Machine Learning for Networking
ML4N

Luca Vassio
Gabriele Ciravegna
Zhihao Wang
Tailai Song

Recap

- Data, Model, Loss
- ERM (parametrized) $\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^n} f(\mathbf{w})$
- Model selection and hyper-parameters tuning through validation set
- How to minimize the function (learn)?

  In some cases, Gradient Descent

What is missing?

- Show combinations of data/model/loss and an **algorithm (technique) to perform ERM**
  - How to find best hypothesis h in H
  - In practice, models are always parametrized
  - How to find find the best parameters **w**

Supervised learning techniques

Regression

- Linear and polynomial regressors
- LASSO regressor
- Decision tree and random forest
- k-Nearest Neighbors
- Neural networks (next lectures)

Classification

- Logistic regression
- Support Vector Machines
- Naive Bayes classifiers
- Decision tree and random forest
- k-Nearest Neighbors
- Neural networks (next lectures)

LASSO regressor

LASSO: design choices

- Regressor
- LASSO (least absolute shrinkage and selection operator)
- **Datapoints** with numeric features and label
- **Model** = space of linear maps
- **Regularized squared loss**

LASSO

- Linear regression requires a training set larger than the number of features (m>n) to not overfit
- However, sometimes m<n

LASSO

- Linear regression requires a training set larger than the number of features (m>n) to not overfit
- However, sometimes m<n
- **Regularization technique**: **penalty term in the loss for using too many features**
- Regularized squared loss

$$L\left((\mathbf{x},y),h^{(\mathbf{w})}\right)=\left(y-\mathbf{w}^T\mathbf{x}\right)^2+\lambda\|\mathbf{w}\|_1.$$

LASSO

- Regularization parameter λ
  - Increasing λ results in a weight vector **w** with increasing number of zero coefficients
  - Works as a **feature selection**

$$L\left((\mathbf{x}, y), h^{(\mathbf{w})}\right) = \left(y - \mathbf{w}^T\mathbf{x}\right)^2 + \lambda\|\mathbf{w}\|_1.$$

LASSO in Python

sklearn.linear_model.Lasso

class sklearn.linear_model.**Lasso**(*alpha=1.0, *, fit_intercept=True, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'*) [source]

Linear Model trained with L1 prior as regularizer (aka the Lasso).

Minimization through coordinate descent (a variation of gradient descent)

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

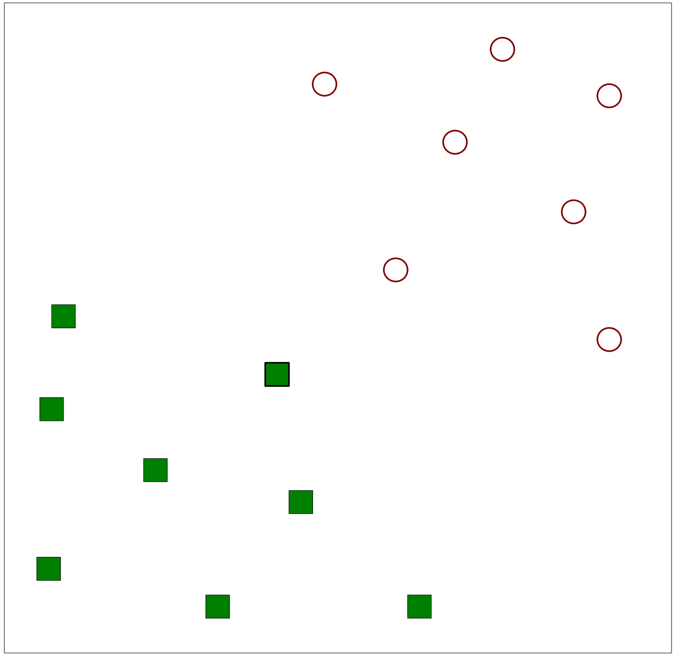
Support Vector Machines (SVM)

SVM: design choices

- Classifier (can be extended as regressor)
- **Datapoints** with numeric features
- Binary label values, e.g., y=-1 vs. y=1
- **Model** = space of linear maps
- **Regularized hinge loss**

Support Vector Machines

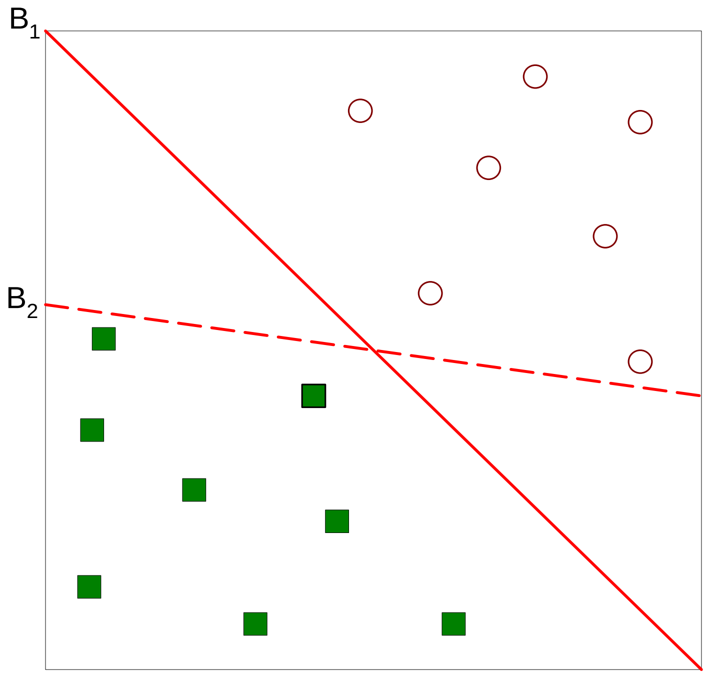
- Find a linear hyperplane (decision boundary) that will separate the data



Support Vector Machines

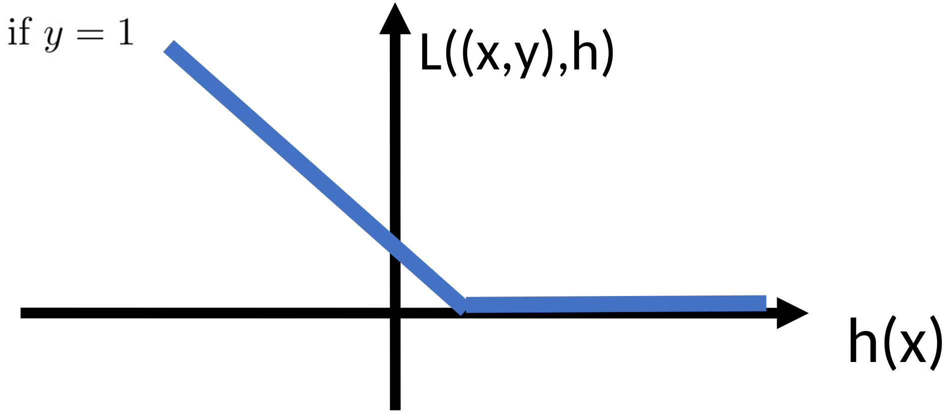
- Find a linear hyperplane (decision boundary) that will separate the data

Which one is better? B1 or B2?



Support Vector Machines

- Hinge loss: $L\left((\mathbf{x},y),h\right) := \max\{0, 1 - yh(\mathbf{x})\}.$
- For a linearly separable dataset, there might be infinite maps with 0 average hinge loss
- Regularized hinge loss:



$$L\left((\mathbf{x},y),h^{(\mathbf{w})}\right) := \max\{0, 1 - y \cdot h^{(\mathbf{w})}(\mathbf{x})\} + \lambda \left\|\mathbf{w}\right\|_2^2$$

