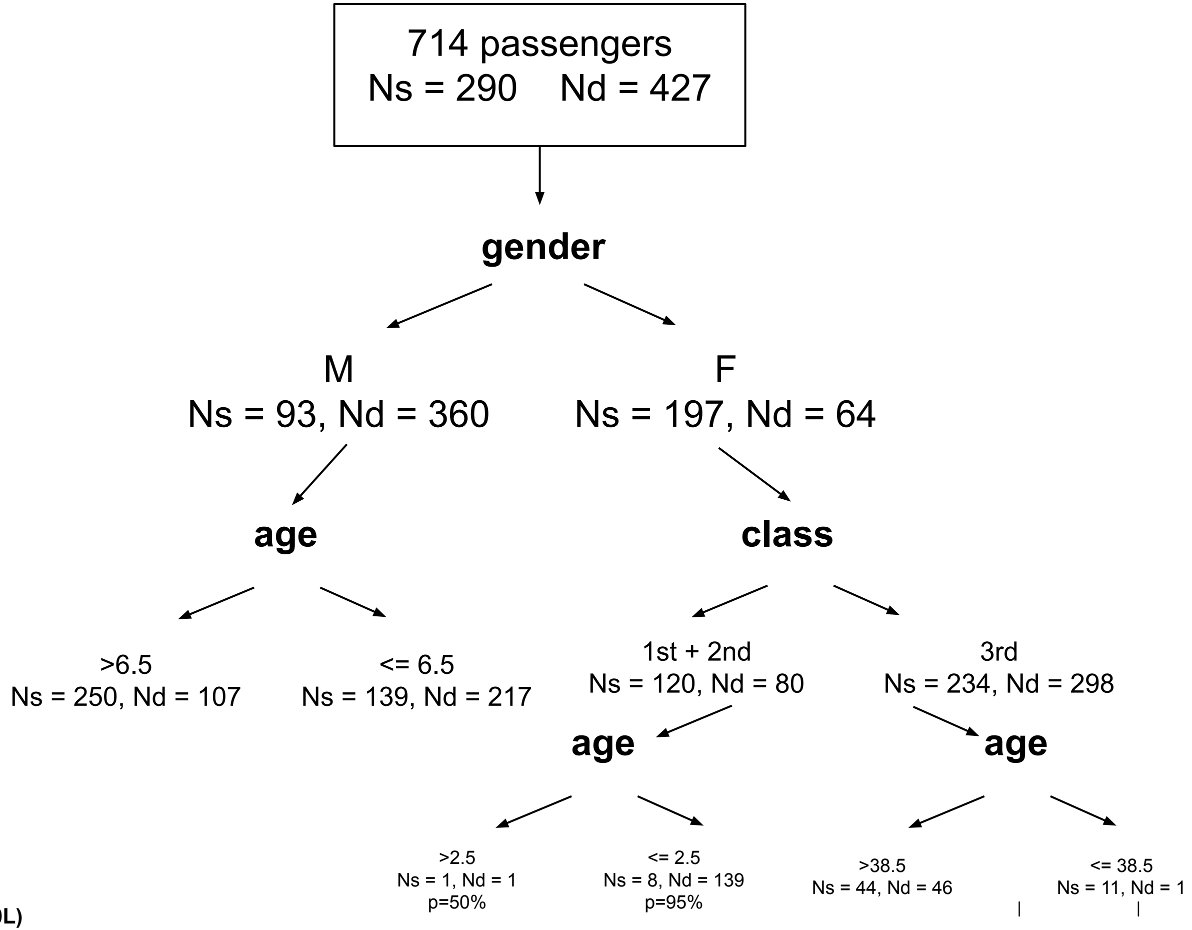
gender (binary, already used)

ticket class (ordinal)

age (continuous)

target variable:

-> survival (y/n)



Single tree

Application:

a robot to predict surviving the Titanic
(https://www.kaggle.com/c/titanic)

features:

