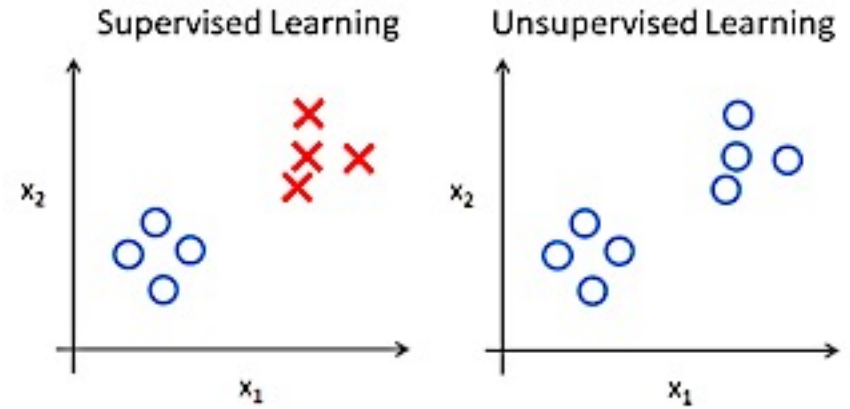
Data mining

Classification and Regression

Supervised learning

- By far the most common case.
- It consists of learning to map input data to known targets, given a set of examples (often annotated by humans).
- Main categories: classification and regression.



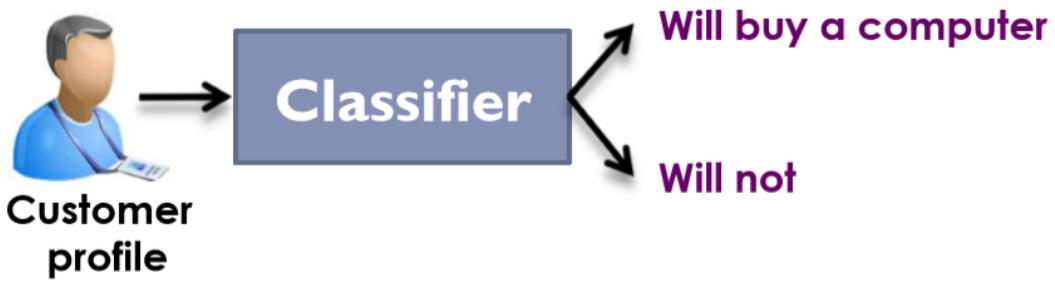
Classification vs. Regression

- Predicts categorical class labels (discrete or nominal).
- Example: What is the weather status tomorrow (rainy, cloudy, sunny)?

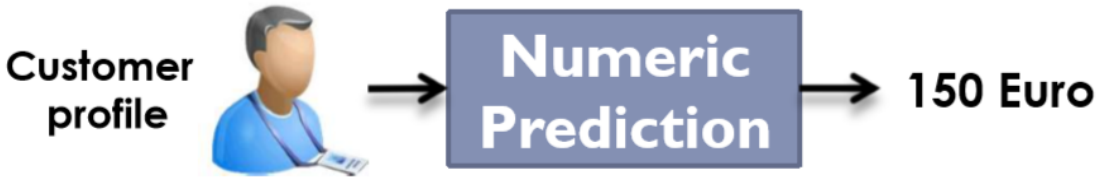
- Models continuous-valued functions.
- Example: What is the temperature going to be tomorrow?

Example

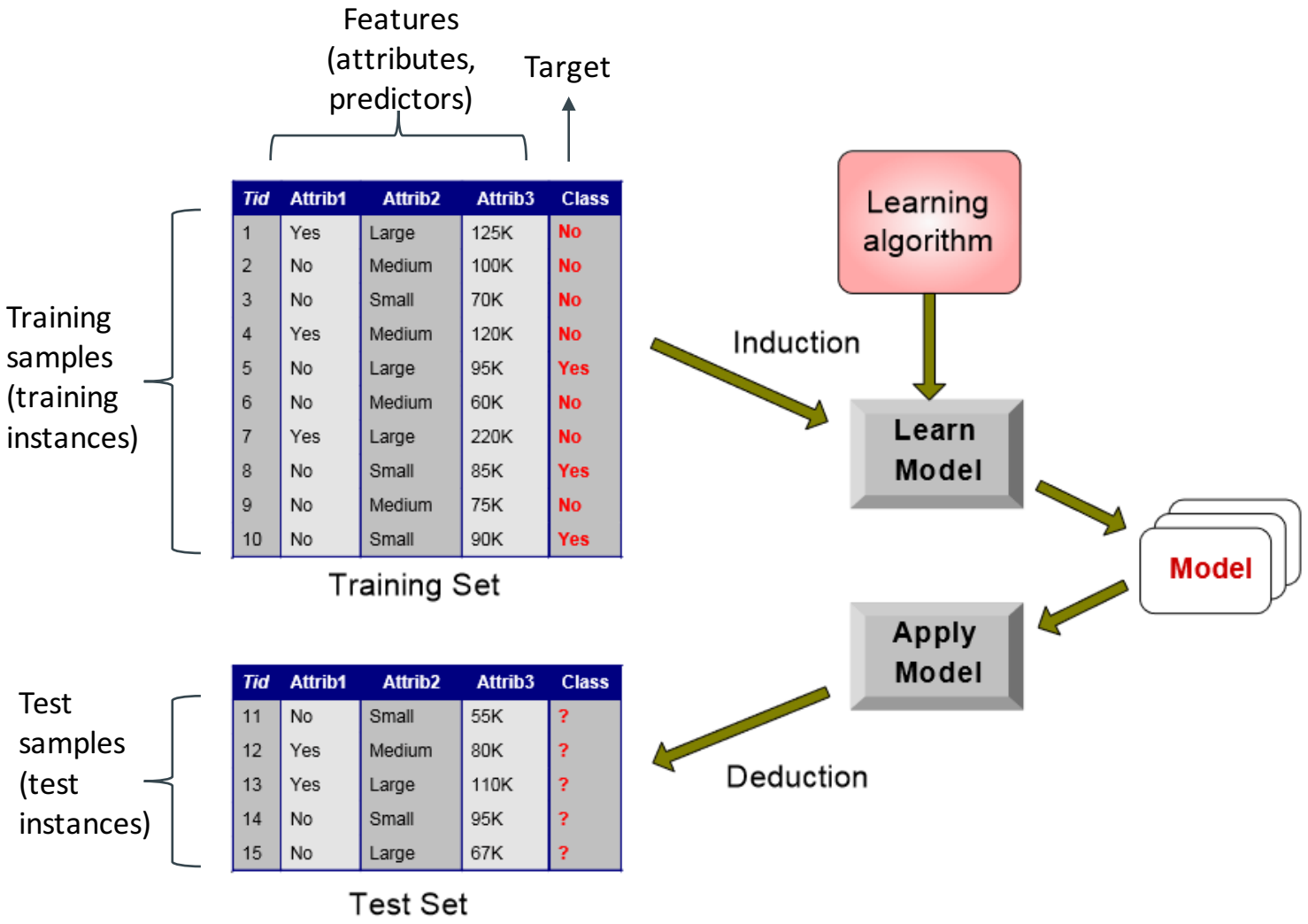
A marketing manager would like to know whether a given costumer will buy a product or not



A marketing manager would like to predict how much a given costumer will spend during a sale

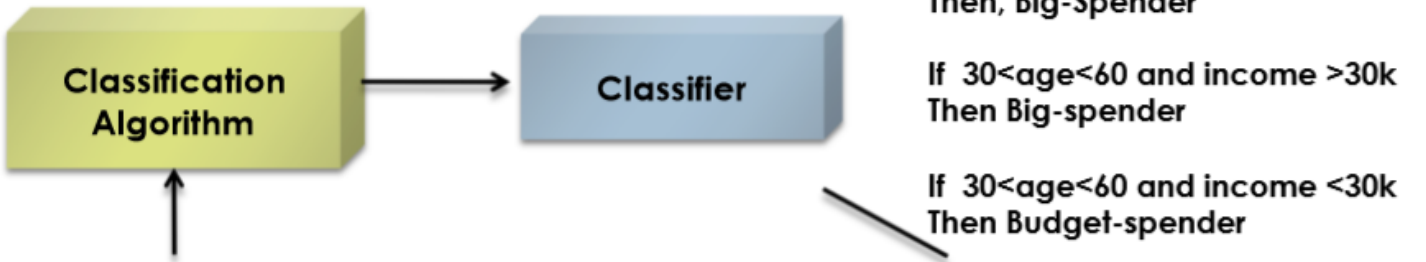


Classification



The model is represented as classification rules, decision trees, mathematical formulas, etc.

Training phase



Training data

Age	Income	Class label
27	28K	Budget-Spender
35	36K	Big-Spender
38	28K	Budget-Spender
65	45K	Budget-Spender
20	18k	Budget-Spender
75	40k	Budget-Spender
28	50k	Big-Spender
40	60k	Big-Spender
60	65k	Big-Spender

Evaluation and usage phases

1-Test the classifier

Test Data

Age	Income	Class label
27	28K	Budget-Spenders
25	36K	Big-Spenders
70	45K	Budget-Spenders
40	35k	Big-Spender

2-If acceptable accuracy

Unlabeled data

Age	Income
18	28K
37	40K
60	45K
40	36k

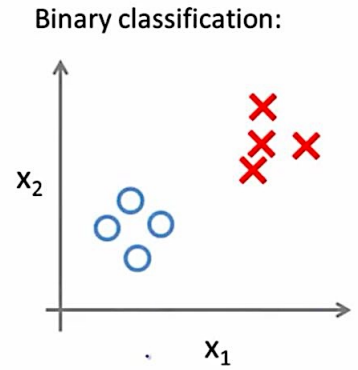
Classified data

Age	Income	Class label
18	28K	Budget-Spenders
37	40K	Big-Spenders
60	45K	Budget-Spenders
40	36k	Budget-Spenders

Binary vs. Multiclass classification

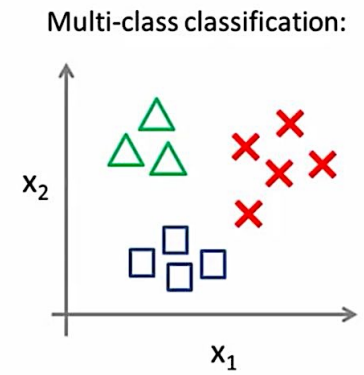
1 Binary classification

Classification tasks with only two classes, typically denoted by {+,−}, {+1,−1}, or {Pos, Neg}.
Example: email spam detection, (pos/neg) sentiment analysis.



2 Multiclass classification

Classification tasks with more than two classes.
Example: email topic detection, (pos/null/neg) sentiment analysis.



Classification algorithms

- Decision tree
- Bayesian methods (Naïve Bayes and Bayesian networks)
- K-Nearest Neighbors (KNN)
- Logistic regression
- Support Vector Machines (SVM)
- Artificial neural networks and Deep learning
- Ensemble learning
- And many more…

Decision Tree

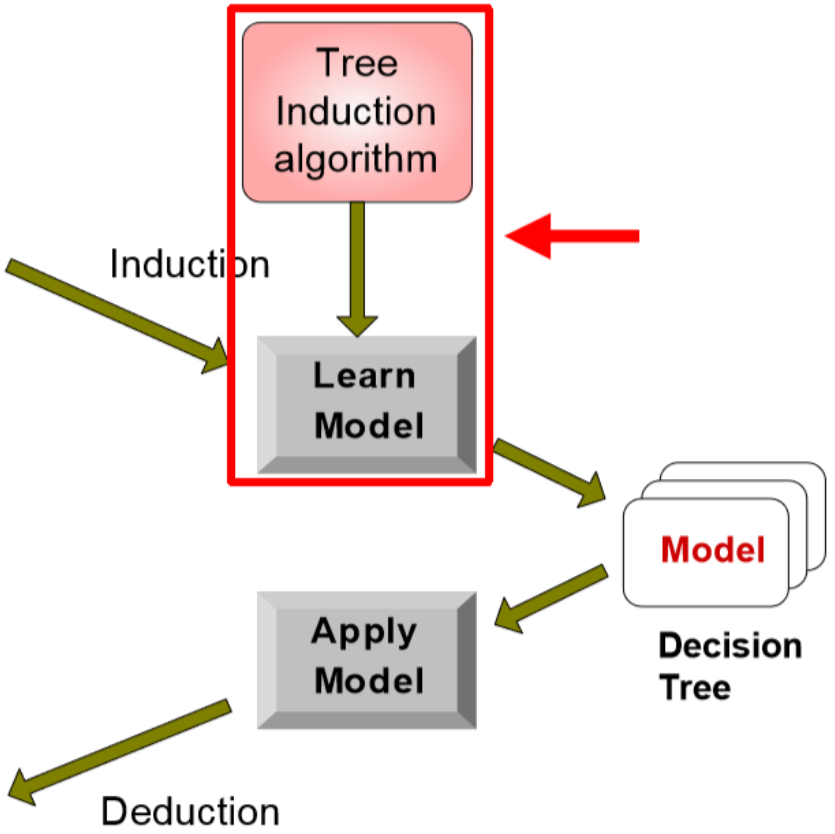
Classification using decision trees

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

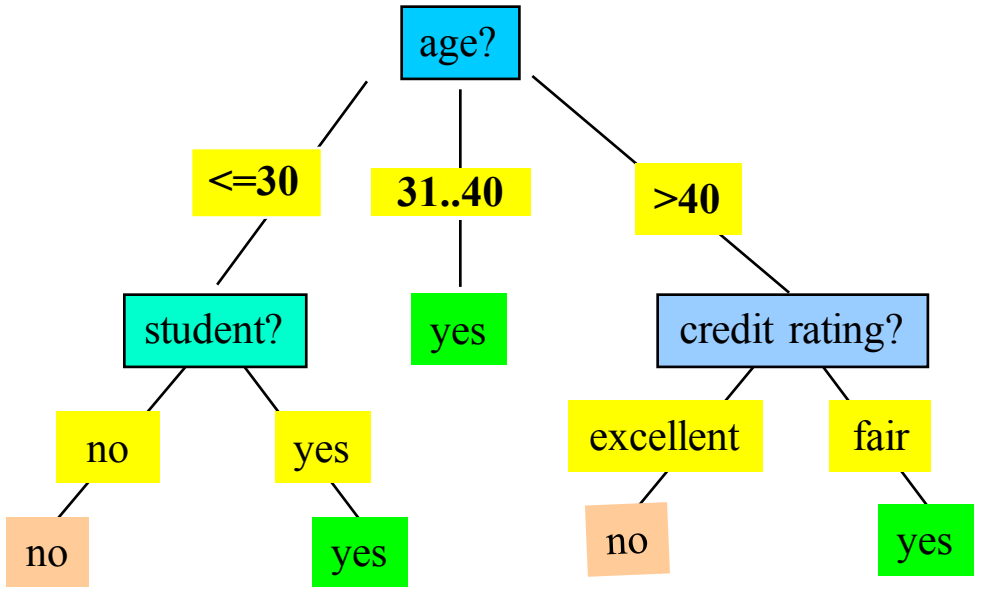
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



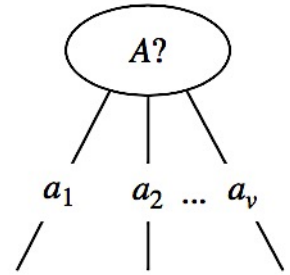
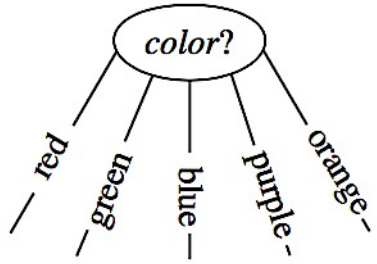
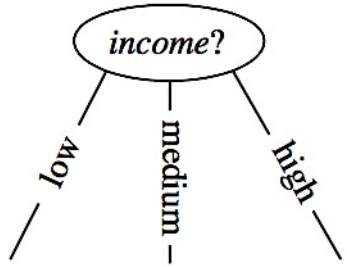
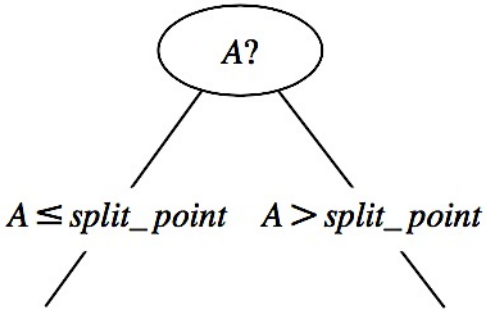
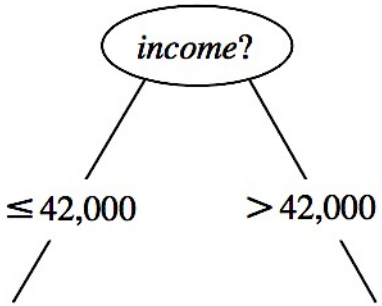
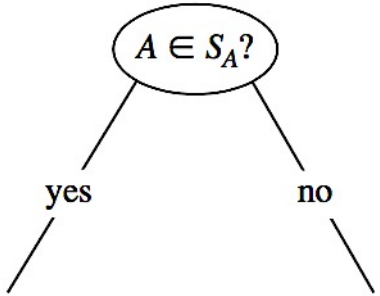
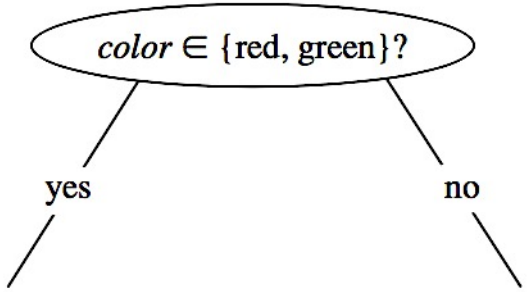
Definition

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



- Internal nodes (non-leaf nodes) denote a test on an attribute.
- Branches represent outcomes of tests.
- Leaf nodes (terminal nodes) hold class labels.
- Root node is the topmost node.