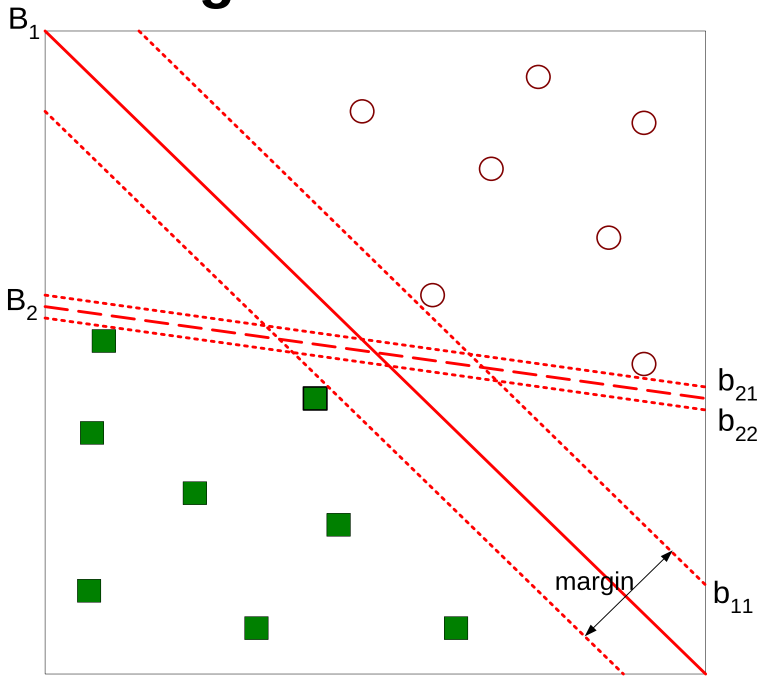
$$\overset{h^{(\mathbf{w})}(\mathbf{x})=\mathbf{w}^T\mathbf{x}}{=} \max\{0, 1 - y \cdot \mathbf{w}^T\mathbf{x}\} + \lambda \left\|\mathbf{w}\right\|_2^2 .$$

Support Vector Machines

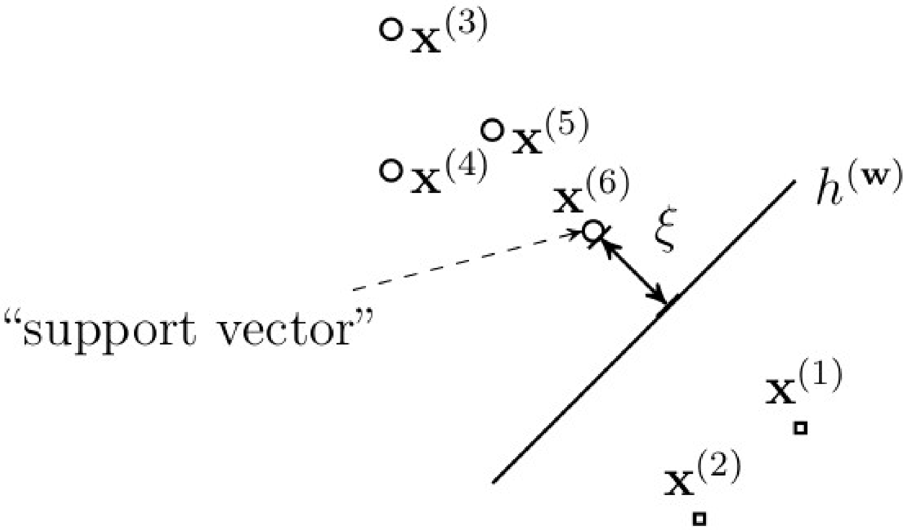
- Minimize average regularized hinge loss = maximize the margin
- Find a linear hyperplane (decision boundary) that will separate the data **and maximizes the margin**

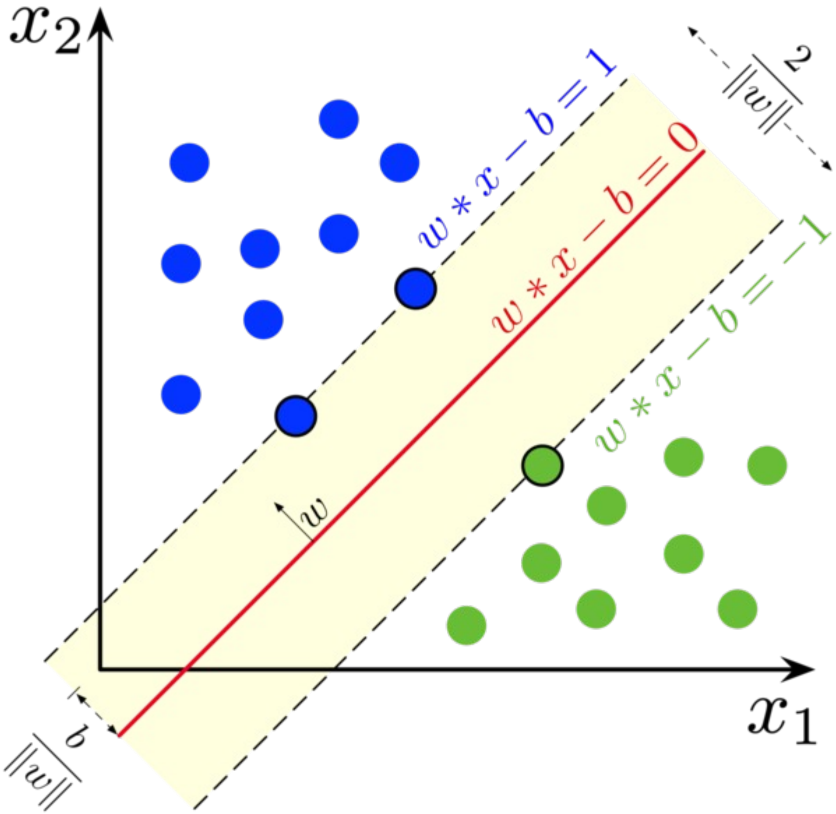
Which one is better? B1 or B2?

B1 is better:
maximizes the margin



Support Vector Machines

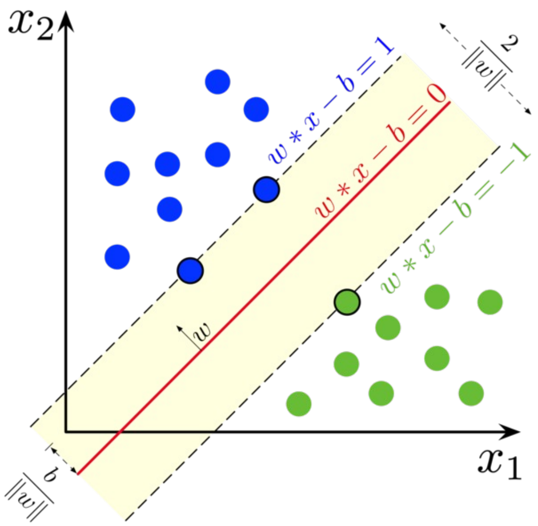




https://en.wikipedia.org/wiki/Support_vector_machine

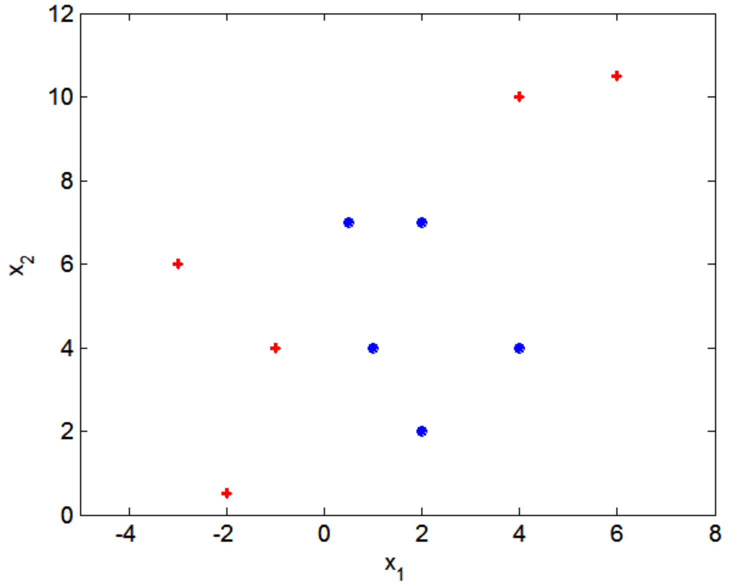
Support Vector Machines

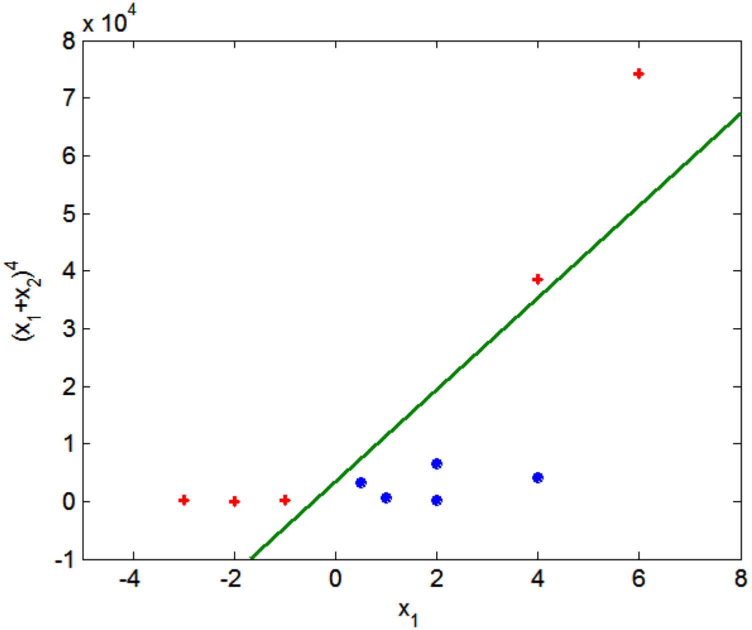
- The loss favors linear maps h(w) that are robust against (small) perturbations of the data points – **more robust than logistic regressor**
- The margin is the minimum distance of all closest point (misclassified have negative distance)
  - We allow for error terms in case there is no hyperplane



Support Vector Machines

- What if decision boundary is not linear?
- Transform data into higher dimensional space (non-linear kernels)





SVM in Python

sklearn.svm.SVC

class sklearn.svm.**SVC**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

C-Support Vector Classification.

sklearn.svm.LinearSVC

class sklearn.svm.**LinearSVC**(*penalty='l2', loss='squared_hinge', *, dual='warn', tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*) [source]

Linear Support Vector Classification.