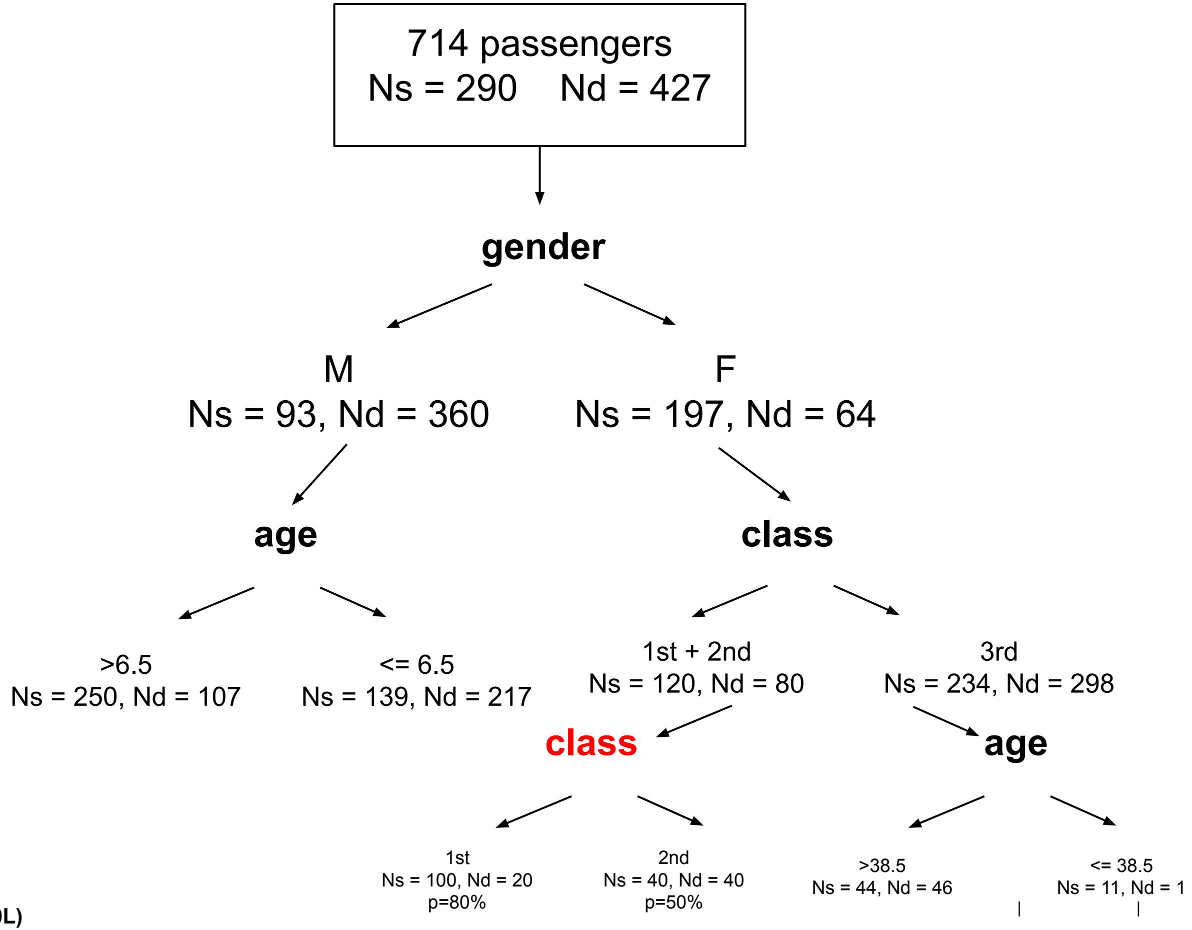
gender (binary, already used)

ticket class (ordinal)