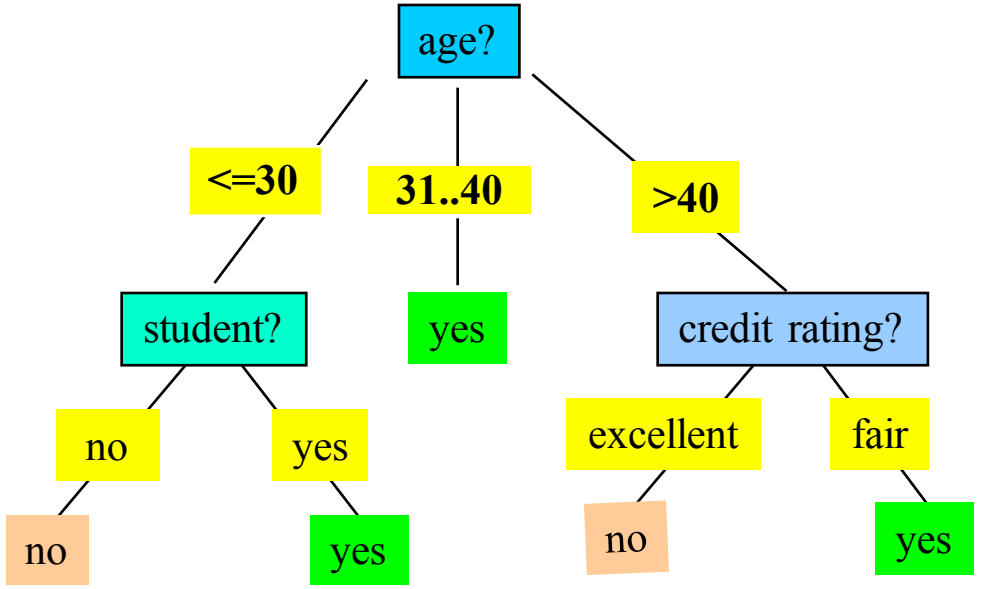
Definition

	Partitioning scenarios	Examples	
A is discrete-valued			
A is continuous-valued			
A is discrete-valued but we want a binary tree			

Usage

- How are decision trees used for classification?
  - A path is traced from the root to a leaf node which holds the prediction for that tuple
  - The attributes of a tuple are tested against the decision tree.

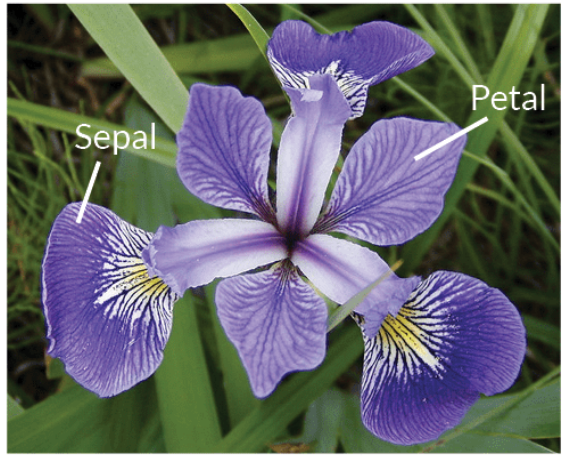
RID	age	income	student	credit-rating	Class
1	youth	high	no	fair	?



Python example


```
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target
print('X shape: ', X.shape)
print('y shape: ', y.shape)
```

```
X shape:  (150, 4)
y shape:  (150,)
```



Iris Versicolor

Iris Setosa

Iris Virginica

Python example

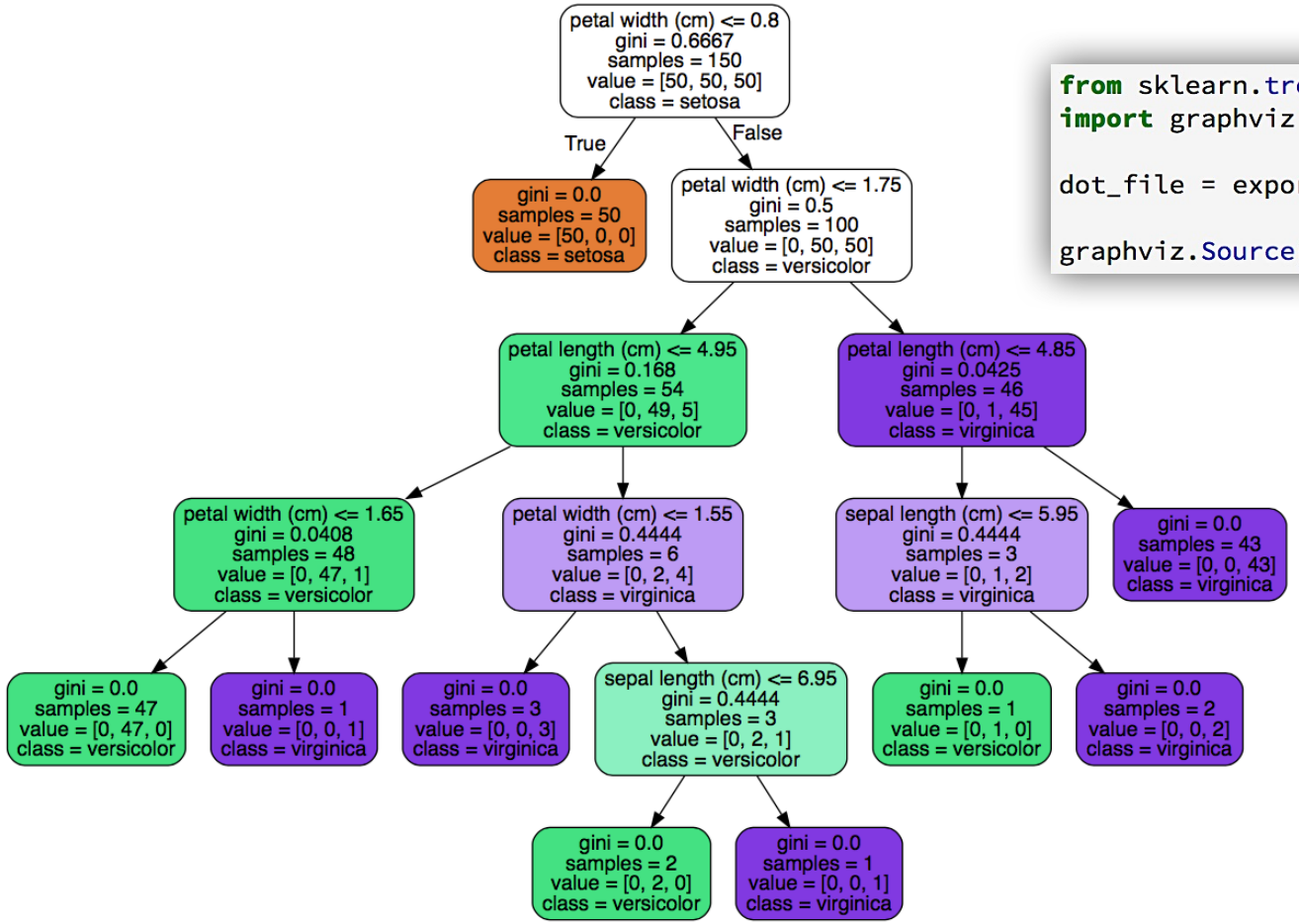

```
from sklearn.tree import DecisionTreeClassifier

# train
clf = DecisionTreeClassifier()
clf.fit(X, y)

# usage
X_test = [[5,3,1,0.5], [6,4,5,2]]
clf.predict(X_test)
```

```
array([0, 2])
```

```
clf.predict_proba(X_test)
```

```
array([[1., 0., 0.],
       [0., 0., 1.]])
```

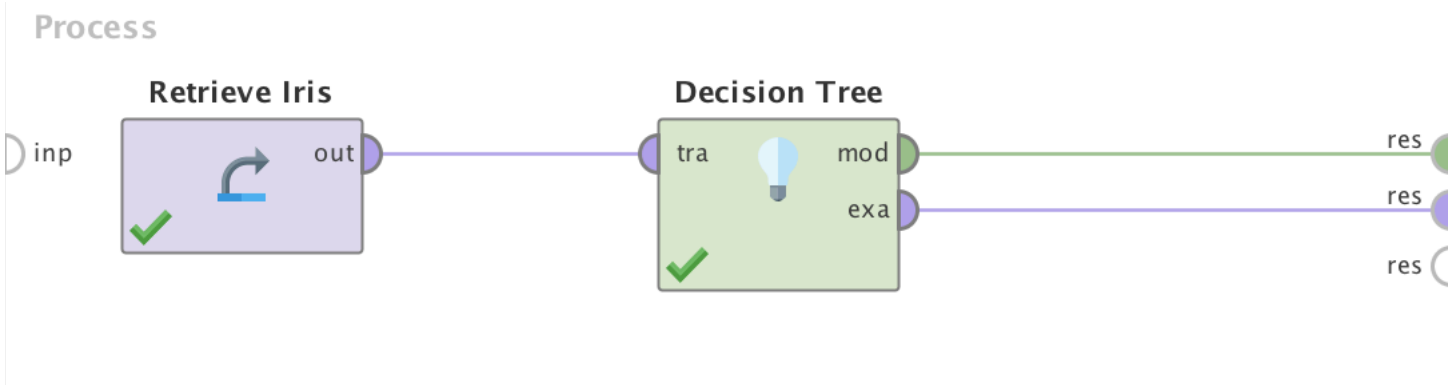
Python example




```
from sklearn.tree import export_graphviz
import graphviz

dot_file = export_graphviz(clf, out_file=None, feature_names=iris.feature_names,
                           class_names=iris.target_names, filled=True, rounded=True)
graphviz.Source(dot_file)
```

Don’t forget to install graphviz package in your system, not just the Python package.

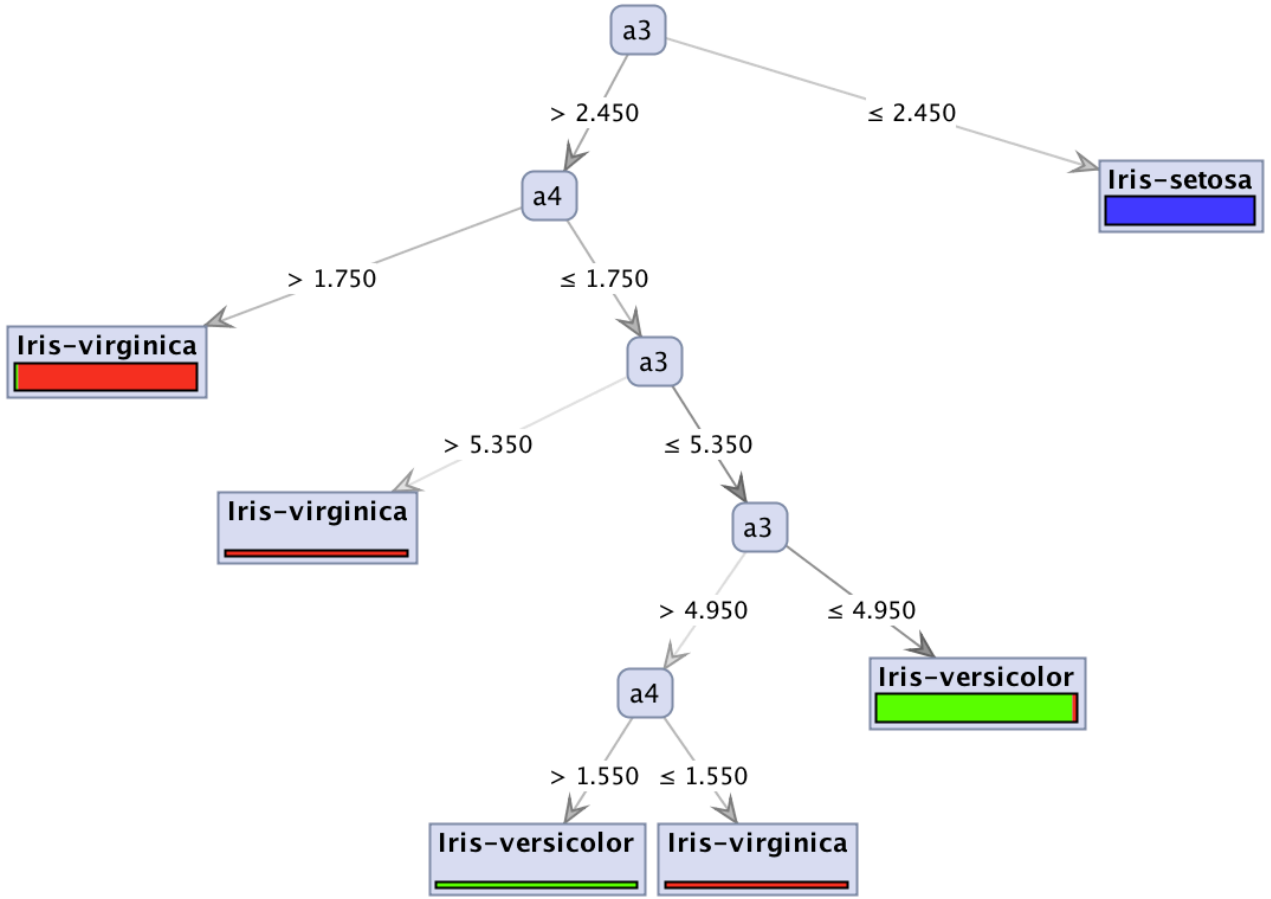
RapidMiner example



ExampleSet (150 examples, 2 special attributes, 4 regular attributes)

Row No.	id	label	a1	a2	a3	a4
1	id_1	Iris-setosa	5.100	3.500	1.400	0.200
2	id_2	Iris-setosa	4.900	3	1.400	0.200
3	id_3	Iris-setosa	4.700	3.200	1.300	0.200
4	id_4	Iris-setosa	4.600	3.100	1.500	0.200
5	id_5	Iris-setosa	5	3.600	1.400	0.200
6	id_6	Iris-setosa	5.400	3.900	1.700	0.400
7	id_7	Iris-setosa	4.600	3.400	1.400	0.300
8	id_8	Iris-setosa	5	3.400	1.500	0.200

RapidMiner example



Learning

- There are many specific decision-tree algorithms, such as:
  - ID3 (Iterative Dichotomiser 3)
  - C4.5 (successor of ID3)
  - CART (Classification And Regression Tree)
  - CHAID (CHi-squared Automatic Interaction Detector)
  - MARS (extends decision trees to handle numerical data better)

Learning

- Basic algorithm adopted by ID3, C4.5 and CART is a greedy algorithm.
- Tree is constructed in a top-down recursive divide-and-conquer manner:
  - At start, all the training tuples are at the root.
  - Attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain).
  - Tuples are partitioned recursively based on selected attributes.
  - When an stopping condition is met, one leaf is constructed.