Learn through quadratic programming (QP) problem and solver, or through gradient descent (and its variations)

https://scikit-learn.org/stable/modules/svm.html

Naive Bayes classifier

Naive Bayes classifier: design choices

- Classifier
- **Datapoints** with numeric features
- Binary label values, e.g., y=-1 vs. y=1
- **model** = space of linear maps
- **0/1 loss**

Bayes classifiers

- ERM with **0/1 loss difficult for gradient methods**
- ERM aims at estimating Bayes risk p(h(**x**) ≠ y) (i.e., expected 0/1 loss)
- Given the features **x**, we want to classify the sample as the class y with maximum p(y | **x**)
- Binary case:

$$\widehat{h}(\mathbf{x}) = \begin{cases} 1 & \text{if } p(y=1|\mathbf{x}) > p(y=-1|\mathbf{x}) \\ -1 & \text{otherwise.} \end{cases}$$

- We do not have this probability distribution → estimate it from training points

Naive Bayes classifier

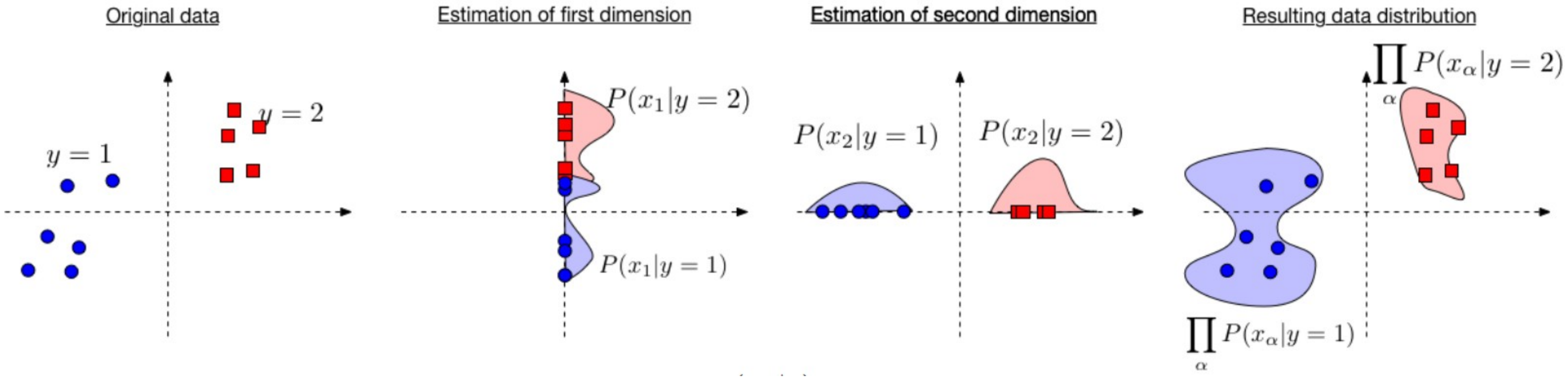
- Bayes' rule:

$$p(y|\mathbf{x}) = p(\mathbf{x}|y)\cdot p(y) \,/\, p(\mathbf{x})$$

- $p(\mathbf{x})$ constant for all y, disregarded for maximum computation
- $p(y)$ probability of class y → estimated by relative frequency of class y in the training set
- How to estimate $p(\mathbf{x}|y)$, i.e. $p(x_1,\ldots,x_n|y)$?
  - Different Bayes estimators for different probabilistic models of features $p(\mathbf{x}|y)$
  - Naive hypothesis: $p(x_1,\ldots,x_n|y) = p(x_1|y)\, p(x_2|y) \ldots p(x_n|y)$
  - Statistical independence of attributes $x_1,\ldots,x_n$

Naive Bayes classifier

- Estimate p(**x**|y) with Naive hypothesis



https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote05.html

Naive Bayes classifier

$$
\begin{aligned}
h(\mathbf{x}) &= \operatorname*{argmax}_y P(y|\mathbf{x}) \\
&= \operatorname*{argmax}_y \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \\
&= \operatorname*{argmax}_y P(\mathbf{x}|y)P(y) && (P(\mathbf{x}) \text{ does not depend on } y) \\
&= \operatorname*{argmax}_y \prod_{\alpha=1}^{d} P(x_\alpha|y)P(y) && (\text{by the naive Bayes assumption}) \\
&= \operatorname*{argmax}_y \sum_{\alpha=1}^{d} \log(P(x_\alpha|y)) + \log(P(y)) && (\text{as log is a monotonic function})
\end{aligned}
$$

https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote05.html

Gaussian Naive Bayes

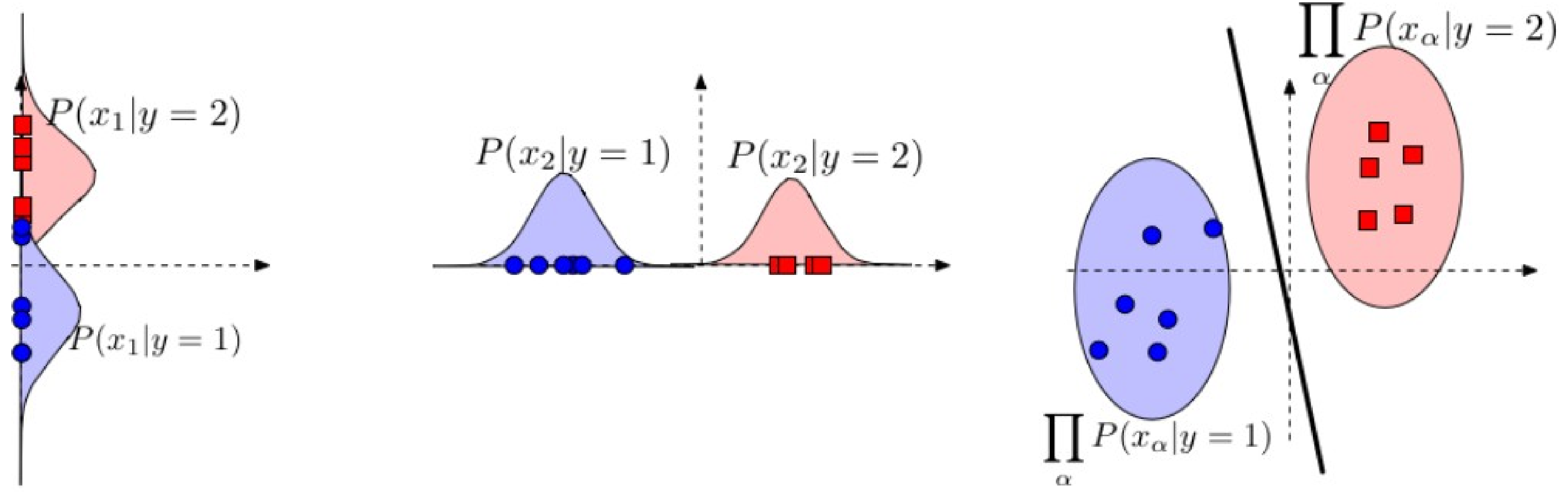
- Gaussian Naive Bayes: assume **x** is Gaussian with mean and variance depending on y

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

- Mean and variance estimated by maximum likelihood (from training set)

Naive Bayes is a linear classifier

With binary labels:



- As in logistic regression, Naive Bayes works in the space of linear maps
- Class is obtained by thresholding a linear map h(**x**)=**w**^{t}**x**

https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote05.html

Gaussian Naive Bayes in Python

sklearn.naive_bayes.GaussianNB

```
class sklearn.naive_bayes.GaussianNB(*, priors=None, var_smoothing=1e-09)
```
[source]

https://scikit-learn.org/stable/modules/naive_bayes.html
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