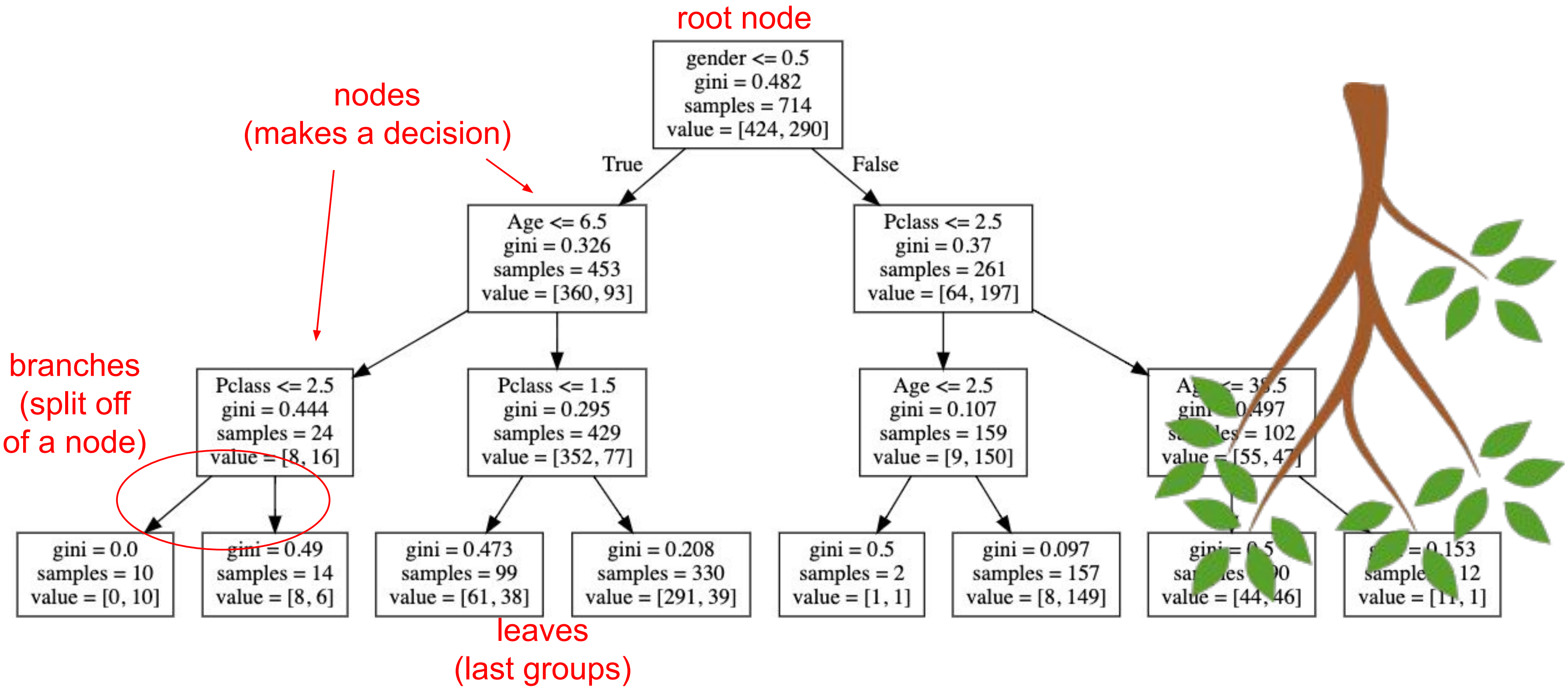
age (continuous)

target variable:

-> survival (y/n)



A single tree



Tree hyperparameters

sklearn.tree.DecisionTreeClassifier¶

class `sklearn.tree.` **`DecisionTreeClassifier`** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*)
[source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *string, optional (default="gini")*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *string, optional (default="best")*

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int or None, optional (default=None)*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int, float, optional (default=2)*

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Tree hyperparameters

gini impurity is the is a measure of how often a randomly chosen element from the set would be <u>incorrectly</u> labeled if it was randomly labeled according to the distribution of labels in the subset (zero if the node falls in just one target category).