Stopping conditions

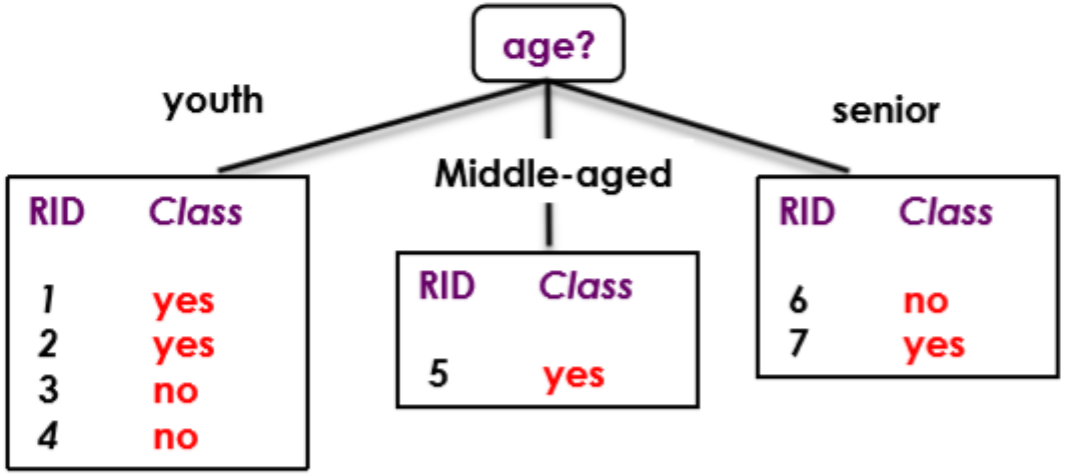
1. All samples for a given node belong to the same class.
2. There are no remaining attributes for further partitioning – majority voting is employed.
3. There are no samples left – majority voting on the parent's samples is employed.

Decision tree learning algorithm

(1) create a node N;
(2) **if** tuples in D are all of the same class, C, **then**
(3) return N as a leaf node labeled with the class C;
(4) **if** *attribute_list* is empty **then**
(5) return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply **Attribute_selection_method**(D, *attribute_list*) to **find** the "best" *splitting_criterion*;
(7) label node N with *splitting_criterion*;
(8) **if** *splitting_attribute* is discrete-valued **and**
multiway splits allowed **then** // not restricted to binary trees
(9) *attribute_list* ← *attribute_list* − *splitting_attribute*; // remove *splitting_attribute*
(10) **for each** outcome j of *splitting_criterion*
// partition the tuples and grow subtrees for each partition
(11) let D_j be the set of data tuples in D satisfying outcome j; // a partition
(12) **if** D_j is empty **then**
(13) attach a leaf labeled with the majority class in D to node N;
(14) **else** attach the node returned by **Generate_decision_tree**(D_j, *attribute_list*) to node N;
endfor
(15) return N;

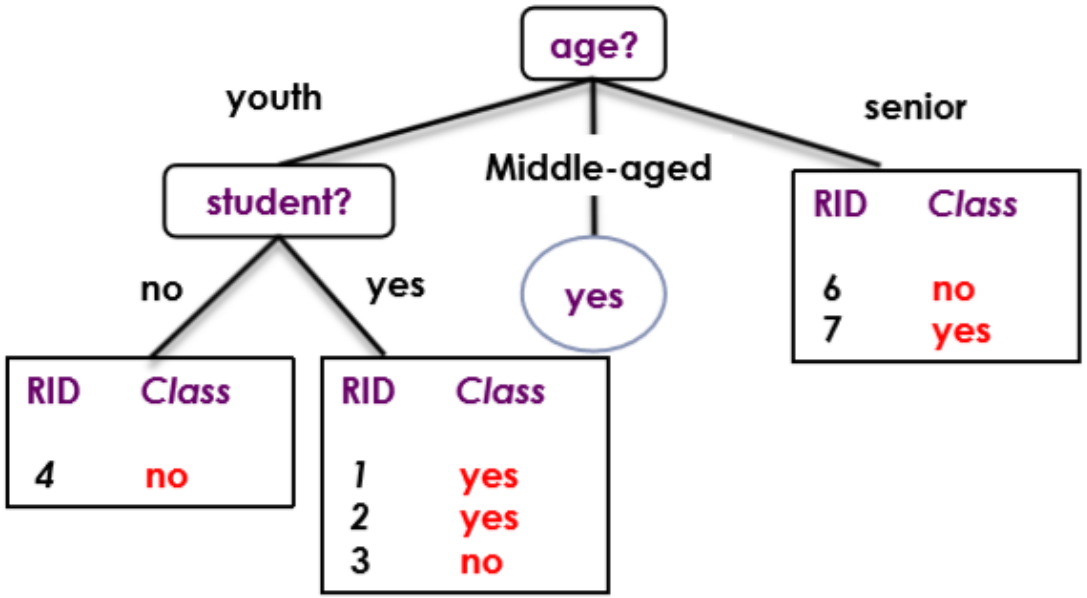
Example

RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes



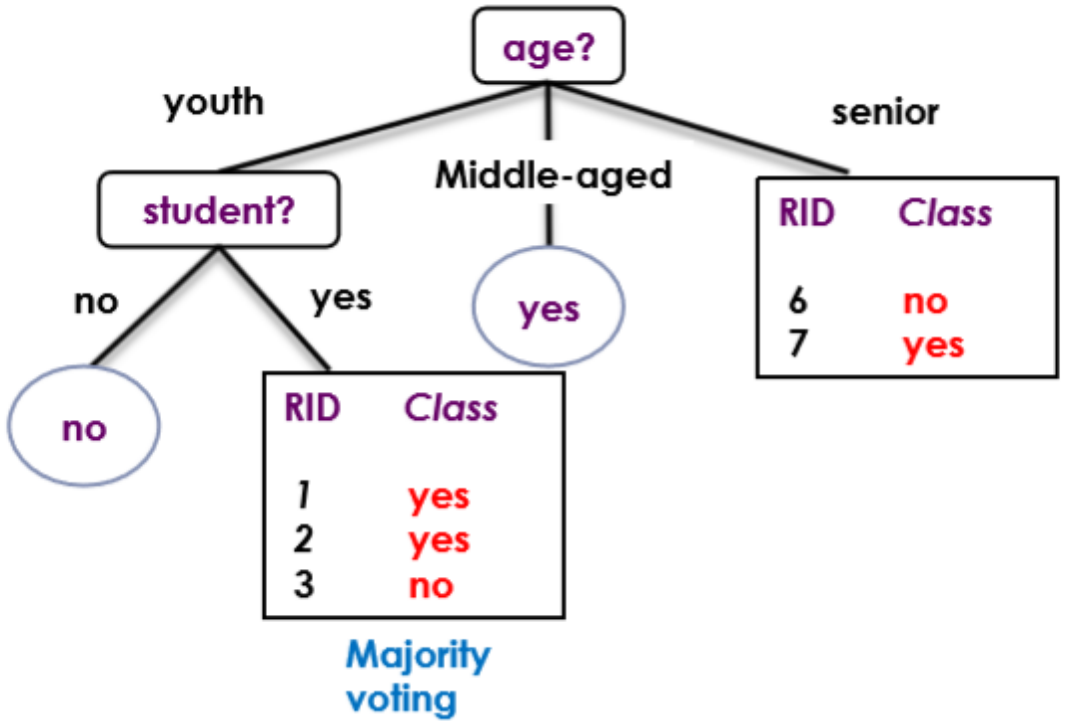
Example

RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes



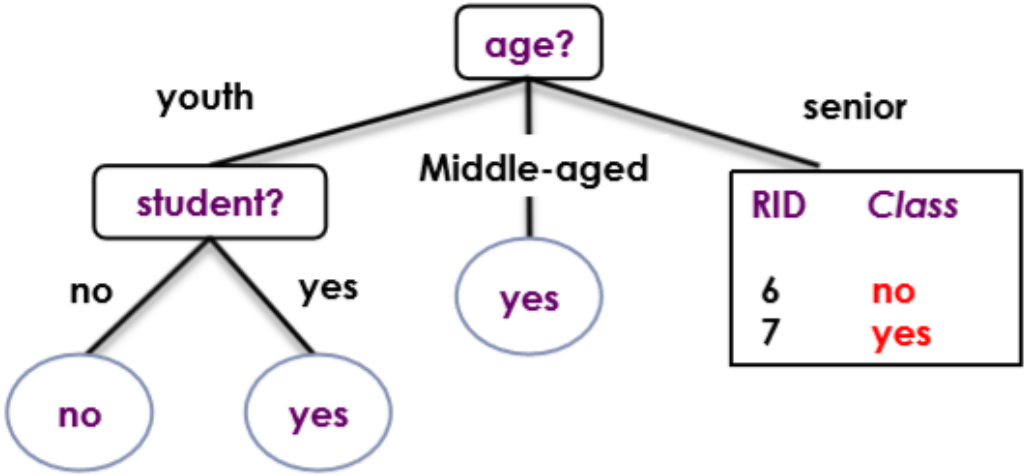
Example

RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes



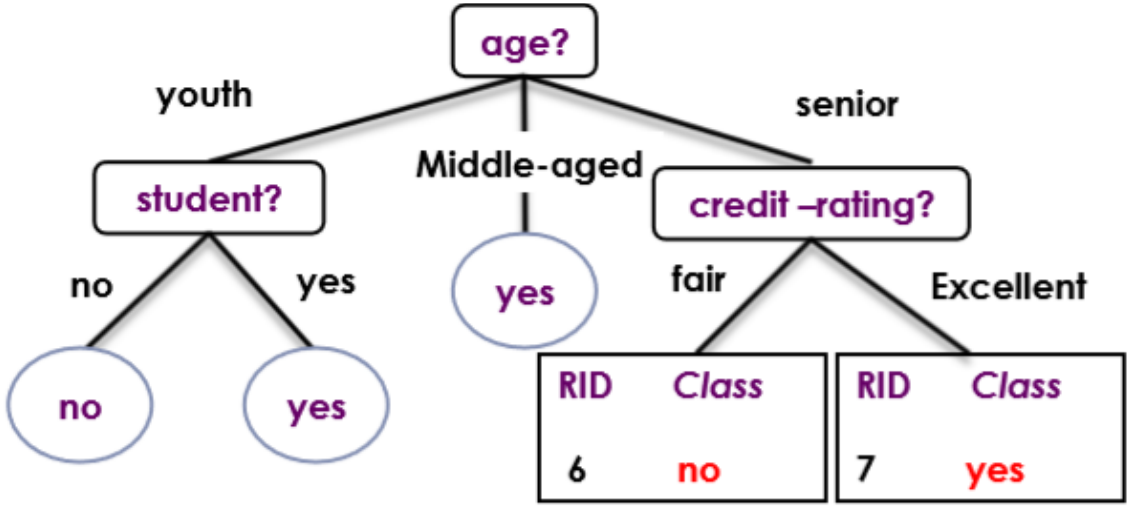
Example

RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes



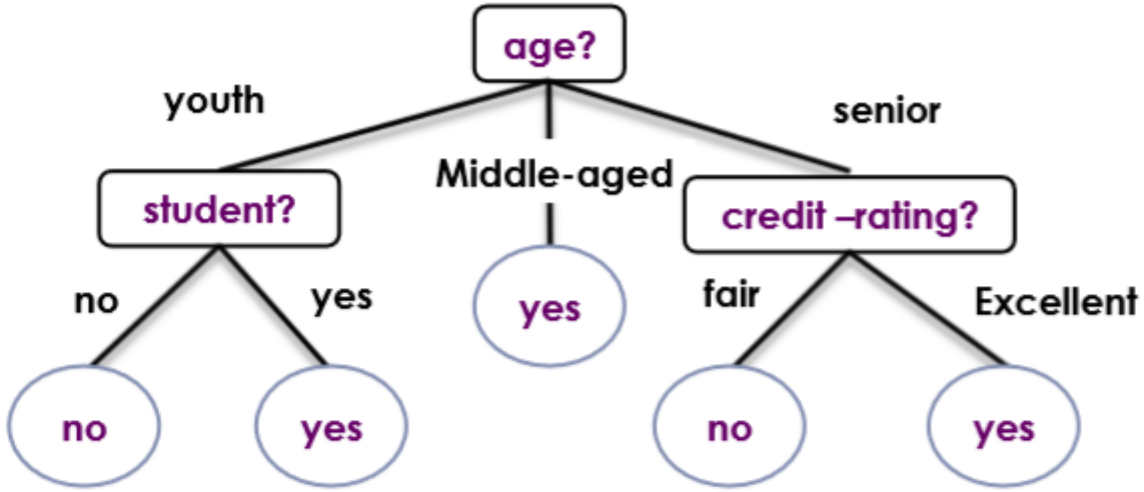
Example

RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes



Example

RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes

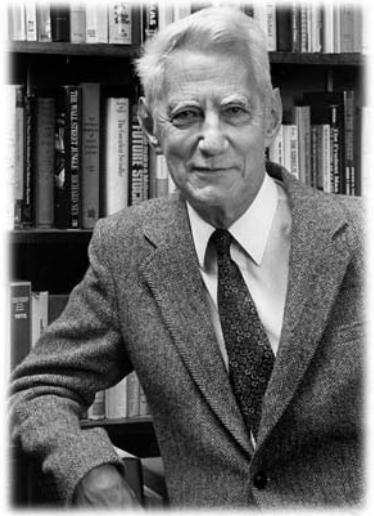


Attribute selection measures

- An attribute selection measure (splitting rule) is a heuristic for selecting the splitting criterion that “best” separates a given data partition D. Ideally
  - Each resulting partition would be pure (containing tuples that all belong to the same class).
  - Provide ranking for each attribute and the attribute with highest score is chosen.
  - It can also determine a split point for continuous-valued attributes or a splitting subset for binary trees.
- We’ll review Information Gain, Gain Ratio, and Gini Index.