Decision tree (DT) and random forest (RF)

Decision tree: design choices

- Regressor and classifier
- **Datapoints** with numeric and categorical features
- Label values arbitrary
- **Model** = piece-wise constant. Maps represented by flow-chart (“decision trees”)
- Different options for **loss** function
  - Losses are optimized locally and greedily, not globally

Decision tree classifier

- Decision trees are models that uses a tree-like model of decisions and their possible outcomes/classes
- Only contains **conditional control statements**
- A decision tree can be represented as a flowchart-like structure in which:
  - each internal node represents a condition test on an attribute
  - each branch represents the outcome of the condition test
  - each leaf node represents a class label

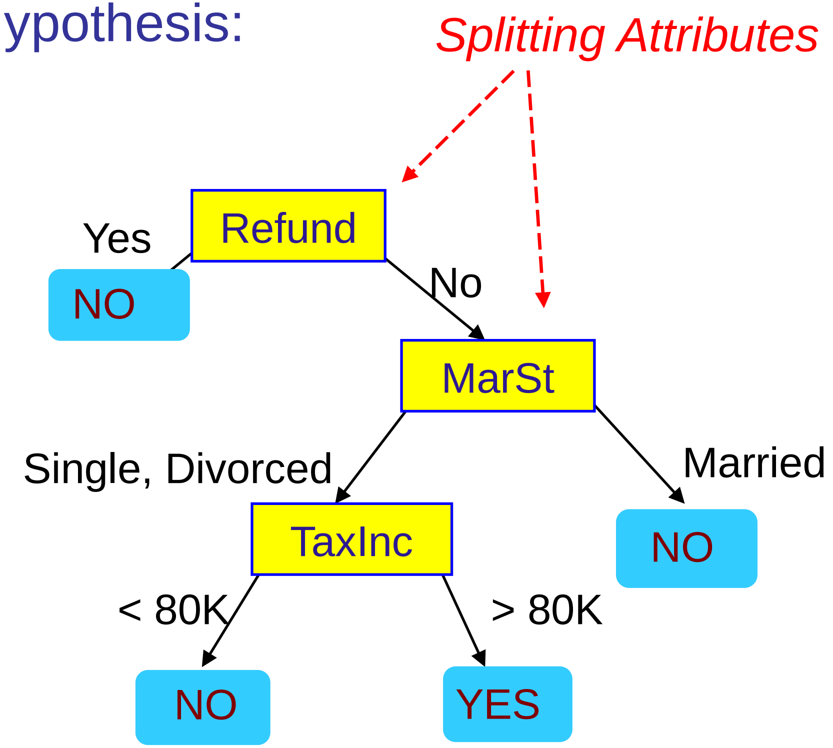
Example of decision tree

Training Data

Tid	Refund	Marital Status	Taxable Income	Cheat
	categorical	categorical	continuous	class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

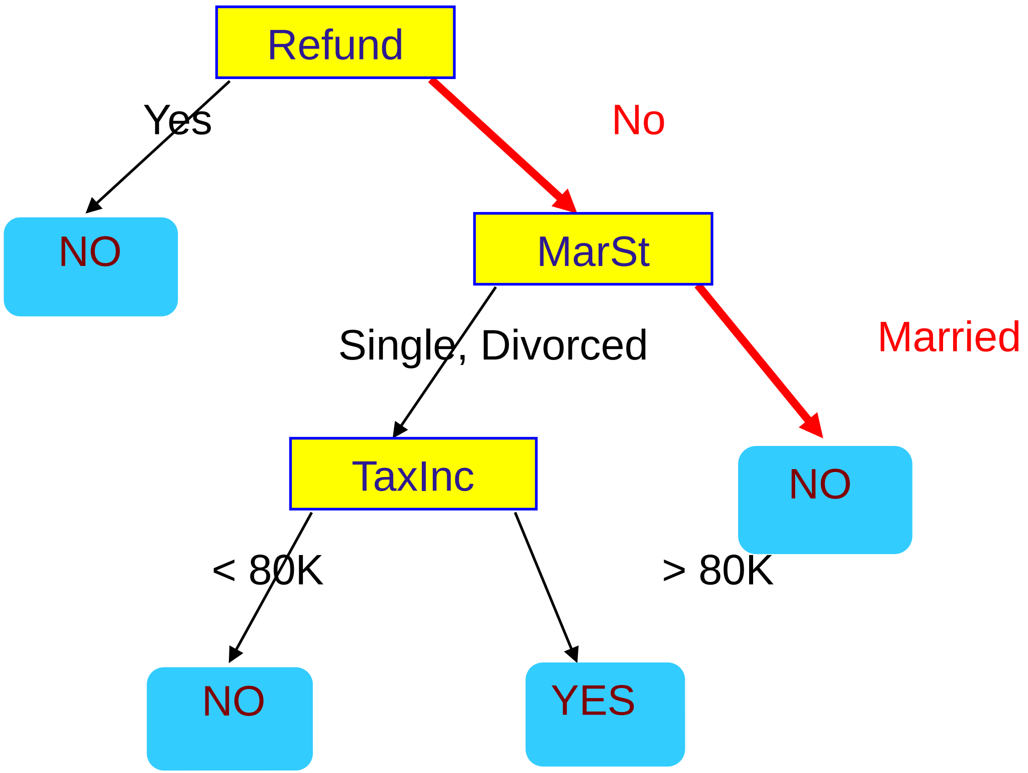
Model: Decision Tree

Hypothesis:

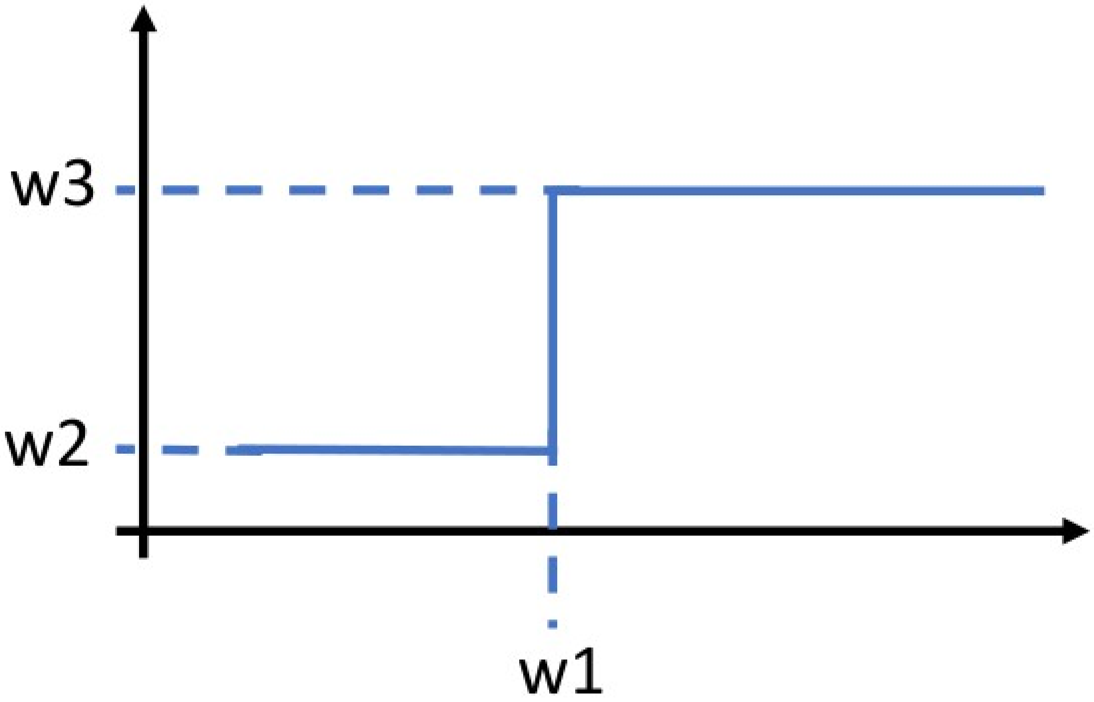


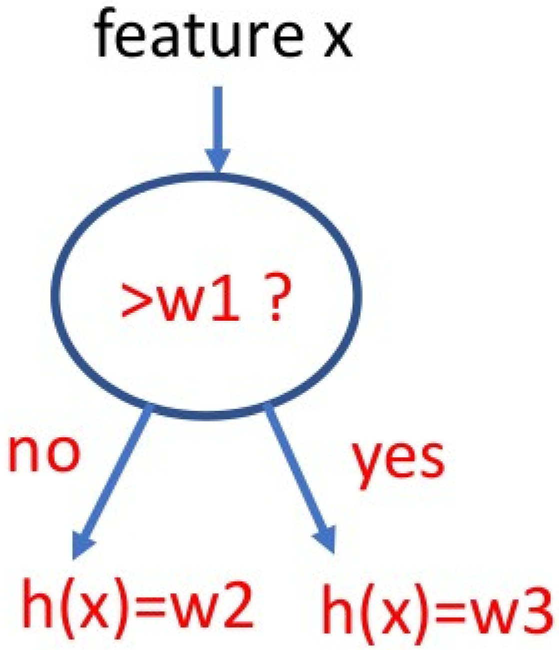
Decision tree inference

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Parametrized DT

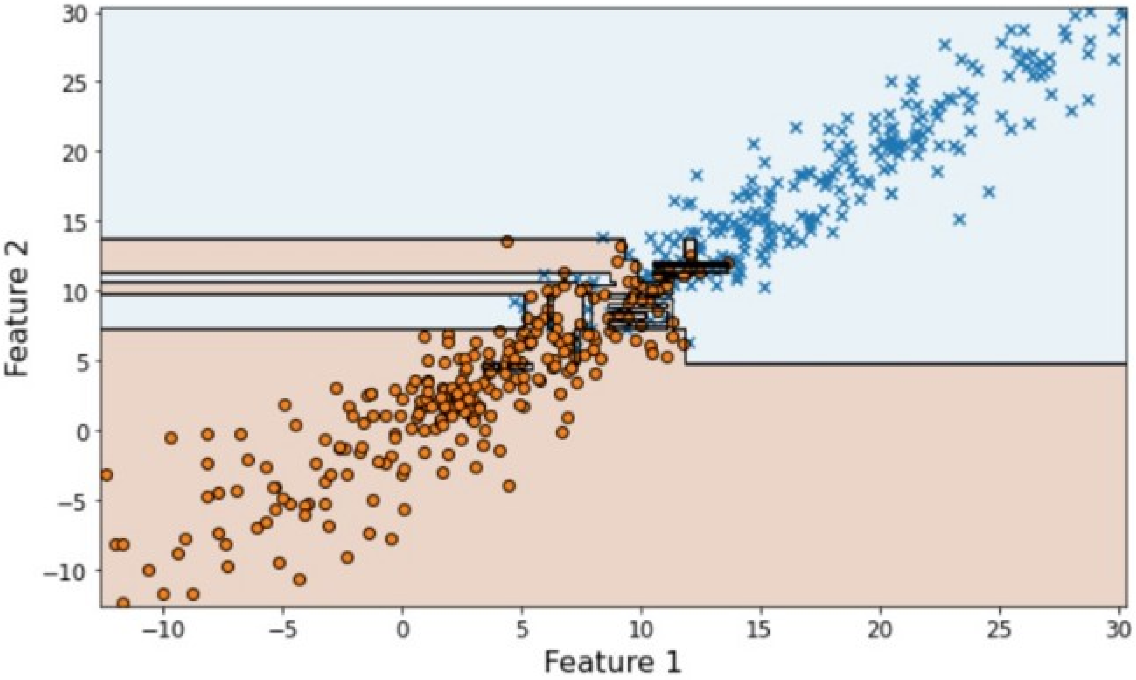




DT decision boundary

Piece-wise constant

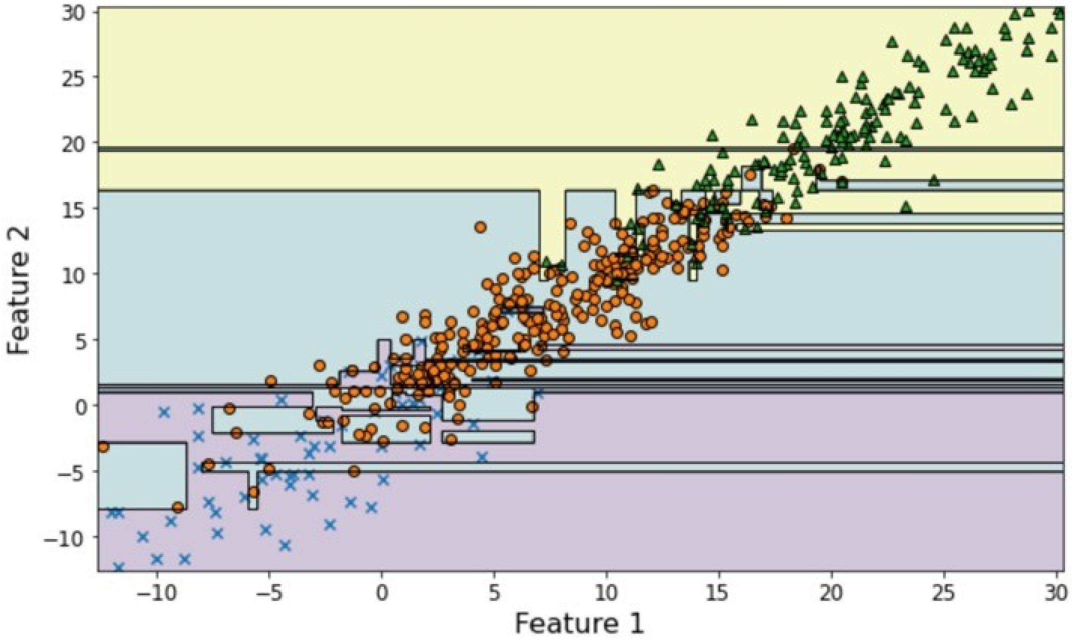
(2 classes)



DT decision boundary - multi-class

Piece-wise constant

(3 classes)



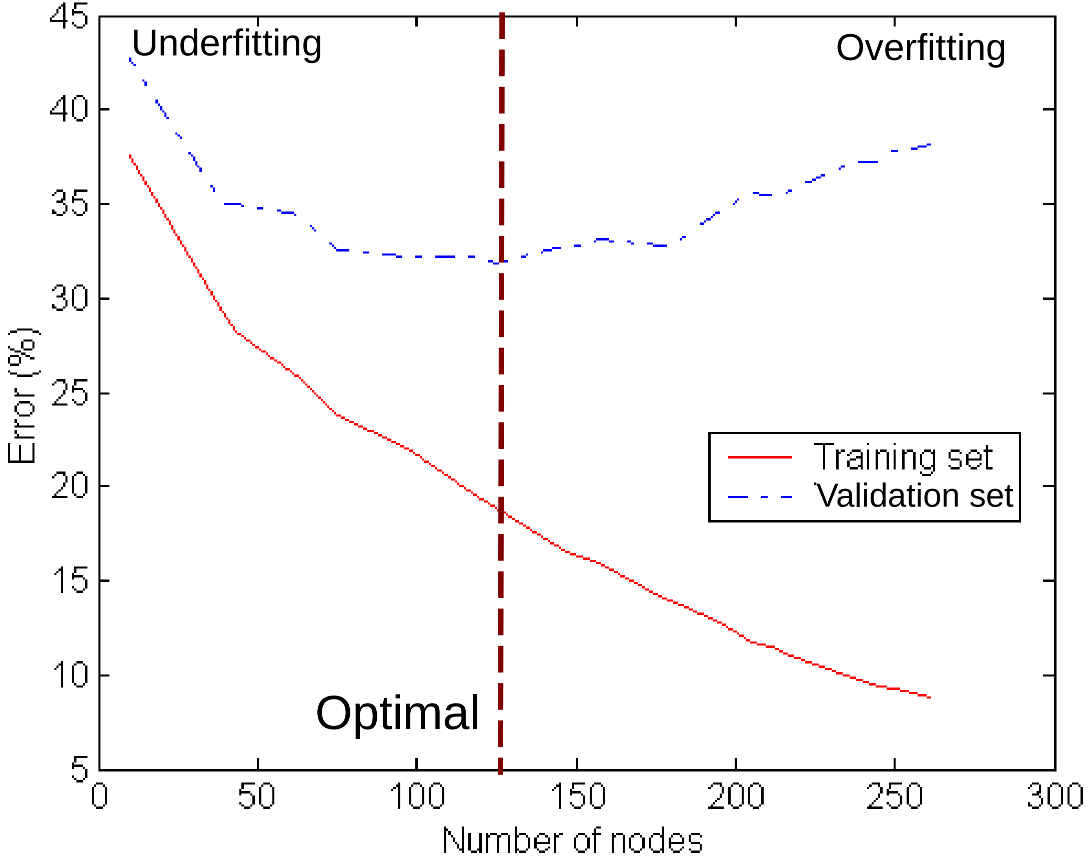
Decision tree learning algorithm

- Many algorithms: Hunt’s Algorithm, CART, ID3, C4.5, C5.0,...
- Greedy strategy: best attribute for the split is selected locally at each step, not a global optimum
- Local losses: Gini Index, information gain