sklearn.tree.DecisionTreeClassifier¶

class `sklearn.tree.` **`DecisionTreeClassifier`** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *string, optional (default="gini")*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *string, optional (default="best")*

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int or None, optional (default=None)*

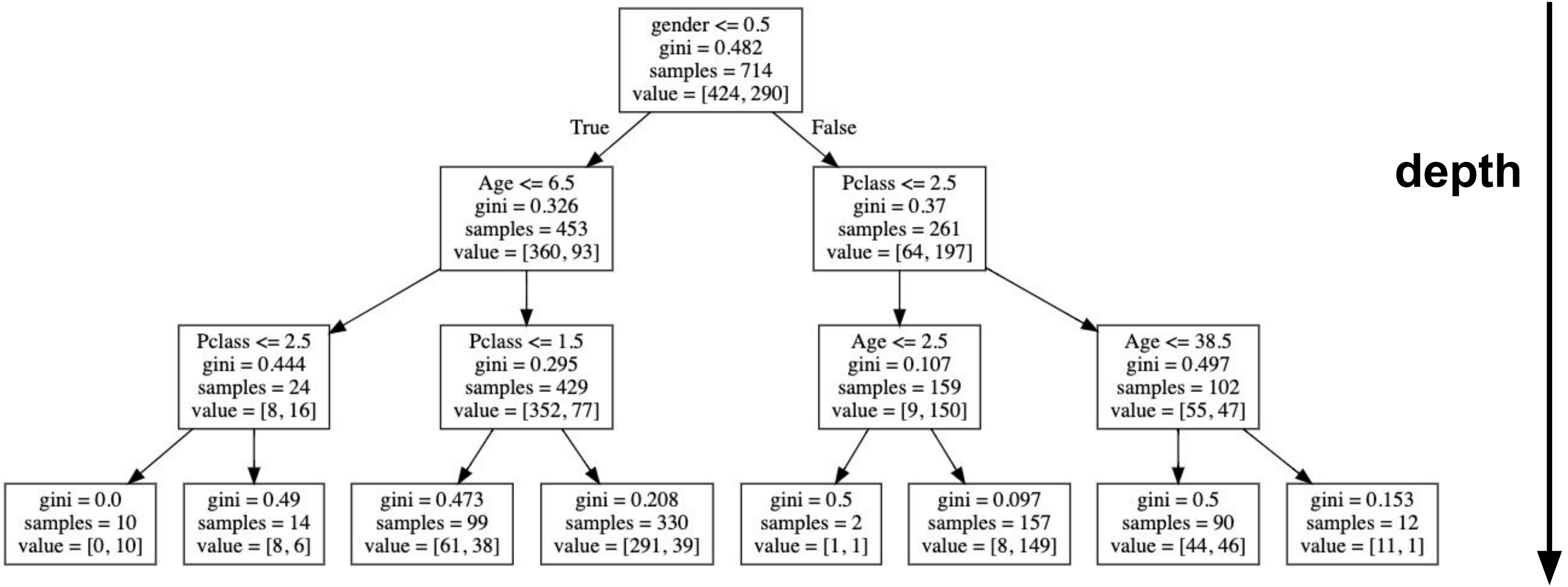
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int, float, optional (default=2)*