Information Gain

1916-2001

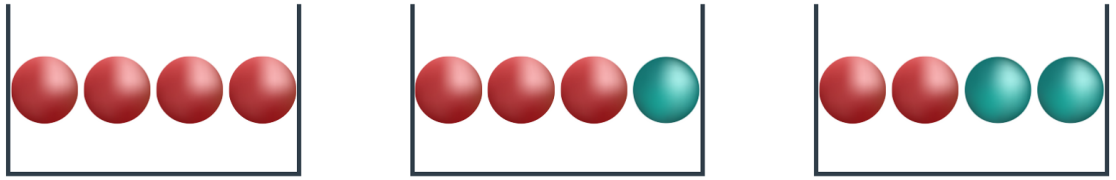


- The concept of information entropy was introduced by Claude Shannon in his 1948 paper "A Mathematical Theory of Communication".
- Information entropy is a concept from information theory. It tells how much information there is in an event. In general, the more uncertain or random the event is, the more information it will contain.

$$X \in \{x_1, x_2, \dots, x_m\}$$
$$\{p_1, p_2, \dots, p_m\}$$

$$H(X) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

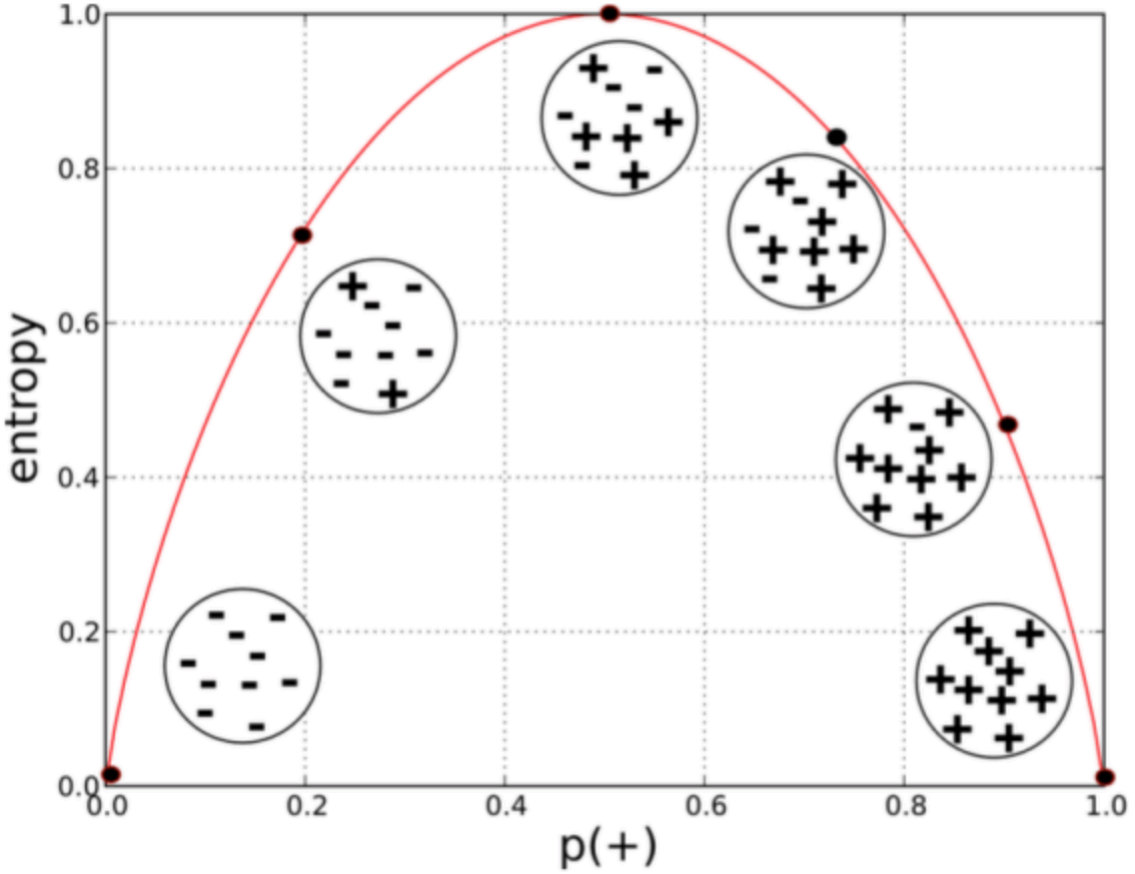
Information Gain



Bucket 1
Entropy: 0

Bucket 2
Entropy: 0.81125

Bucket 3
Entropy: 1



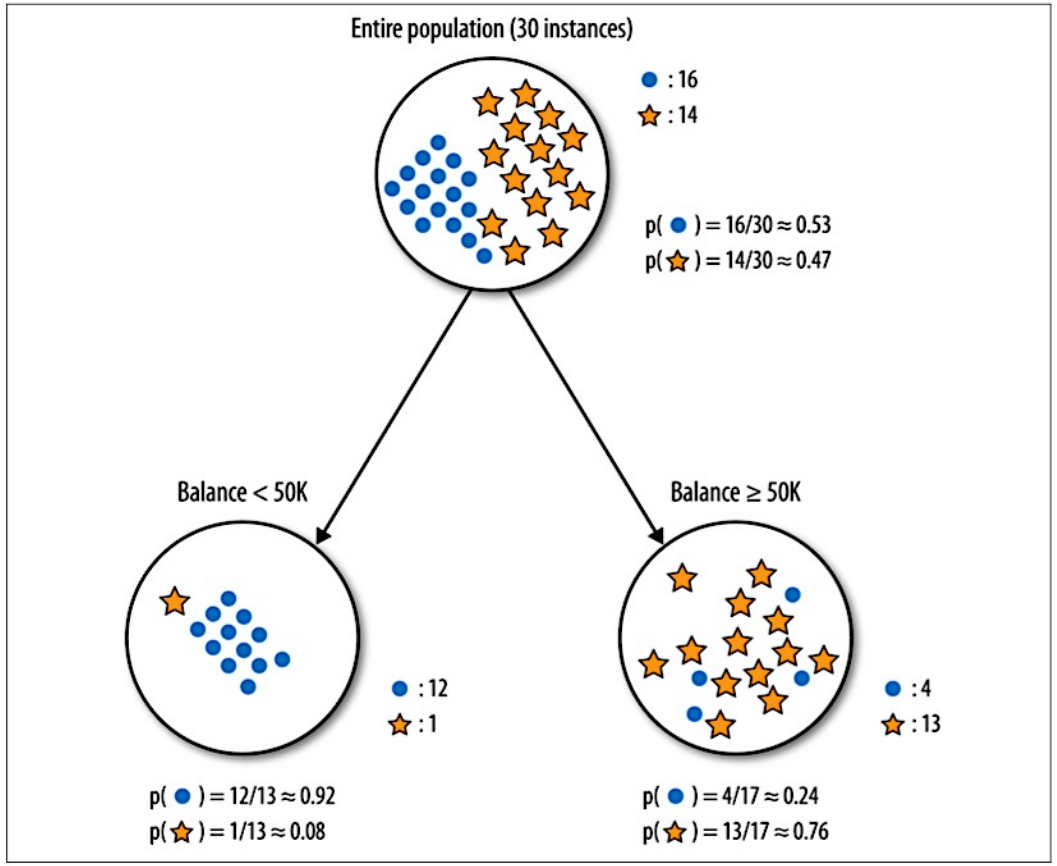
https://medium.com/udacity/shannon-entropy-information-gain-and-picking-balls-from-buckets-5810d35d54b4

Source: Data Science for Business

Information Gain

- High Entropy
  - X is from a uniform like distribution (flat histogram)
  - Values sampled from it are less predictable

- Low Entropy
  - X is from a varied distribution (histogram has many lows and highs)
  - Values sampled from it are more predictable



Source: Data Science for Business

Information Gain: step-I

- Step1: Compute entropy of D.
  - The average amount of information needed to identify the class label of a tuple in D.

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

m: the number of classes

p_i: the probability that an arbitrary tuple in D belongs to class C_i estimated by: $|C_{i,D}|/|D|$ (proportion of tuples of each class)

A log function to the base 2 is used because the information is encoded in bits

Information Gain: step-I

RID	age	income	student	credit-rating	class:buy_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle-aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle-aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle-aged	medium	no	excellent	yes
13	middle-aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

9 tuples in class yes
5 tuples in class no

$$Info(D) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940 \text{ bits}$$

Information Gain: step-II

- Step2: For each attribute, compute the weighted entropy after partioning using that attribute.
  - Suppose that we want to partition the tuples in D on some attribute *A* with values $\{a_1, a_2, \ldots, a_v\}$
  - D is split into *v* partitions $\{D_1, D_2, \ldots, D_v\}$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$|D_i|/|D|$: the weight of the j[th] partition.

Information Gain: step-II

RID	age	income	student	credit-rating	class:buy_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle-aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle-aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle-aged	medium	no	excellent	yes
13	middle-aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) + \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) = 0.694 \text{ bits.}$$

Information Gain: step-III

- Step3: Compute information gain.

$$Gain(A) = Info(D) - Info_A(D)$$

- The attribute with the highest information gain is chosen as the splitting attribute.

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$
$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Gain Ratio

- Information gain measure is biased towards attributes with a large number of values.
  - For example, it prefers features like student_number which is not a useful feature in classification at all.
- C4.5 (a successor of ID3) uses gain ratio to overcome this problem (normalization of information gain).

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

$$\text{GainRatio(A)} = \frac{Gain(A)}{SplitInfo(A)}$$

Gain Ratio: Example

RID	age	income	student	credit-rating	class:buy_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle-aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle-aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle-aged	medium	no	excellent	yes
13	middle-aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

$$Gain(income) = 0.029 \quad \Rightarrow \quad GainRatio(income) = 0.029/1.557 = 0.019$$

Gini Index

- The Gini index is used in CART and measures the impurity of a dataset.
- If a data set *D* contains examples from *n* classes, gini(D) is defined as:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

where p_j is the relative frequency of class *j* in *D*.

- If a data set *D* is split on A into two subsets D_1 and D_2, the $gini_A(D)$ is defined as:

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

- Reduction in impurity:

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

Gini Index: Example

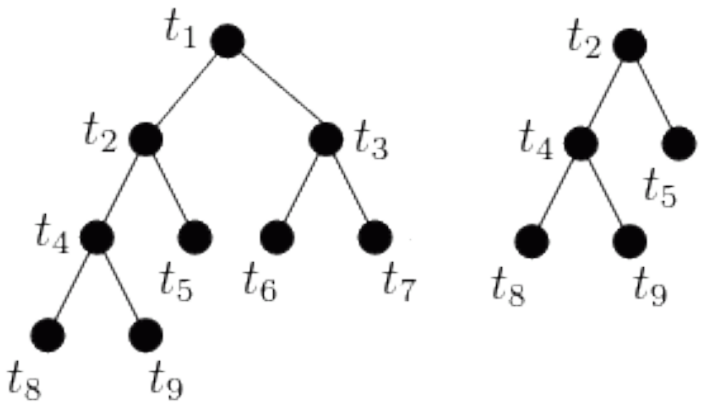
RID	age	income	student	credit-rating	class:buy_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle-aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle-aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle-aged	medium	no	excellent	yes
13	middle-aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$
\begin{aligned}
&Gini_{income \in \{low,medium\}}(D) \\
&\quad = \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\
&\quad = \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) \\
&\quad = 0.443 \\
&\quad = Gini_{income \in \{high\}}(D).
\end{aligned}
$$

$Gini_{\{low, high\}}$ is 0.458; $Gini_{\{medium, high\}}$ is 0.450. Thus, split on the {low, medium} (and {high}) since it has the lowest Gini index

Pruning

- To make the model simpler to reduce the chance of overfitting.
- The principle of *Occam's razor*: given two explanations for something, the explanation most likely to be correct is the simplest one.



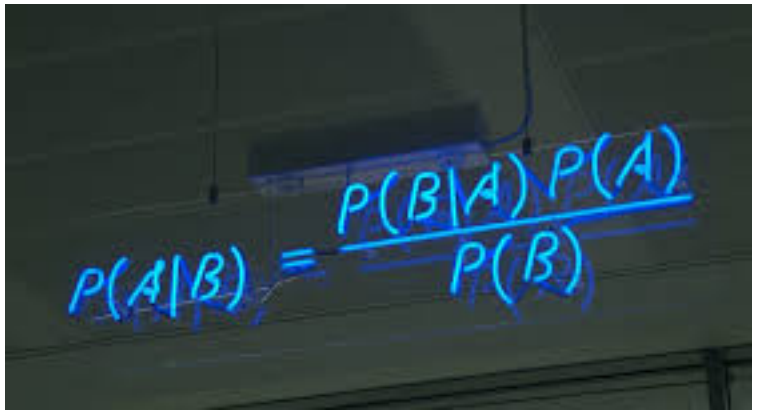
Pre-pruning vs. Post-pruning

- Pre-Pruning
  - Stop the algorithm before it becomes a fully-grown tree.
- Post-pruning
  - Grow decision tree to its entirety.
  - Trim the nodes of the decision tree in a bottom-up fashion.
  - If generalization error improves after trimming, replace sub-tree by a leaf node.
  - Class label of leaf node is determined from majority class of instances in the sub-tree.

Bayesian Method

Bayesian method

- A probabilistic method
  - Predicts class membership probabilities
- Naïve Bayes classifier
  - Based on Bayes' theorem
  - Acceptable accuracy and speed when applied to large databases
  - Incremental



Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}$$

- Proof is simple
- Let $\mathbf{X}$ be a data sample with unknown class label
- Let *H* be a hypothesis that $\mathbf{X}$ belongs to class C
- Classification is to determine P(H|$\mathbf{X}$) (posteriori probability)
- P(H) (prior probability): the initial probability
  - E.g., $\mathbf{X}$ will buy computer, regardless of age, income, etc.
- P($\mathbf{X}$): probability that sample data is observed
- P($\mathbf{X}$|H) (likelihood): the probability of observing the sample $\mathbf{X}$, given that the hypothesis holds
  - E.g., Given that $\mathbf{X}$ will buy computer, the prob. that $\mathbf{X}$ is 31..40, medium income, etc.