Decision tree validation curve

Important hyper-parameter:

Depth of tree/Number of nodes



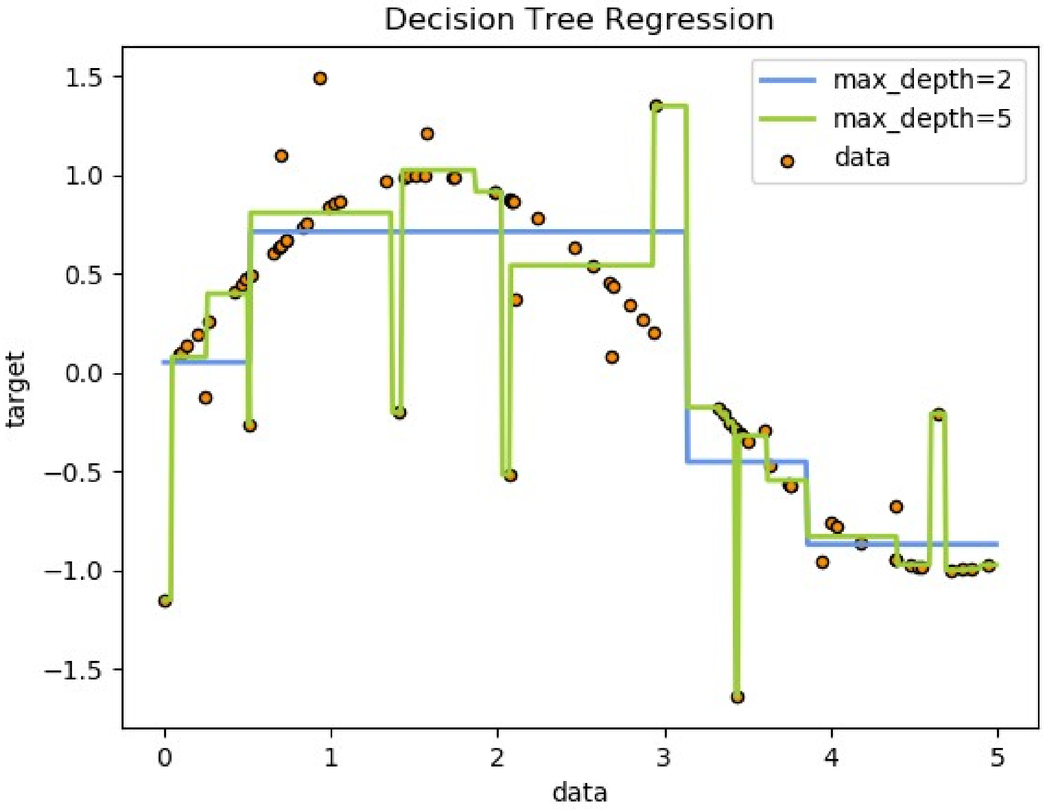
Decision tree

- Allows for non-linear decision boundary
- Extremely fast at classifying unknown records
- Easy to interpret for small-sized trees

Decision tree regression

- Decision trees applied to regression work similarly to the ones for classification
- In regression, the trees output a single number per leaf node instead of a label
- A tree can predict a non linear function

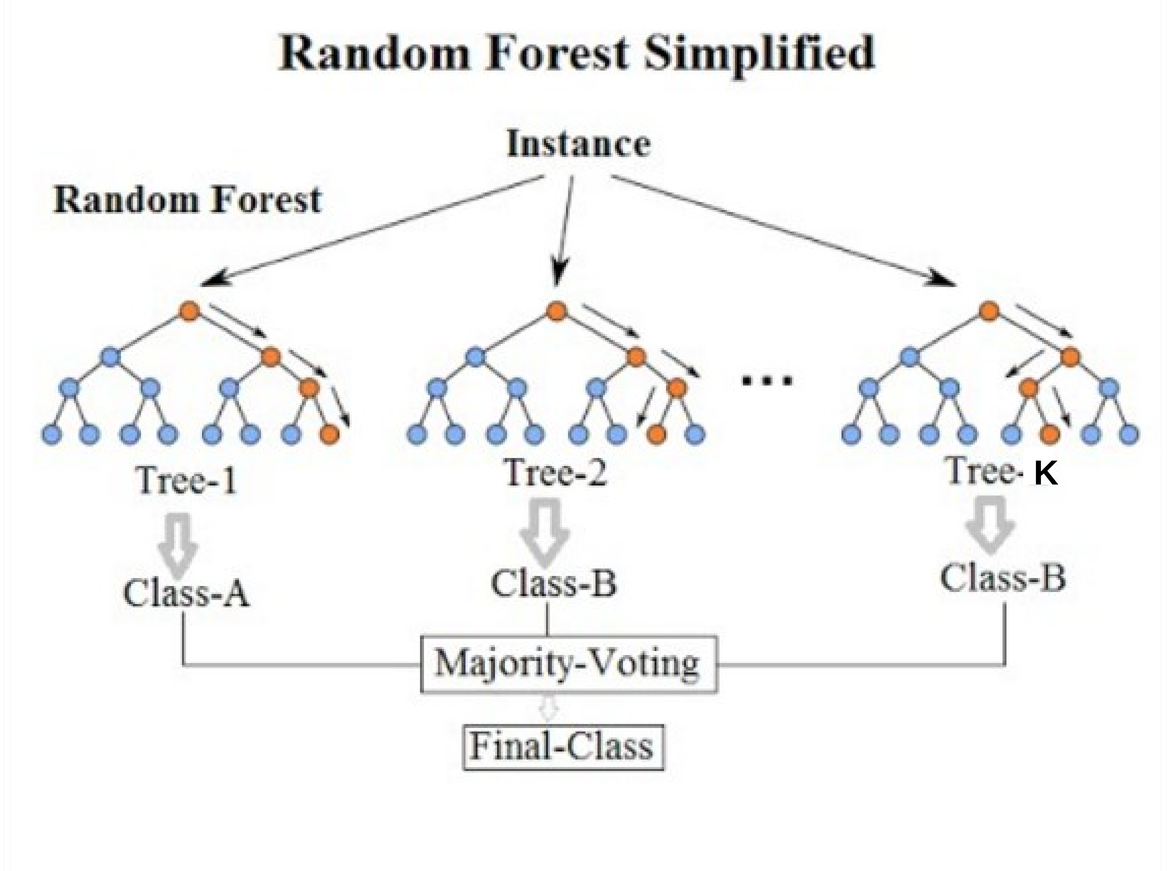
Decision tree regression



Random forest classifier

- Random forest is an ensemble classifier that consists of **many decision trees**
- Outputs the class according to the results of the trees
  - E.g., the mode of the output classes – majority voting
- For each tree of the **K trees** of the forest, choose a **subset of n' features** (n is the total number of features) and a **subset of m' training samples** (m is the total number of samples in the training data)

Random forest classifier



Random forest classifier

- How to select number of trees K?
  - Default number: more trees → more computational time
  - Validation curve (build trees until the validation error no longer decreases)
- How to select number of features n’ and samples m’?
  - Defaults: half of them, proportional to data,…
  - Validation curve
- Can also rank features

DT and RF in Python

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, class_weight=None, ccp_alpha=0.0*) [source]

sklearn.tree.DecisionTreeRegressor

class sklearn.tree.**DecisionTreeRegressor**(**, criterion='squared_error', 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, ccp_alpha=0.0*) [source]

sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

sklearn.ensemble.RandomForestRegressor

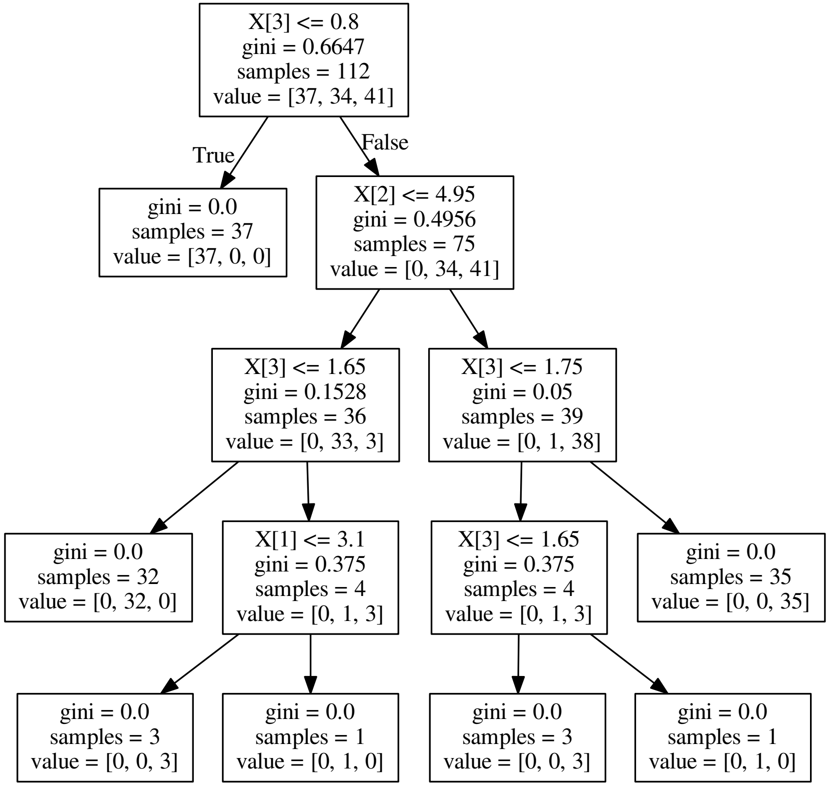
class sklearn.ensemble.**RandomForestRegressor**(*n_estimators=100, *, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=1.0, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None*) [source]