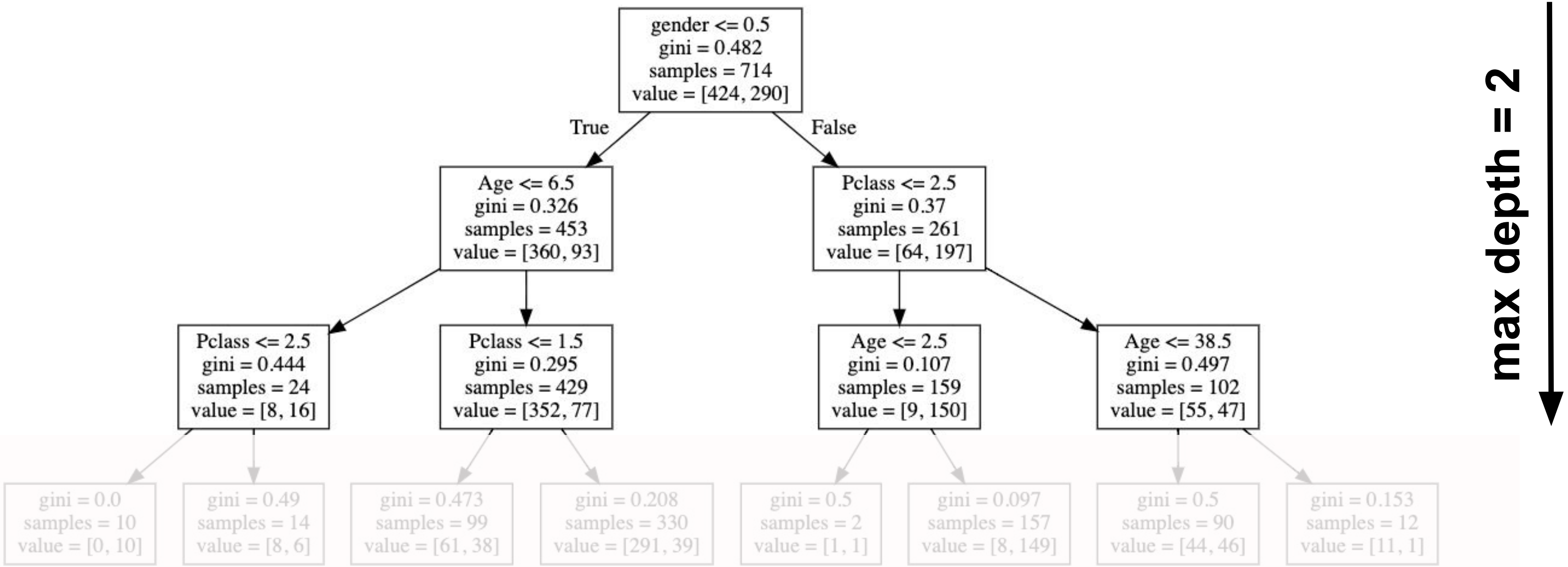
The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Tree hyperparameters