Classification based on Bayes' theorem

- Predicts $\mathbf{X}$ belongs to C_i if the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|\mathbf{X})$ for all classes (maximum posteriori)

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since $P(\mathbf{X})$ is constant for all classes,
  - $\underset{i}{\operatorname{argmax}} P(C_i|\boldsymbol{X}) = \underset{i}{\operatorname{argmax}} P(\boldsymbol{X}|C_i)P(C_i)$

Example I: Probability of smokers get cancer

Smoke	Cancer
Y	N
Y	Y
N	N
N	Y

S \ C	Y	N
Y	200	1800
N	100	2200

$$P(\text{cancer}) = \frac{300}{4300}$$

$$P(\text{smoke} \mid \text{cancer}) = \frac{200}{300}$$

$$P(\text{cancer} = \text{yes} \mid \text{smoke} = \text{yes}) = \frac{P(\text{cancer})\,P(\text{smoke} \mid \text{cancer})}{P(\text{smoke})} = \frac{\frac{300}{4300} \times \frac{200}{300}}{a} = \frac{0.0465}{a}$$

$$P(\overline{\text{cancer}} \mid \text{smoke}) = 1 - P(\text{cancer} \mid \text{smoke}) = \frac{P(\overline{\text{cancer}})\,P(\text{smoke} \mid \overline{\text{cancer}})}{P(\text{smoke})} = \frac{\frac{4000}{4300} \times \frac{1800}{4000}}{a} = \frac{0.0419}{a}$$

If you need to compute the exact probabilities: $a = 0.0465 + 0.0419 = 0.0884$

Example II: Healthy/Patient classification

Age	Weight	Blood pressure	Healthy / Patient
25	60	12	Healthy
27	80	14	Patient
…	…	…	…

X: (Age=23, Weight=65, Blood pressure=13)

$$P(\text{healthy} \mid X) = \frac{P(\text{healthy})\,P(X \mid \text{healthy})}{P(X)},\ P(\text{patient} \mid X) = \frac{P(\text{patient})\,P(X \mid \text{patient})}{P(X)}$$

$$\frac{P(\text{healthy})\,P(X \mid \text{healthy})}{P(X)} \gtrless \frac{P(\text{patient})\,P(X \mid \text{patient})}{P(X)}$$

$$P(\text{healthy}) = \frac{\text{count of healthy people}}{\text{count of samples}},\ P(\text{patient}) = \frac{\text{count of patient people}}{\text{count of samples}} = 1 - P(\text{healthy})$$

$$P(X \mid \text{healthy}) = \frac{\text{count of healthy people with age=23 \& weight=65 \& blood pressure=13}}{\text{count of healthy}} \rightarrow \text{Unlikely!!}$$

Naïve Bayes classifier

- A simplifying assumption: attributes are conditionally independent:

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- If A_k is categorical, $P(x_k|C_i)$ is the number of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on the Gaussian distribution

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Example

RID	Age	Income	Student	Credit	Buy
1	youth	high	No	fair	No
2	youth	high	No	excellent	No
3	middle	high	No	fair	Yes
4	senior	medium	No	fair	Yes
5	senior	low	Yes	fair	Yes
6	senior	low	Yes	excellent	No
7	middle	low	Yes	excellent	Yes
8	youth	medium	No	fair	No
9	youth	low	Yes	fair	Yes
10	senior	medium	Yes	fair	Yes
11	youth	medium	Yes	excellent	Yes
12	middle	medium	No	excellent	Yes
13	middle	high	Yes	fair	Yes
14	senior	medium	No	excellent	No

Prior probabilities	P (buy = Y)	9 / 14 = 0.643
	P (buy = N)	5 / 14 = 0.357
Compute $P(X_k \mid C_i)$ for each class and each attribute		
P (age $\leq$ 30 | buy = Y)		2 / 9 = 0.222
P (age $\leq$ 30 | buy = N)		3 / 5 = 0.600
P (31 $\leq$ age $\leq$ 40 | buy = Y)		4 / 9 = 0.444
P (31 $\leq$ age $\leq$ 40 | buy = N)		0
P (student = Y | buy = Y)		6 / 9 = 0.667
P (student = Y | buy = N)		1 / 5 = 0.200
P (credit = F | buy = Y)		6 / 9 = 0.667
P (credit = F | buy = N)		2 / 5 = 0.400
…		…

Training phase

Example

X = (age <= 30 , income = medium, student = yes, credit_rating = fair)

P(X|buys_computer = “yes”) = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = “no”) = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

P(X|buys_computer = “yes”) x P(buys_computer = “yes”) = 0.044 x 0.643 = 0.028
P(X|buys_computer = “no”) x P(buys_computer = “no”) = 0.019 x 0.357 = 0.007

Therefore, X belongs to class (“buys_computer = yes”)
With what probability?

Avoiding the zero-probability problem

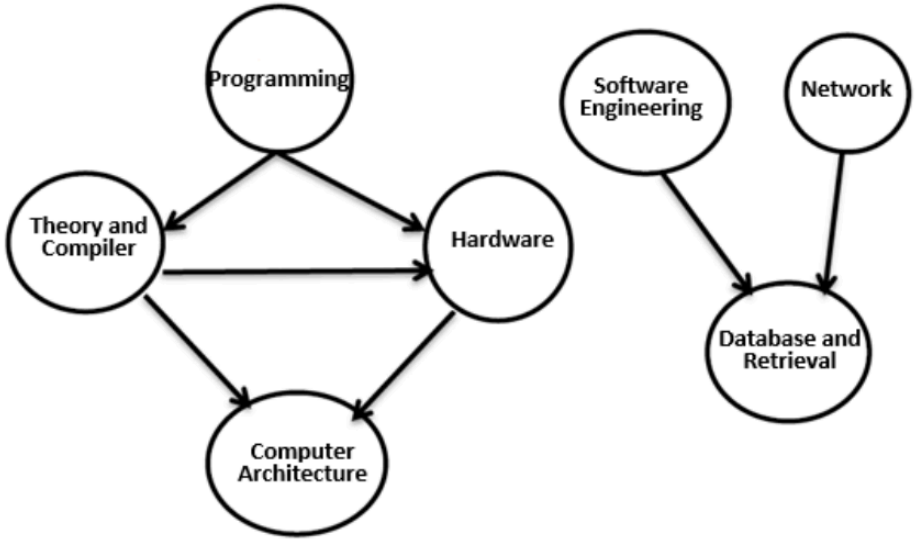
- Naïve Bayes requires each conditional probability be non-zero. Otherwise, the predicted probability will be zero

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

- Example:
  - Consider a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)
- Use Laplacian correction (or add-one smoothing)
  - Adding 1 to each case
    - Prob(income = low) = 1/1003
    - Prob(income = medium) = 991/1003
    - Prob(income = high) = 11/1003

Naïve Bayes classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in many cases
- Disadvantages
  - Strong assumption: class conditional independencies
    - How to deal with these dependencies? Bayesian Networks

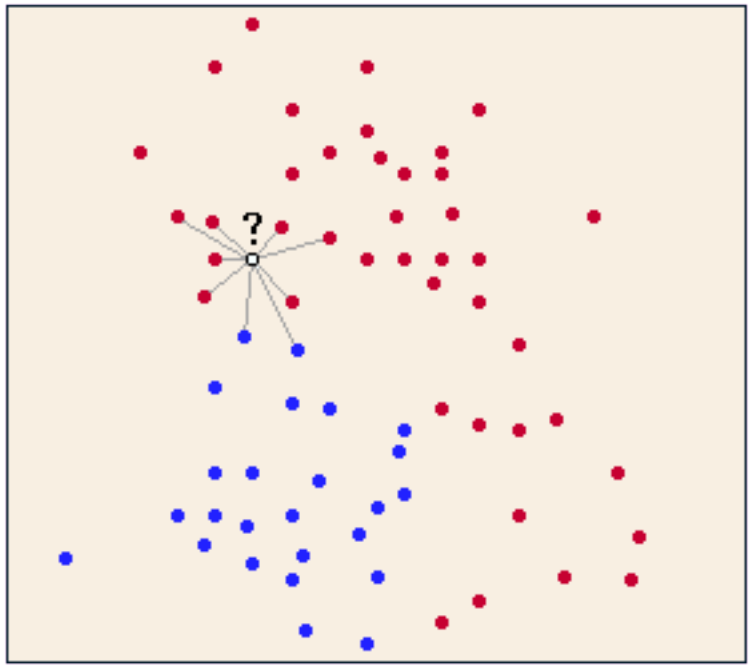


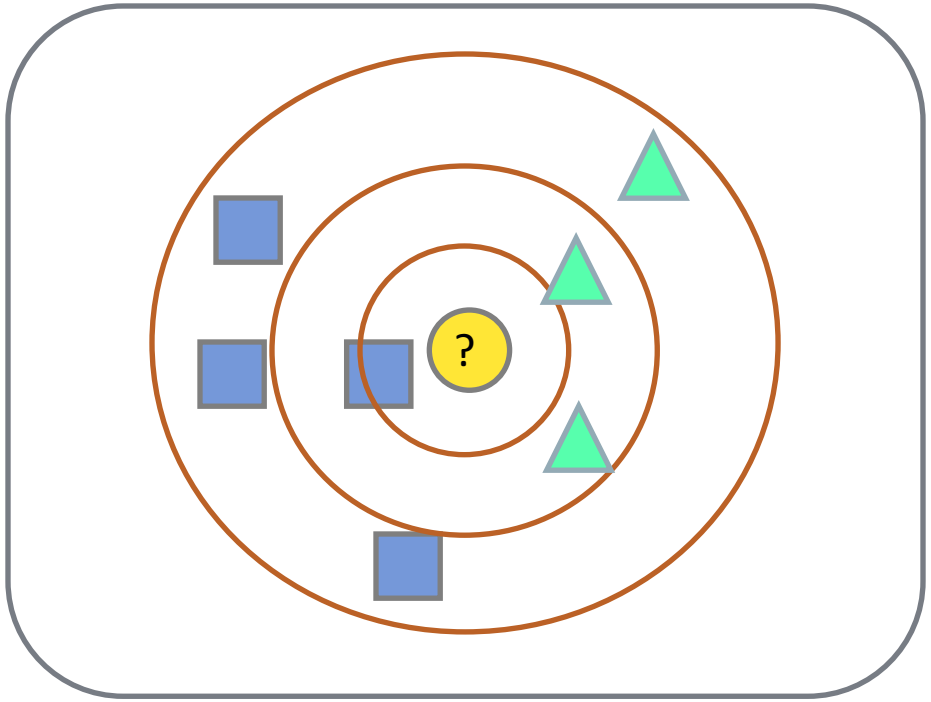
Alireza Malakouti Khah, "Investigating the hierarchy of computer engineering courses using Bayesian networks," MSc Dissertation, University of Qom, 2017.

Faezeh Abdollahi Pour, "Image steganalysis using Bayesian networks," MSc Dissertation, University of Qom, 2018.

K-Nearest Neighbors (KNN)

KNN





- k = 1:
  - Belongs to square class
- k = 3:
  - Belongs to triangle class
- k = 7:
  - Belongs to square class

KNN

- Among the simplest of all data mining algorithms
- Among the top 10 data mining algorithms
  - Wu, Xindong, et al. "Top 10 algorithms in data mining." *Knowledge and information systems* 14.1 (2008): 1-37.
- Requires 3 things:
  - Feature space (training data)
  - Distance metric
  - The value of *k*

KNN

- Applicable to both classification and regression problems.
- Combining the labels of k nearest neighbors:
  - Take the majority vote (average) of labels among the neighbors
  - Weighting: $w = \frac{1}{d}$ or $\frac{1}{d^2}$
- Example:
  - Assume these are the three nearest neighbors

Value	5	8	9
Distance	3	2	5

$$\text{prediction} = \frac{\frac{1}{3}\times 5 + \frac{1}{2}\times 8 + \frac{1}{5}\times 9}{\frac{1}{3} + \frac{1}{2} + \frac{1}{5}}$$

- Choosing the value of k:
  - If k is too small, sensitive to noisy points
  - If k is too large, neighborhood may include irrelevant points
  - Choose an odd value for k, to eliminate ties

Issues

01 Nearest neighbor classifiers are lazy learners
All computations deferred until classification

02 Training phase is fast, but applying phase is very slow

03 The KNN classifier in high dimensions suffers from the curse of dimensionality problem

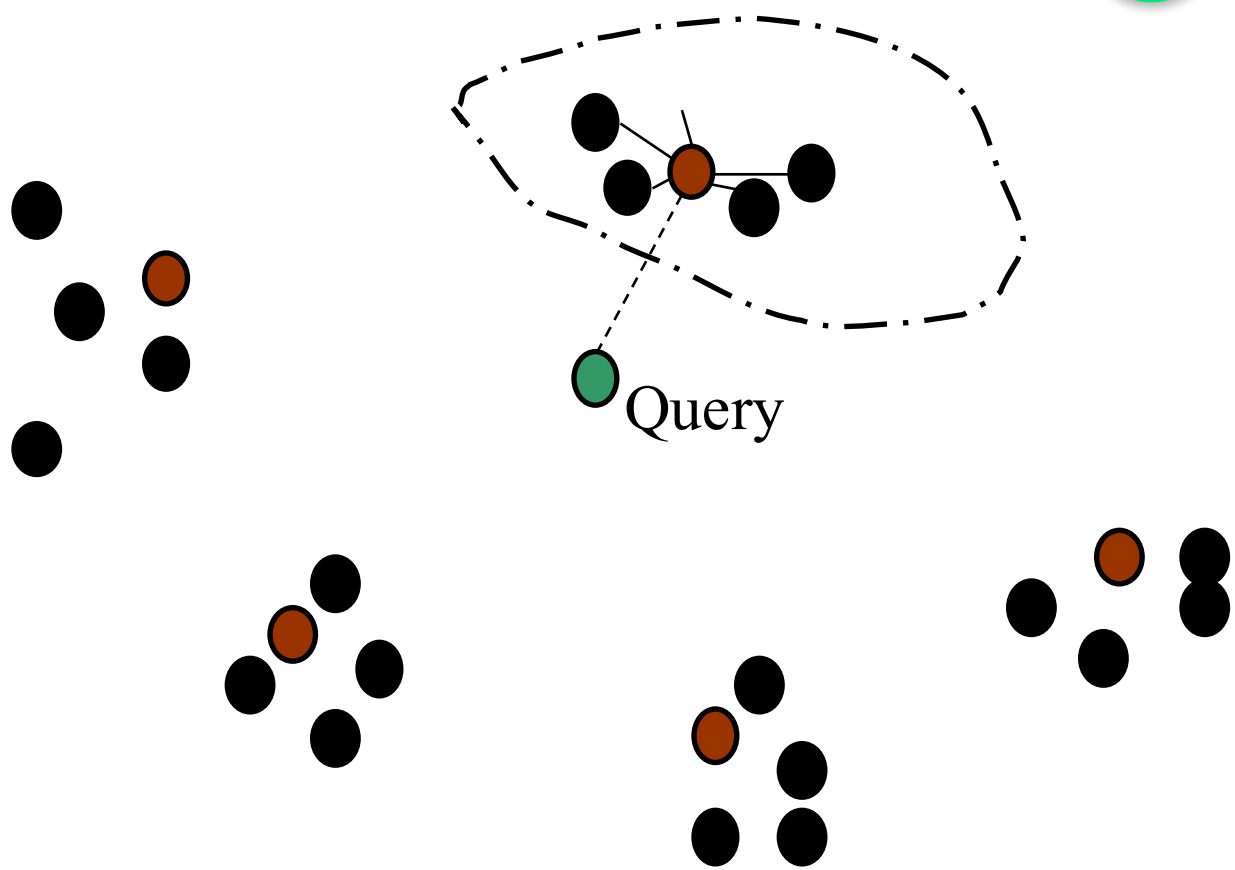
04 Needs large number of training samples

05 KNN method is instance-bases learning (non-parametric)

06 Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes

Speeding up KNN

- Cluster training data
- Seek the nearest neighbors to the test point in the nearest cluster
- Tradeoff between accuracy and speed:
  - Consider b nearest clusters

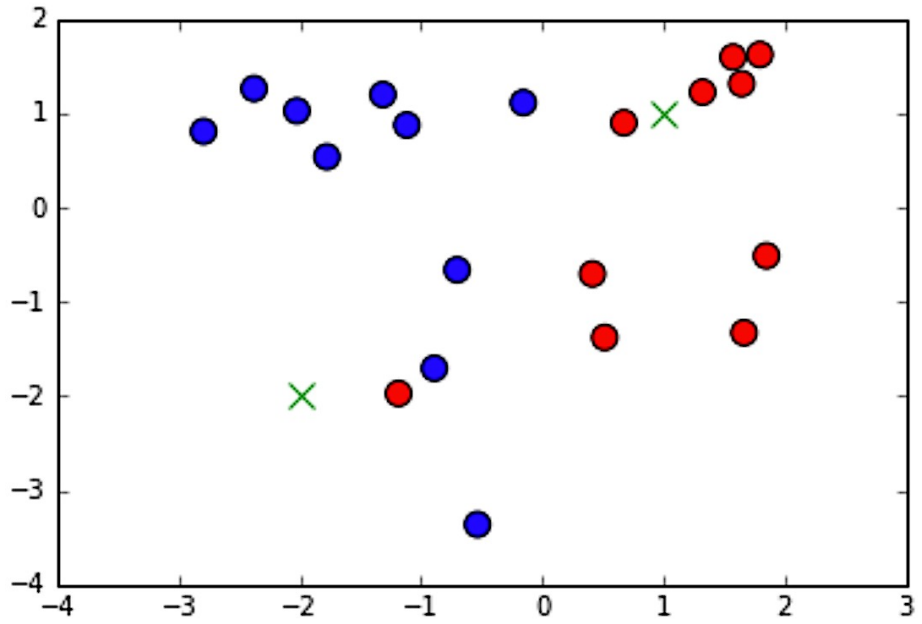


Stanford CS276 course

Python example


```
from sklearn.datasets import make_classification
X, y = make_classification(n_features=2, n_redundant=0,
                           n_informative=2, n_samples=20)
```

```
plt.scatter(X[:,0], X[:,1], cmap='bwr', s=100, c=y)
plt.scatter([-2,1], [-2,1], marker='x',s=100, c='g')
plt.show()
```