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Decision trees in Python

Once trained, we can export the tree in Graphviz format:

```
import graphviz
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
dot_data = tree.export_graphviz(clf,out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iris")
```



k-nearest neighbors (k-NN)

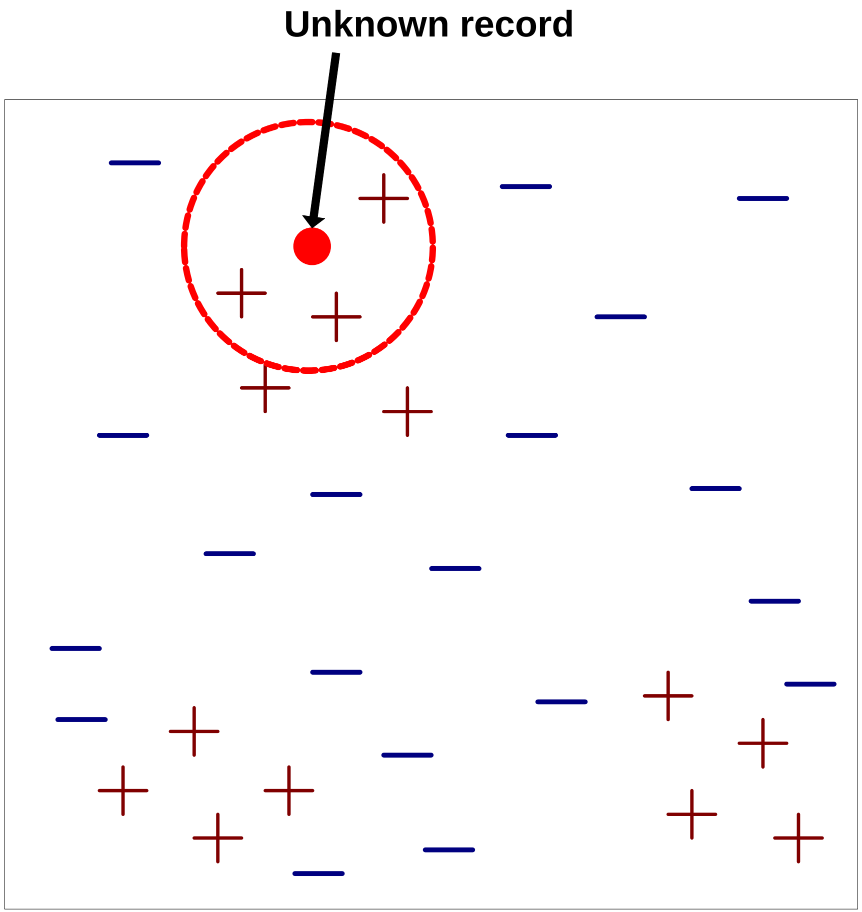
k-Nearest Neighbors: design choices

- Regressor and classifier
- **Datapoints** with numeric features
- **Label** values **numeric or categoric**
- **Model:** piecewise constant map - hypothesis space of k-NN depends on a labeled dataset (training set) D
- **No loss** function
  - Nothing is optimized
  - No training

k-Nearest Neighbors

- **Instance-based learning**: it does not construct a general internal model
- Simply stores instances of the training data
- Uses **k closest points** for performing classification/regression
  - k data points that have the k smallest distance to **x**

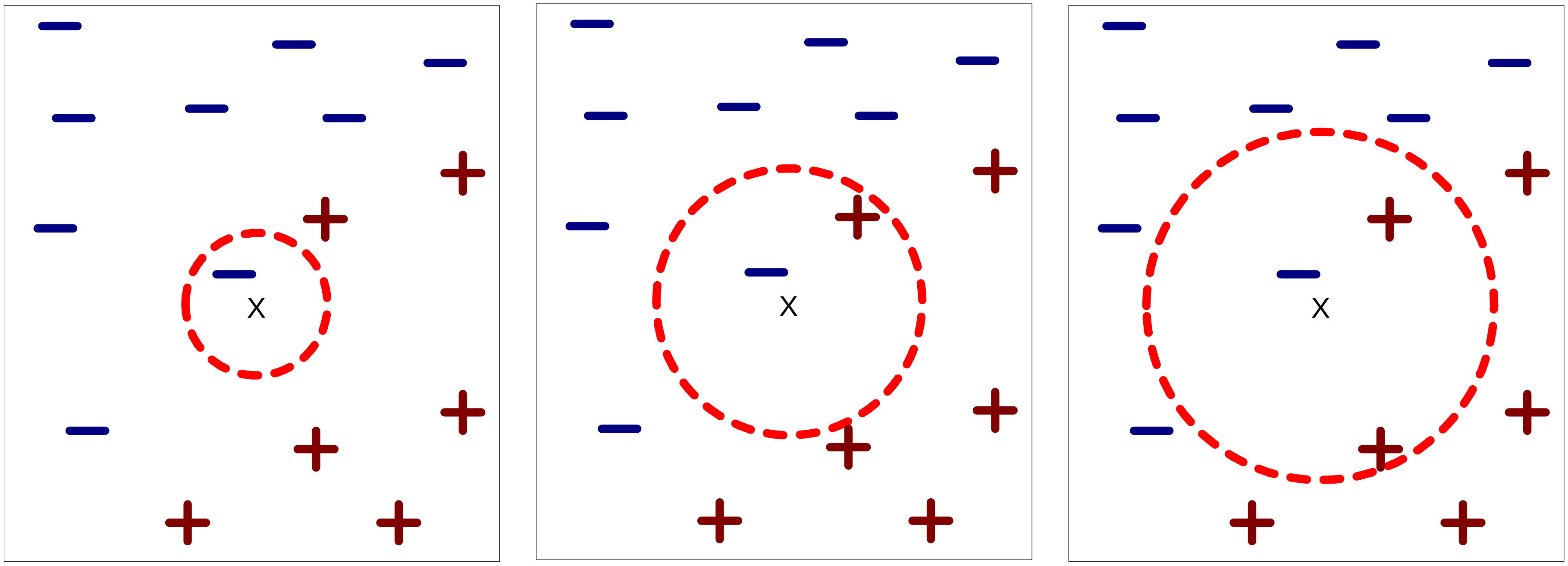
k-Nearest Neighbors classifier



- Requires
  - The set of stored records
  - Distance metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by majority vote)