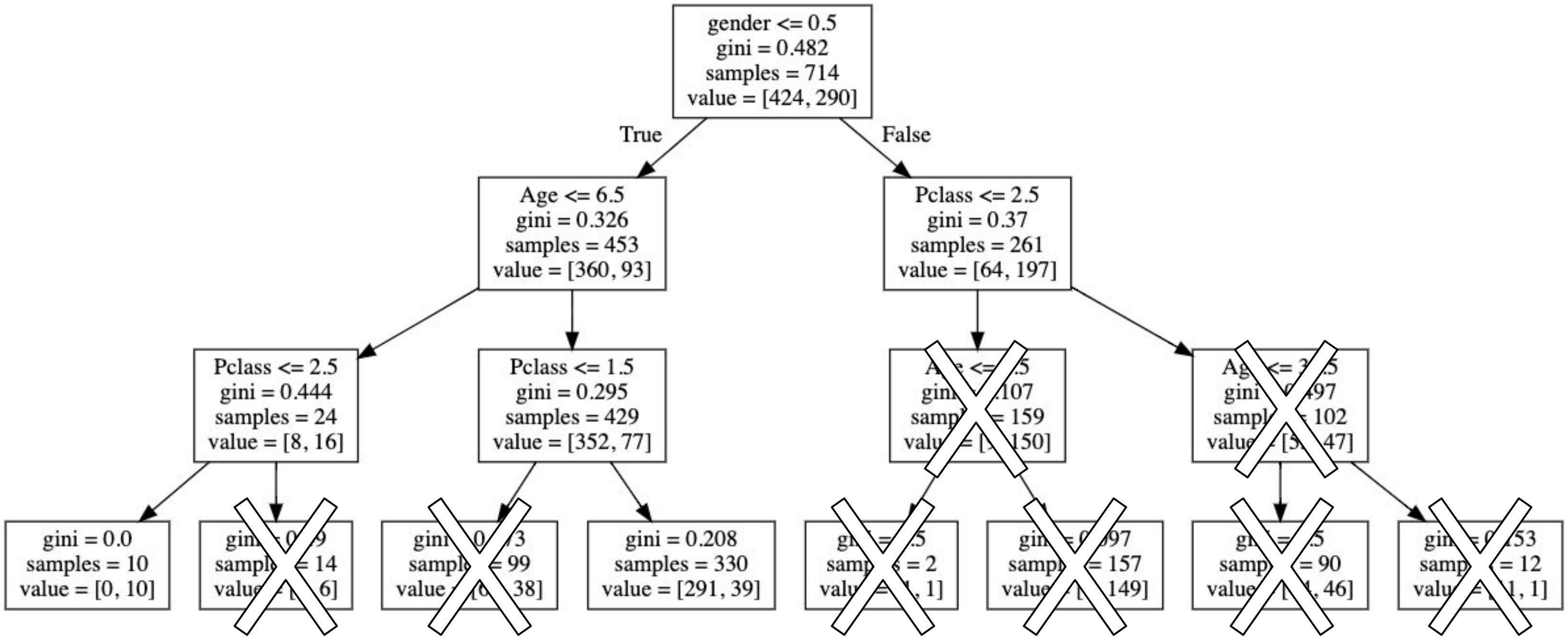


Tree hyperparameters



prevents overfitting

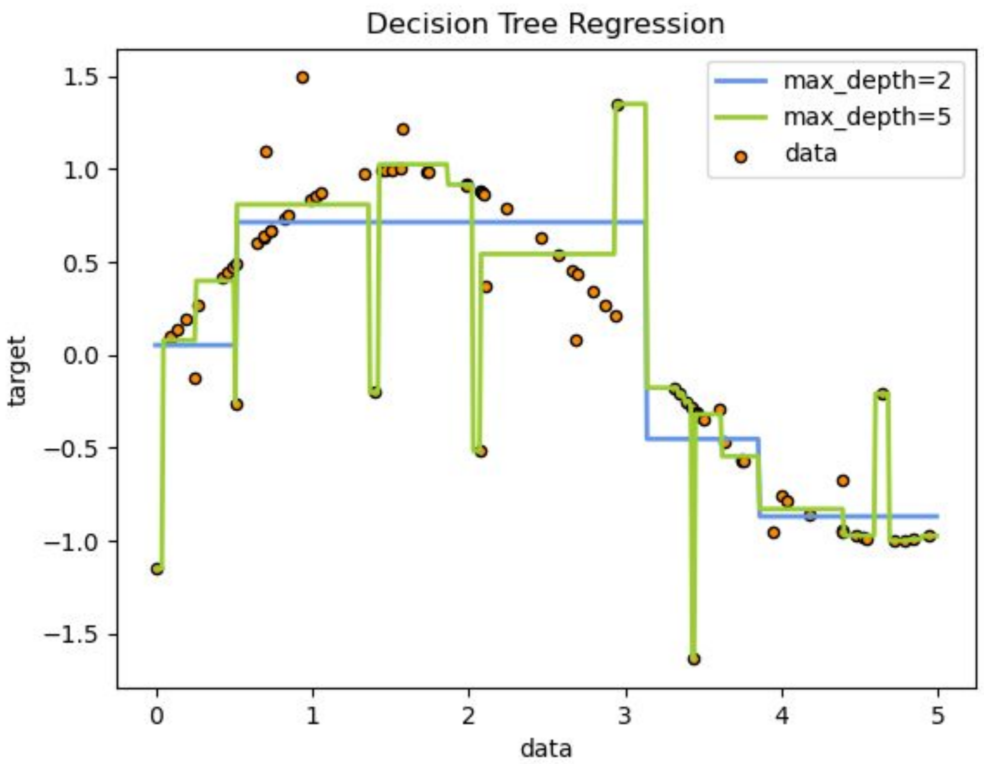
Tree hyperparameters



Alternative: tree pruning

Decision tree regression

Trees can be used for regression
(think about it as very many small classes)



Tree ensembles

Issue with trees

Variance: different trees lead to different results

why?

because calculating the criterion for every split and every note is an intractable problem (think about continuous variables)!