Python example

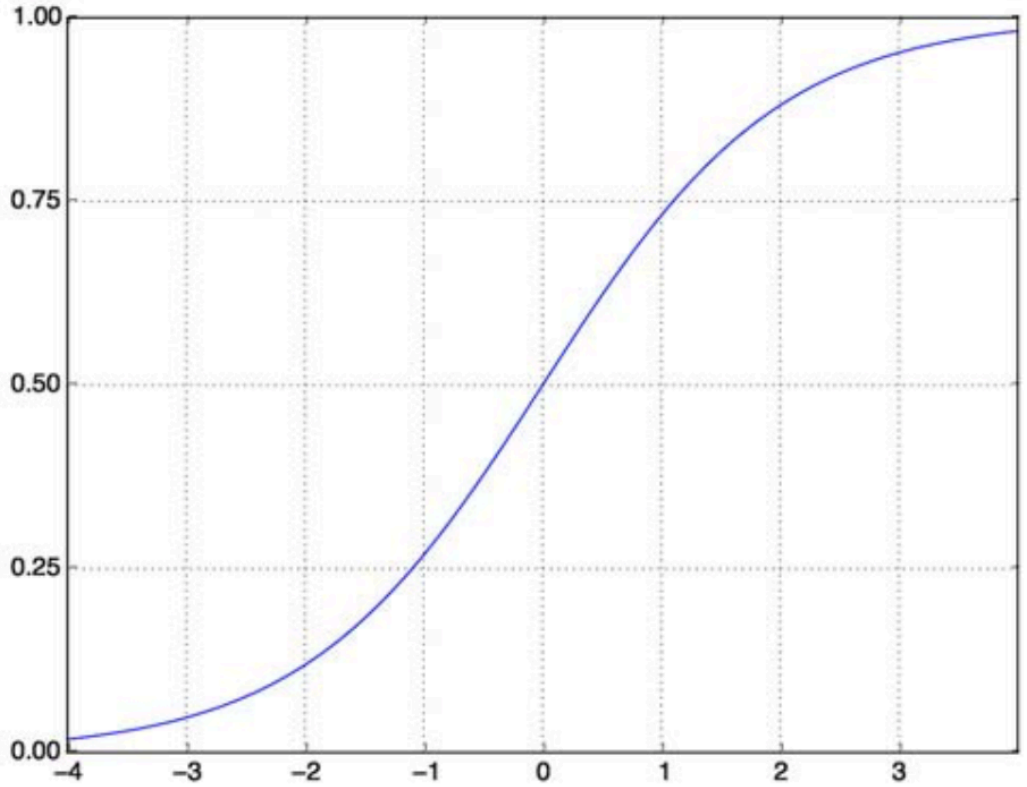

```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=1)
clf.fit(X,y)
print('With k=1: ', clf.predict([[-2,-2], [1,1]]))

clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X,y)
print('With k=3: ', clf.predict([[-2,-2], [1,1]]))
```

```
With k=1:  [1 1]
With k=3:  [0 1]
```

Logistic Regression

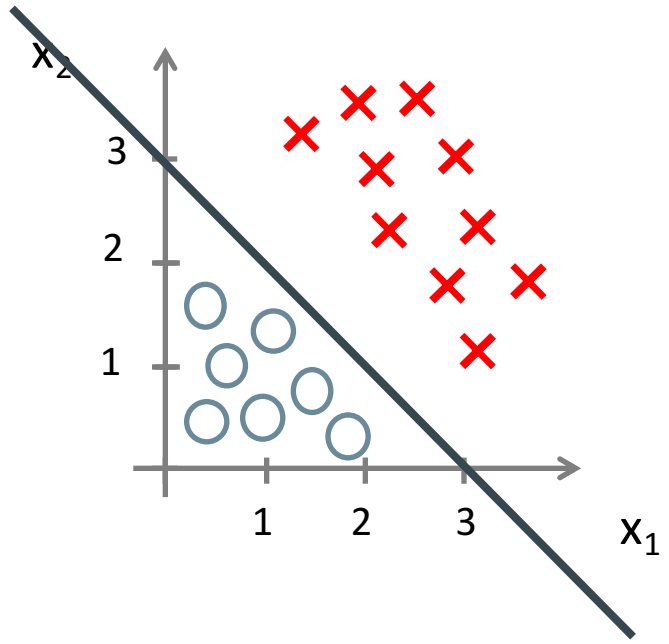
Logistic regression

- Gives the probability of different classes, instead of just predicting the class.
- $Z = w_0 + w_1X_1 + w_2X_2 + \cdots + w_nX_n$
- We apply sigmoid (logistic) function on Z to obtain something that can be interpreted as probability.
  - $\sigma(Z) = \frac{1}{1+e^{-Z}}$



Logistic regression

- Assume a threshold like 0.5 and
  - Predict y=1 if $\sigma(z) \geq 0.5$
  - That is, predict y=1 if $w_0 + w_1 X_1 + w_2 X_2 + \cdots \geq 0$
- For example, predict y=1 if
  - $-3 + X_1 + X_2 \geq 0$
- It is a linear classifier:
  - It has a linear decision boundary.



Machine Learning Coursera course by Andrew Ng

Python example


```
import pandas as pd
data = pd.read_csv(
    'http://www.karlin.mff.cuni.cz/~pesta/prednasky/NMFM404/Data/binary.csv')
data.head()
```

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

```
data.describe()
```

	admit	gre	gpa	rank
count	400.000000	400.000000	400.000000	400.00000
mean	0.317500	587.700000	3.389900	2.48500
std	0.466087	115.516536	0.380567	0.94446
min	0.000000	220.000000	2.260000	1.00000
25%	0.000000	520.000000	3.130000	2.00000
50%	0.000000	580.000000	3.395000	2.00000
75%	1.000000	660.000000	3.670000	3.00000
max	1.000000	800.000000	4.000000	4.00000

Python example


```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(data[['gre','gpa','rank']], data['admit'])
print(clf.coef_)
print(clf.intercept_)
```

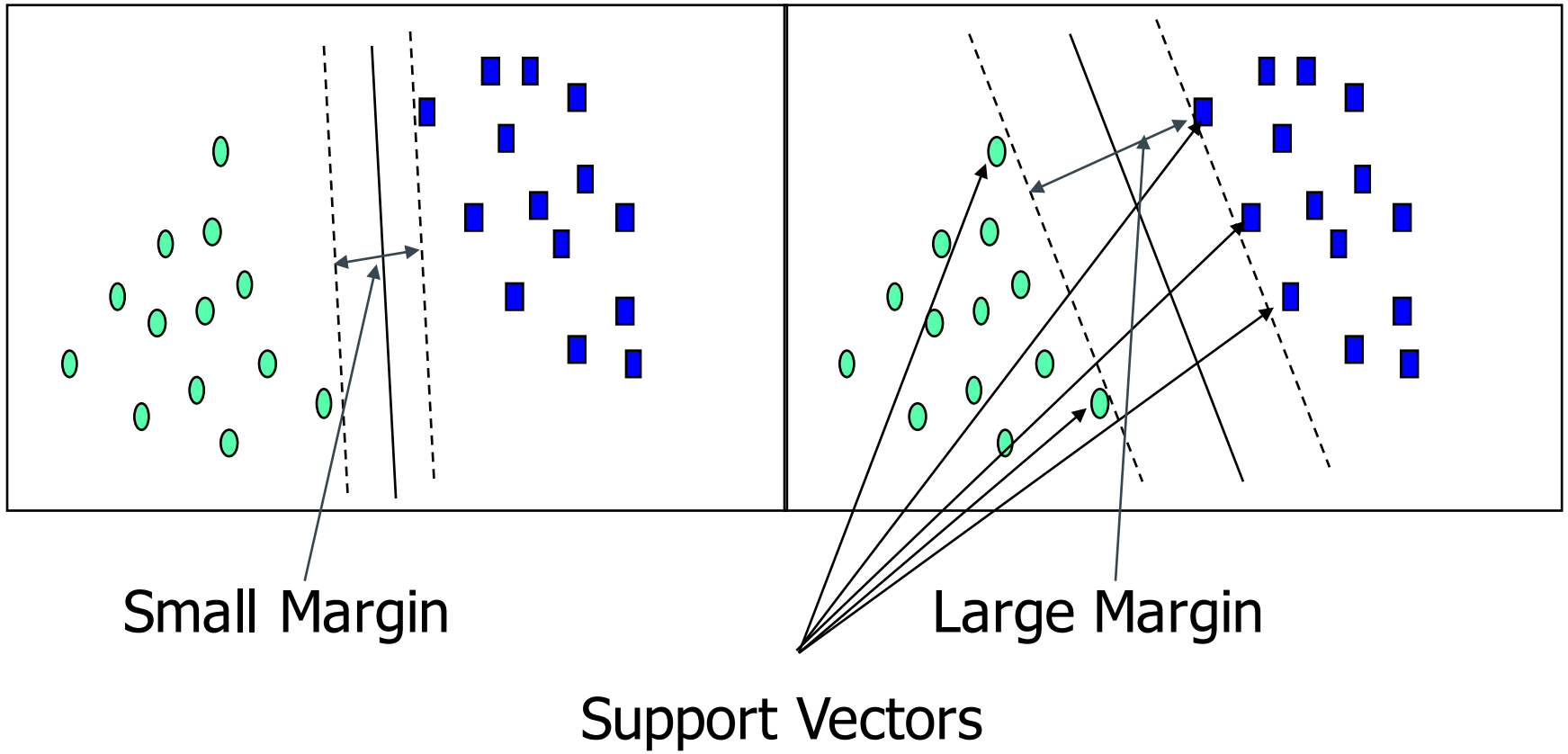
```
[[ 0.00188723  0.31928562 -0.60537482]]
[-1.5115323]
```

```
test_data = [[600, 3, 2], [700, 3.7, 1]]
print(clf.predict(test_data))
print(clf.predict_proba(test_data))
```

```
[0 1]
[[0.65296209 0.34703791]
 [0.40479791 0.59520209]]
```

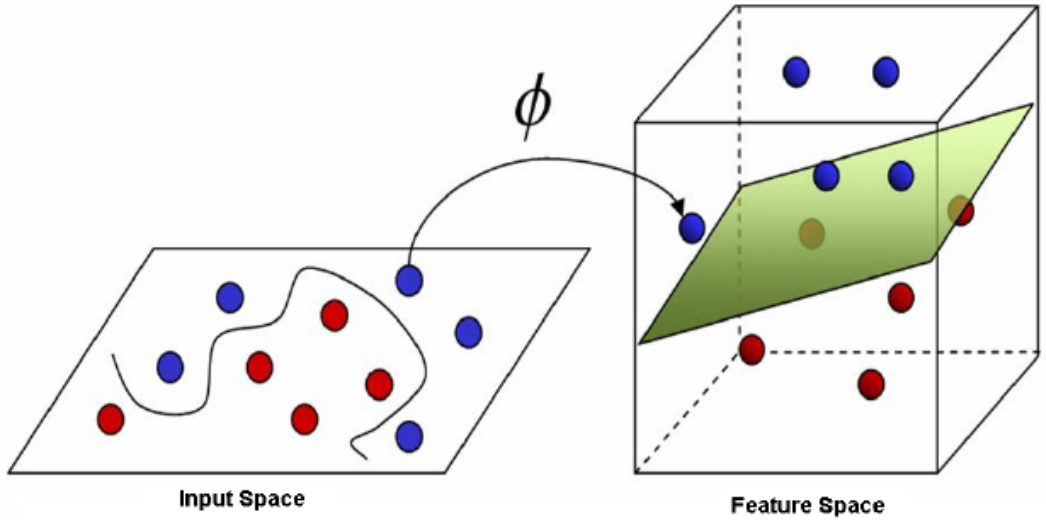
Support Vector Machine (SVM)

SVM



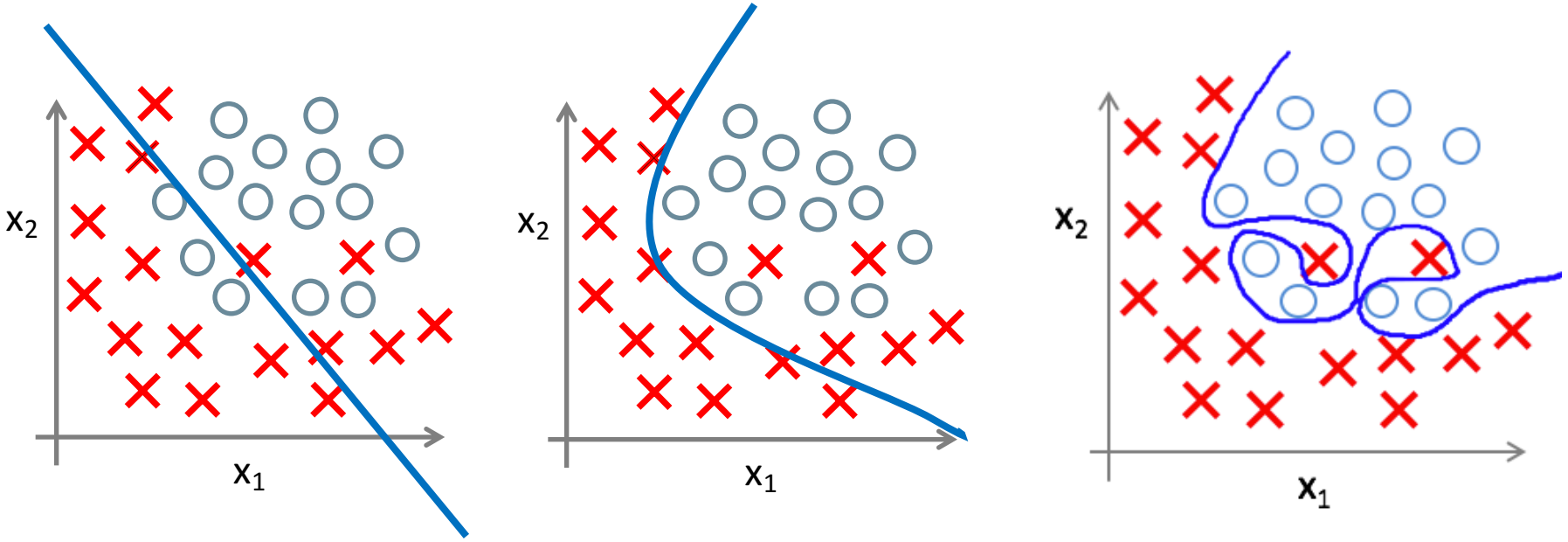
SVM

- With an appropriate nonlinear mapping to a sufficiently high dimension (kernel trick), data from two classes can always be separated by a hyperplane.
- With the new dimension, SVM searches for the optimal linear decision boundary.