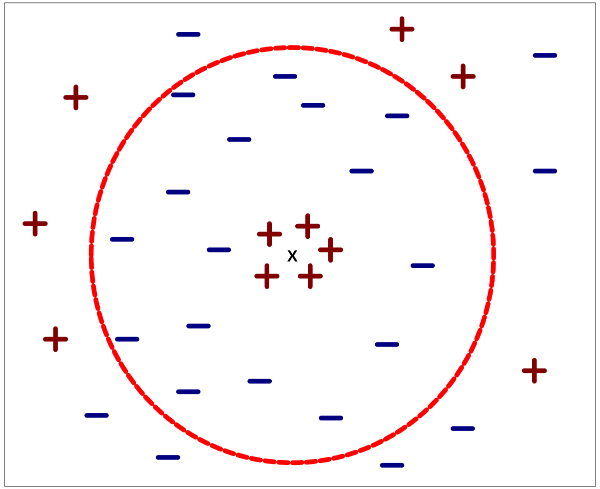
k in Nearest Neighbors



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

k in Nearest Neighbors

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



k-Nearest Neighbors

- Scaling issues
  - Attribute domain should be normalized/standardized to prevent distance measures from being dominated by one of the attributes
  - Example: height [1.5m to 2.0m] vs. income [$10K to $1M]
- If many features, curse of dimensionality

k-NN in Python

sklearn.neighbors.KNeighborsClassifier

class `sklearn.neighbors.`**`KNeighborsClassifier`**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*) [source]

sklearn.neighbors.KNeighborsRegressor

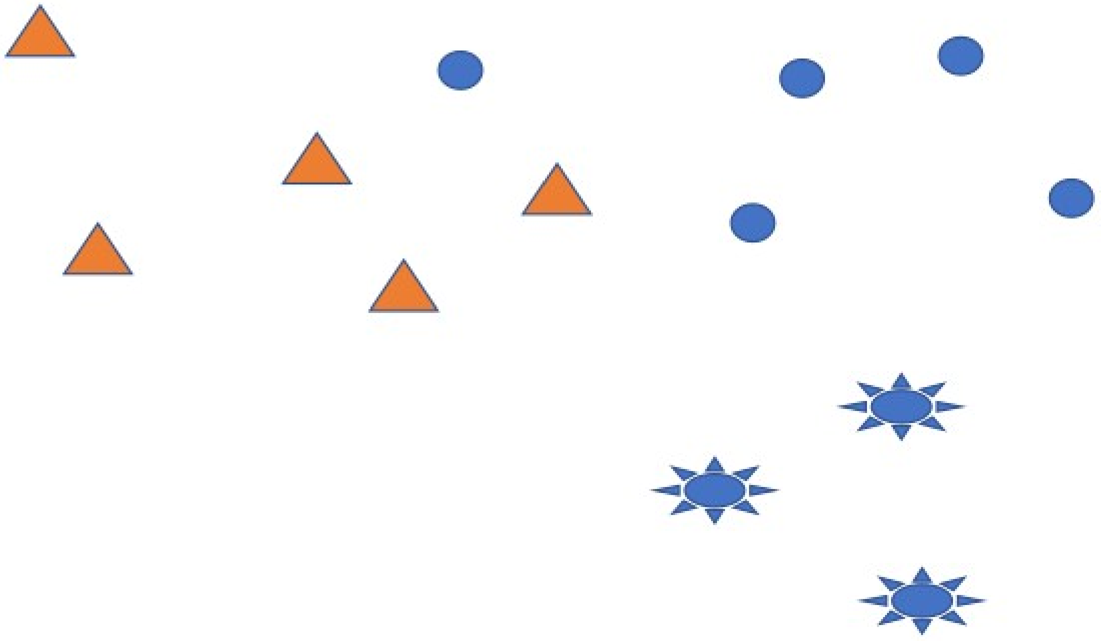
class `sklearn.neighbors.`**`KNeighborsRegressor`**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*) [source]

https://scikit-learn.org/stable/modules/neighbors.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

Multi-class and multi-label classification

https://scikit-learn.org/1.5/modules/multiclass.html

Multi-class classification



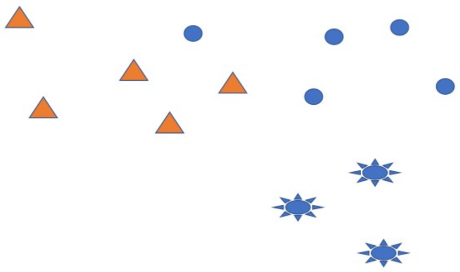
Multi-class classification

- One-vs-one
  - data points with label values ”1”, “2”, “3”
  - break into **3 binary classification problems, one for each pair o classes**
  - Problem 1: label values “1” vs “2”
  - Problem 2: label values “2” vs “3”
  - Problem 3: label values ”3” vs “1”
  - The class which received the most votes is selected

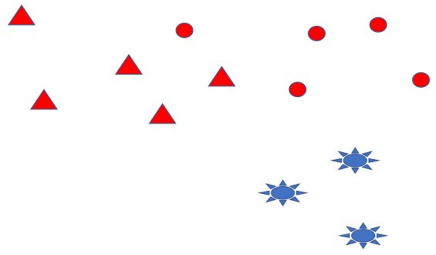
Multi-class classification

- One-vs-rest
  - data points with label values ”1”, “2”, “3”
  - break into **3 binary classification problems**
  - Problem 1: label values “1” ,“either 2 or 3”
  - Problem 2: label values “2”, “either 1 or 3”
  - Problem 3: label values ”3”, “either 2 or 3”

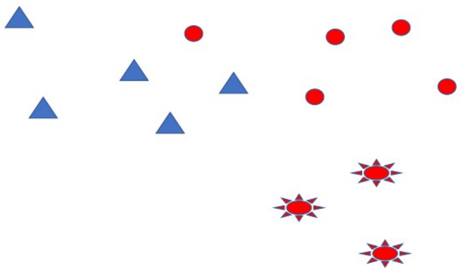
One-vs-Rest



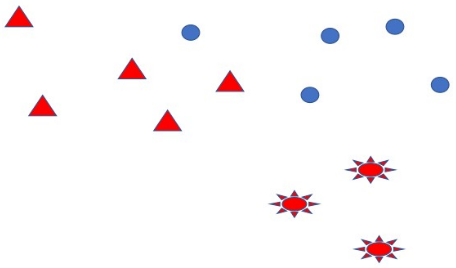
Sub-Problem 1 (red/blue)



Sub-Problem 2 (red/blue)



Sub-Problem 3 (red/blue)



Multi-class classification

- One-vs-rest
  - **one sub-problem for each label value k**
  - sub-problem k: “label = c or not”
  - learn hypothesis h_k for each sub-problem
  - predict c with highest confidence/probability $|h_k|$

$$\hat{y} = \underset{k \in \{1...K\}}{\operatorname{argmax}} f_k(x)$$

Multi-class logistic regression

- Specific **loss** functions for multi-class data
- 0/1 loss also works for > 2 label values (classes)
- How to encode confidence in predictions?
- **Soft-max (extending logistic loss to K classes)**:

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

https://en.wikipedia.org/wiki/Softmax_function

Multi-label classification

Label 1 = 1 or 0 if **car** present or not

Label 2 = 1 or 0 if **person** present or not

Label 3 = 1 or 0 if **tree** present or not

Label 4 = 1 or 0 if a **cat** present or not

Multi-label classification

- Naive approach
  - consider **each label separately**
  - solve binary/multi-class problem for each label
  - ignores correlations among different labels

Multi-label classification

- Multi-class approach
  - **Each combination of label values defines a category**
  - Obtain a multi-class problem with many classes
  - Possibly huge number of resulting categories

New class label	Label 1	Label 2	Label 3	Label 4
1	0	0	0	0
2	0	0	1	0’
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	1	0’
10	1	0	1	1
11	1	1	0	0
12	1	1	0	1
13	1	1	1	0
14	1	1	1	1
15	1	0	0	1
16	0	0	0	1

Multi-label classification

- Multi-task learning approach
  - **Each individual label results in separate learning task**
  - Use similarities between learning tasks
  - Similarities inform **regularization techniques**
  - **Combine the loss to learn together the two hypothesis**

Any questions?

Self-assessment quiz

- Write a **decision tree regressor** with **accuracy 1 on training** for the following training data. Show it as a flowchart. What is its output on the test data?

- Given training data, what would predict a **1-NN regressor** on test data, if using **euclidean distance**?

Train data

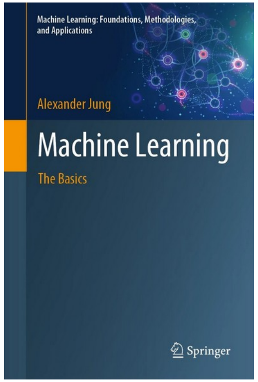
Feature 1	Feature 2	Feature 3	Label
10	5	1	7
7	0	1	-2
7	0	2	5
5	4	0	4
-1	0	0	0
10	1	1	6
2	1	1	2

Test data

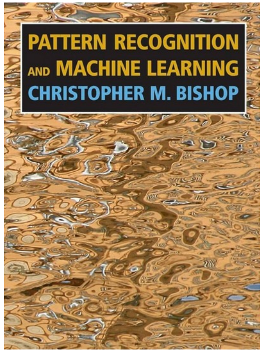
Feature 1	Feature 2	Feature 3	Label
4	3	1	3
5	1	0	7

References: readings

- Chapters 3-4-5



- Chapter 3-4-7



Slide acknowledgments

- Alexander Jung – Aalto University
- Tania Cerquitelli and Elena Maria Baralis – Politecnico di Torino
- Kilian Weinberger – Cornell Univesity