Solution: use many trees and take an ensemble decision

Random forests

Many parallel trees

Gradient Boosted Trees

A series of trees

Ensemble methods

run multiple versions of the same model with some small (stochastic or progressive) variation and learn from the ensemble of methods

Random forests

trees run in parallel (independently of each other)

each tree uses a random subset of observations/features (bootstrap - bagging)

class predicted by majority vote:

what class do most trees think a point belong to

Gradient Boosted Trees

trees run in series (one after the other)

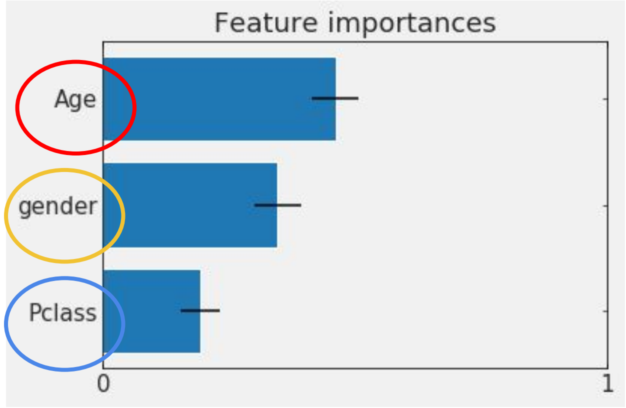
each tree uses different weights for the features learning the weights from the previous tree

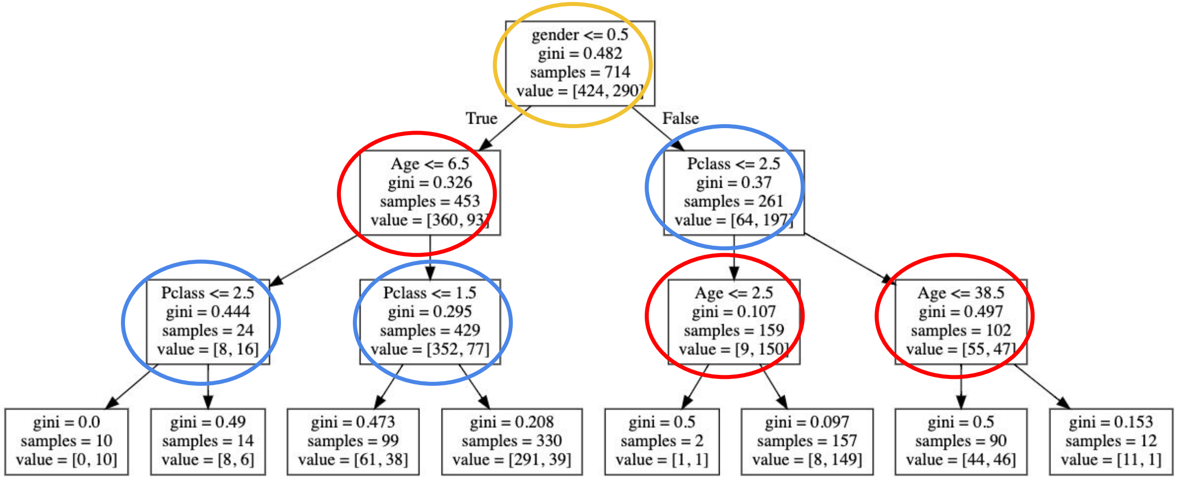
the last tree has the prediction

Feature importance

In principle **CART methods are interpretable:**

you can measure the influence that each feature has on the decision : feature importance





In practice the interpretation is complicated by covariance of features