http://thecaffeinedev.com/2-support-vector-machine-learning-math-behind-part2/

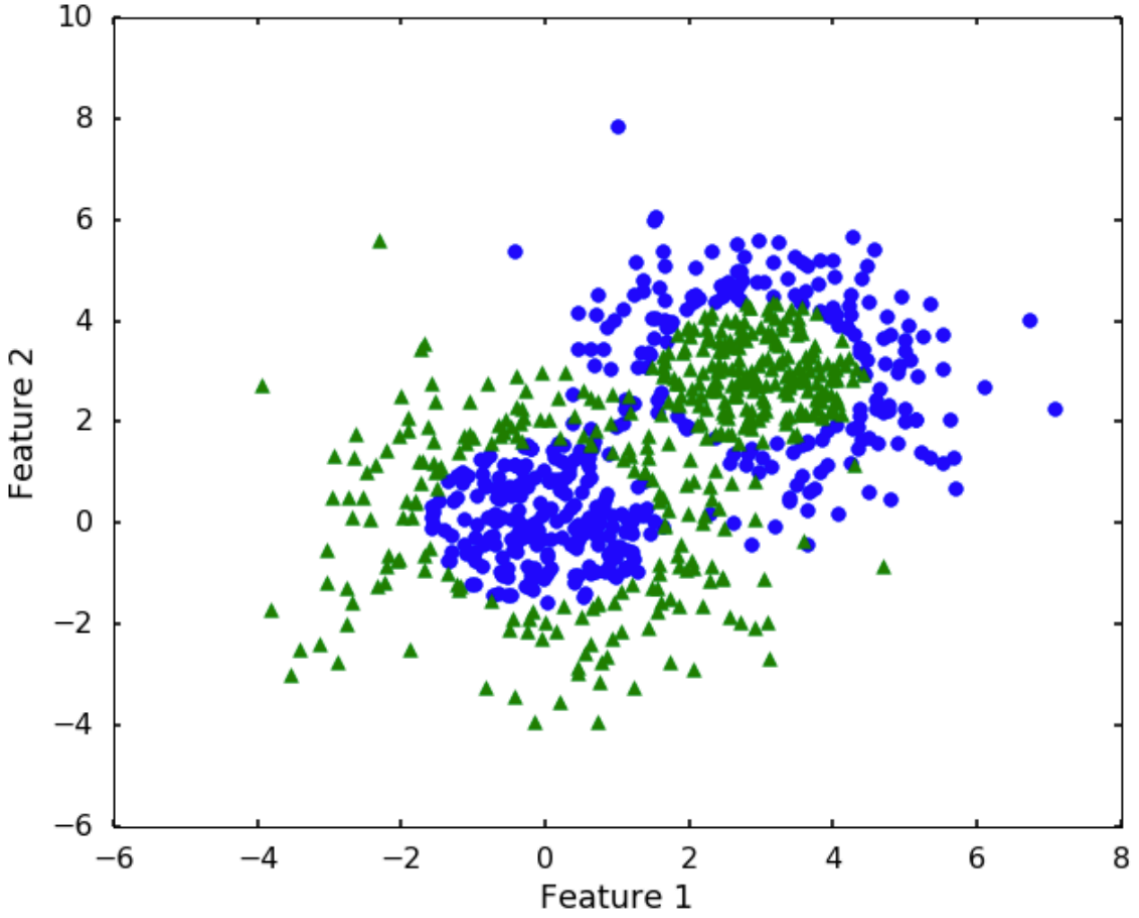
SVM



Underfit OK Overfit

In sklearn, this is controlled by C parameter of SVM and gamma parameter of rbf kernel.

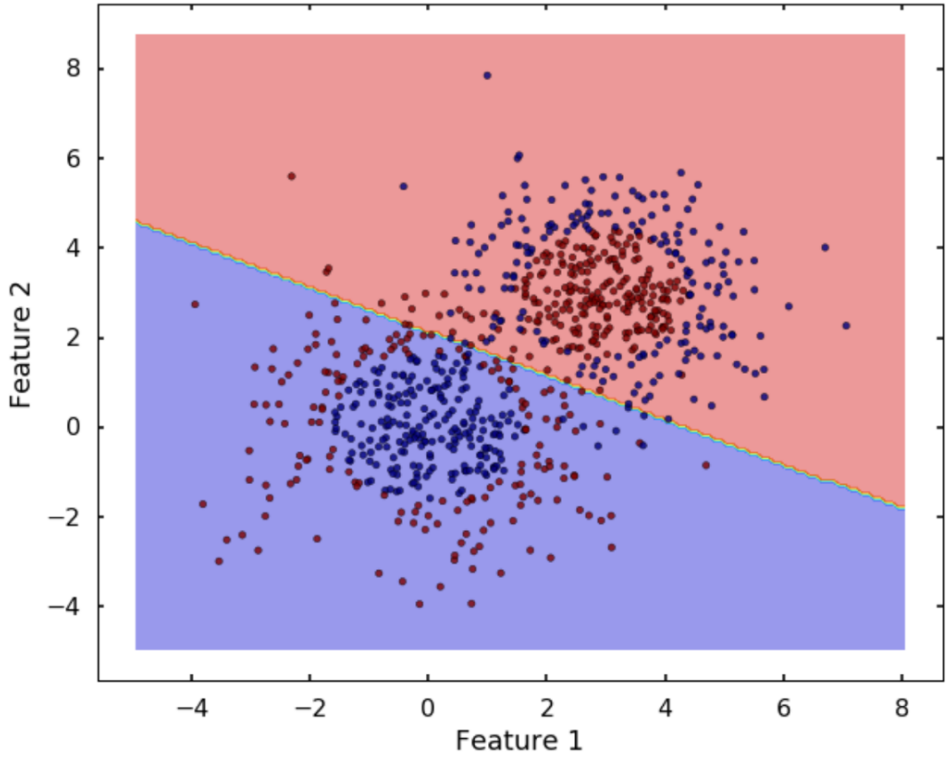
Python example



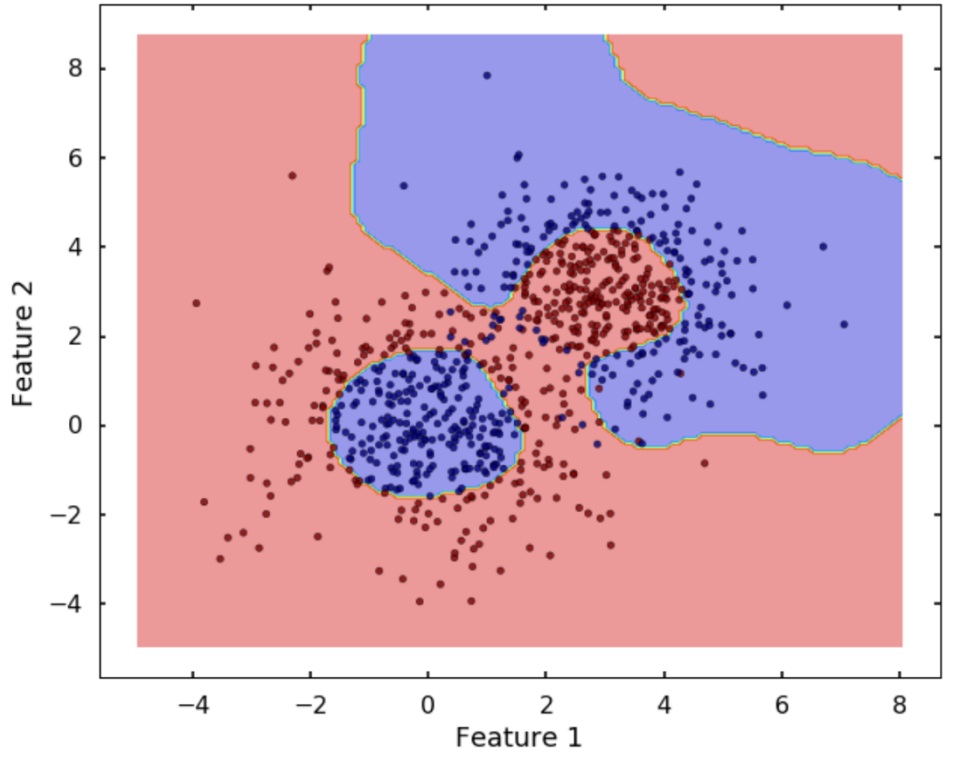
http://qingkaikong.blogspot.com/2016/12/machine-learning-8-support-vector.html

Python example


```
clf = svm.SVC(kernel='linear')
clf.fit(X,y)
```



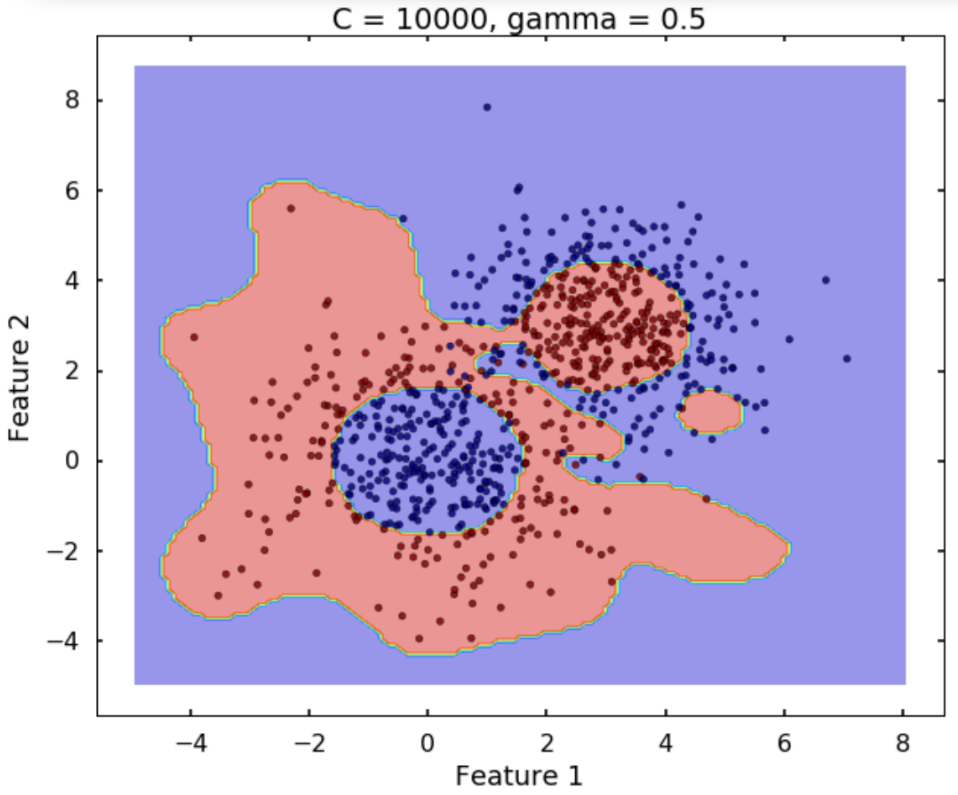

```
clf = svm.SVC(kernel='rbf')
clf.fit(X,y)
```



Python example

- A large C makes the cost of misclassification high.
- This will force the algorithm to fit the data with more flexible model.
- But this will also make the SVM more easier to overit the data.

```
clf = svm.SVC(kernel='rbf', C = 10000, gamma = 0.5)
clf.fit(X,y)
```



Ensemble Learning

Wisdom of crowd

Ensemble learning

- Motivations:
  - Ensemble model improves accuracy and robustness over single model methods.
  - A complex problem can be decomposed into multiple sub-problems that are easier to understand and solve (divide-and-conquer approach).
- Other applications:
  - Distributed computing
  - Large-scale data
  - Multiple sources of data
  - Privacy-preserving applications

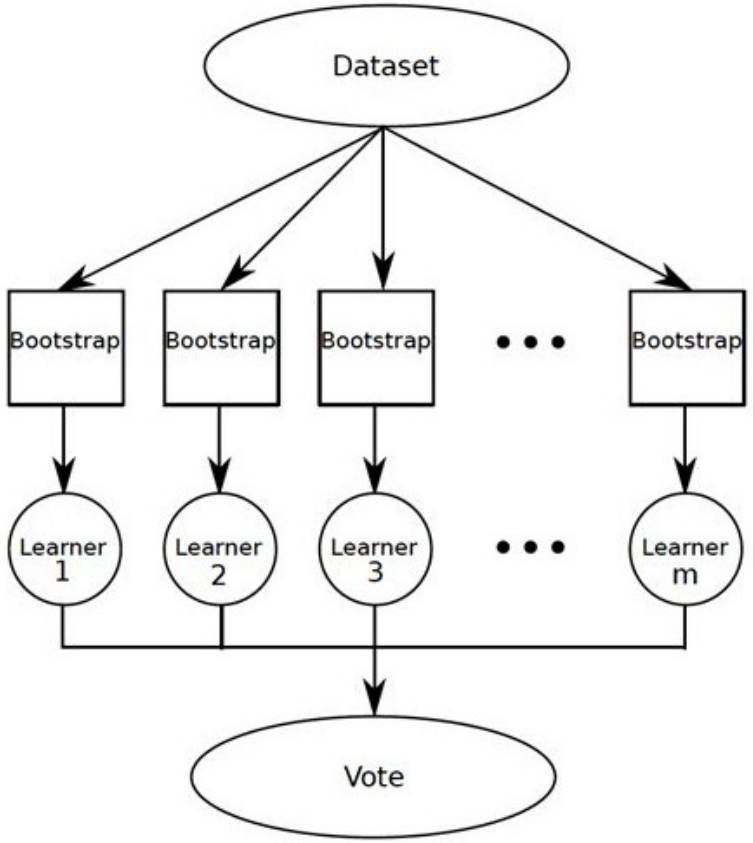
Ensemble learning

- Requirements of base learners:
  - High variety
  - Acceptable accuracy

- Popular methods:
  - Bagging
  - Boosting
  - Stacking

1. Bagging: Bootstrap Aggregation

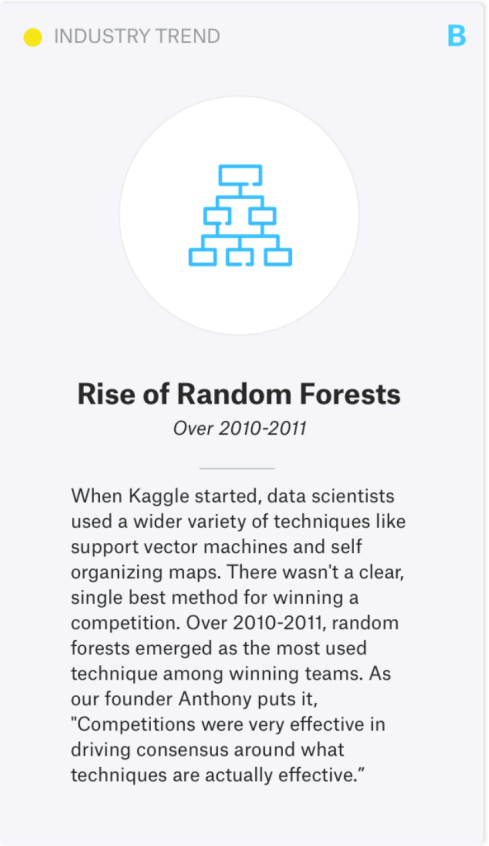
- Training
  - Given a set D of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: to classify an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Regression: take the average value instead of voting
- Bagging produces a combined model that often performs significantly better than the single model built from the original training data, and is never substantially worse.



Source: KDnuggets.com

Random forest

- Bagging helps most if the underlying learning scheme is unstable in that small changes in the input data can lead to quite different classifiers. This is true for decision trees.
- Random forest:
  - Each classifier in the ensemble is a decision tree classifier and is generated using a random selection of attributes at each node to determine the split.



http://blog.kaggle.com/2017/06/06/weve-passed-1-million-members/

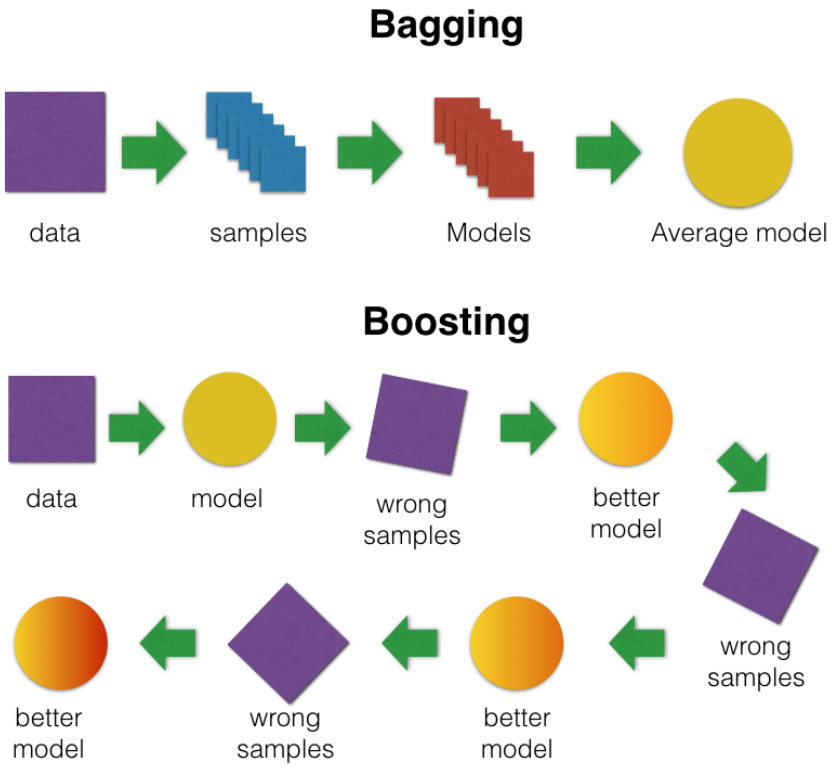
Random forest


```python
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()

# train
clf = RandomForestClassifier()
clf.fit(iris.data, iris.target)

# usage
X_test = [[5,3,1,0.5], [6,4,5,2]]
print('Predicted labels:', clf.predict(X_test))
print('Predicted probabilities:\n', clf.predict_proba(X_test))
print('Feature importances:', clf.feature_importances_)
```

```
Predicted labels: [0 2]
Predicted probabilities:
 [[1. 0. 0.]
 [0. 0. 1.]]
Feature importances: [0.11157549 0.02615675 0.29713803 0.56512973]
```

2. Boosting

- Comparing with bagging:
  - The same base classifiers are used in both
  - Boosting uses weighted voting/averaging
  - Bagging is parallel while boosting is sequential
  - Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data.



Source: bradzzz.gitbooks.io

2. Boosting

- How boosting works?
  - A series of k classifiers is iteratively learned
  - Weights are assigned to each training tuple
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to pay more attention to the training tuples that were misclassified by M_i
  - The final M* combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy

Adaboost

- Given a set of *d* tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If a tuple is misclassified, its weight is increased, otherwise it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- Terminate if $error(M_i)$ equal to zero, or greater or equal to 0.5.
- The weight of classifier M_i's vote is

$$\log \frac{1 - error(M_i)}{error(M_i)}$$

Adaboost


```
from sklearn.ensemble import AdaBoostClassifier

# train
clf = AdaBoostClassifier()
clf.fit(iris.data, iris.target)

# usage
X_test = [[5,3,1,0.5], [6,4,5,2]]
print('Predicted labels:', clf.predict(X_test))
print('Predicted probabilities:\n', clf.predict_proba(X_test))
```

```
Predicted labels: [0 2]
Predicted probabilities:
 [[9.99993050e-01 6.78858954e-06 1.61467062e-07]
 [3.07612557e-16 2.19633469e-02 9.78036653e-01]]
```

Microsoft LightGBM


```
from lightgbm import LGBMClassifier

# train
clf = LGBMClassifier()
clf.fit(iris.data, iris.target)

# usage
X_test = [[5,3,1,0.5], [6,4,5,2]]
print('Predicted labels:', clf.predict(X_test))
print('Predicted probabilities:\n', clf.predict_proba(X_test))
```

```
Predicted labels: [0 2]
Predicted probabilities:
 [[9.82109178e-01 1.70043317e-02 8.86489818e-04]
 [2.12142867e-03 4.70510235e-02 9.50827548e-01]]
```

XGBoost

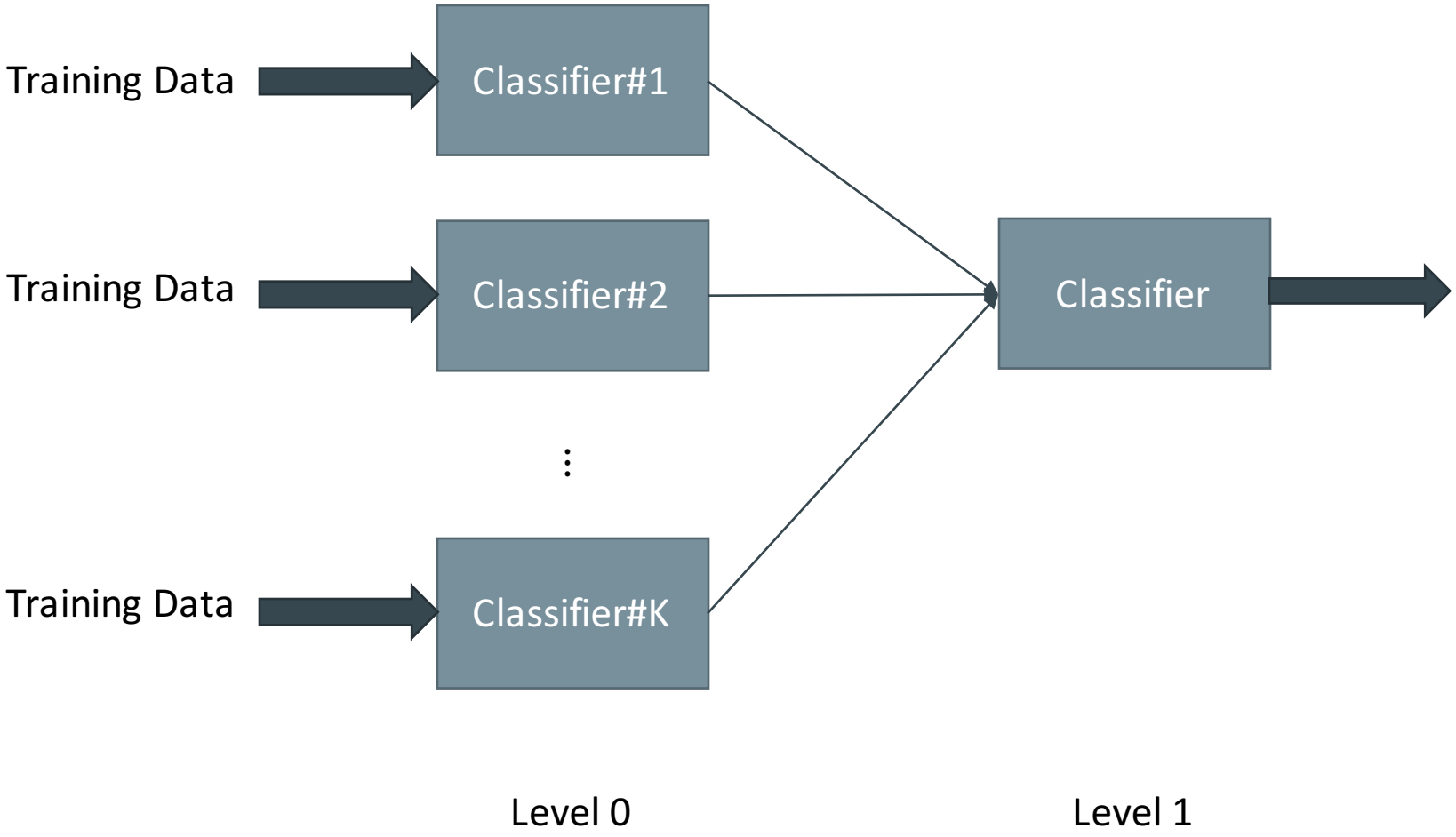

```
from xgboost import XGBClassifier

# train
clf = XGBClassifier()
clf.fit(iris.data, iris.target)

# usage
X_test = [[5,3,1,0.5], [6,4,5,2]]
print('Predicted labels:', clf.predict(X_test))
print('Predicted probabilities:\n', clf.predict_proba(X_test))
```

```
Predicted labels: [0 2]
Predicted probabilities:
 [[ 0.99407768  0.0039501   0.00197222]
 [ 0.01047136  0.07157398  0.91795462]]
```

A good tutorial on hyperparameter tuning in XGBoost:
http://analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

3. Stacking

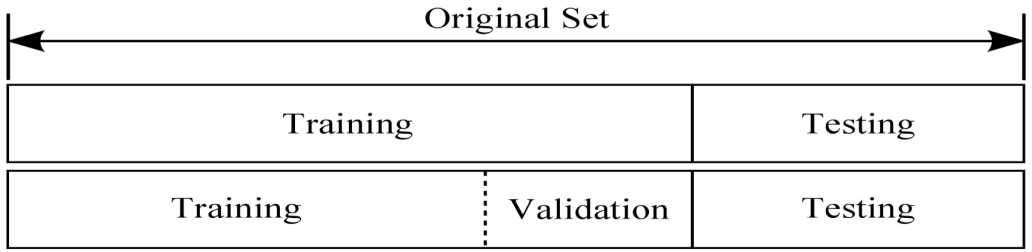
- It introduces the concept of a metalearner, which replaces the voting procedure.
- Normally is used to combine models of different types.
- Because most of the work is already done by the level-0 learners, it makes sense to choose a rather simple algorithm for the level-1 classifier.
- Use out-of-fold predictions (OOF) as the training data for the level 1 classifier.



Evaluation

Evaluation

- To compare different models.
- To tune the hyperparameters such as
  - K in KNN, number of layers in neural networks, the best pruning of a decision tree, etc.
- The main goal of ML is generalization. We want to measure the generalization ability of our model.
- Hold-out method: You train on the *training data* and evaluate your model on the *validation data*. Once your model is ready, you test it one final time on the *test data*.
- Shuffle data before splitting.



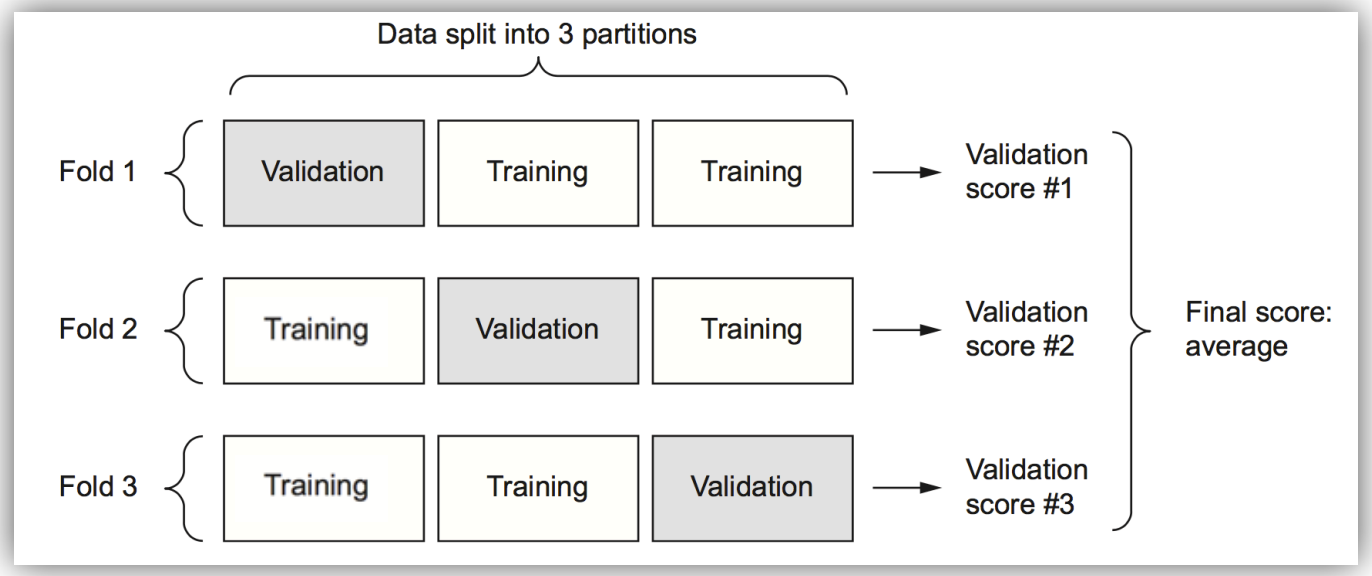
Evaluation

- Why not just train and test?
  - We do this tuning by using as a feedback signal the performance of the model on the validation data.
  - It can quickly result in *overfitting to the validation set (information leak).*
  - At the end of the day, you'll end up with a model that performs artificially well on the validation data, but not on completely new data.

375	▾373	Not So Dumb Any More		0.29006	293	2mo

K-fold cross validation

- When you have few data points, the validation set would end up being very small. This would prevent you from reliably evaluating your model. So, we use k-fold cross validation.
- Typical values for k: 5, 10, N (leave-one-out method).



Evaluation metrics

Confusion Matrix

Predict class / True class	Positive	Negative
Positive	TP	FN
Negative	FP	TN

TP: True Positive
TN: True Negative
FP: False Positive (type I error)
FN: False Negative (Type II error)

Evaluation metrics

- **Accuracy**: percentage of tuples that are correctly classified

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

- Is it always a good measure?
  - Consider a cancer detection system always predicts “no cancer”
- Not a good measure for imbalanced data.

Evaluation metrics

- **Precision**: what percentage of tuples that the classifier labeled as positive are actually positive?

$$precision = \frac{TP}{TP + FP}$$

Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total
cancer = yes	90	210	300
cancer = no	140	9560	9700
Total	230	9770	10000

Precision = 90/230 = 39.13%

Evaluation metrics

- **Recall**: what percentage of positive tuples did the classifier label as positive?

$$Recall = \frac{TP}{TP + FN}$$

Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total
cancer = yes	90	210	300
cancer = no	140	9560	9700
Total	230	9770	10000

Recall = 90/300 = 30%

Evaluation metrics

- Is it enough to have good precision OR good recall?
- We should combine precision and recall into one measure.
- The most popular way is by harmonic mean: F-measure

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

Evaluation metrics

- For multi-class problems:
  - Micro: Compute all TP, FN, and FP
  - Macro: Average F-measure of different classes
  - Weighted: Weighted average F-measures of different classes, where the weight of each class is proportional to the number of samples in that class.

```
from sklearn.metrics import f1_score

true = [0,1,2,0,1,2,0]
pred = [0,2,1,0,0,2,0]

print('Micro:', f1_score(true, pred, average='micro'))
print('Macro:', f1_score(true, pred, average='macro'))
print('Weighted:', f1_score(true, pred, average='weighted'))
```

```
Micro: 0.5714285714285714
Macro: 0.4523809523809524
Weighted: 0.5102040816326531
```

$C_0 = \{TP: 3, FP: 1, FN: 0\}$

$P = \frac{3}{4}, R = 1, F_1 = 0.86$

$C_1 = \{TP: 0, FP: 1, FN: 2\}$

$P = 0, R = 0, F_1 = 0$

$C_2 = \{TP: 1, FP: 1, FN: 1\}$

$P = 0.5, R = 0.5, F_1 = 0.5$

For micro: $P = \frac{4}{7}, R = \frac{4}{7}$

Python example


```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score

data = load_breast_cancer()
X, y = data.data, data.target

models = {'Decision Tree':DecisionTreeClassifier(),
          'Logistic Regression':LogisticRegression(),
          'Random Forest':RandomForestClassifier(),
          'Adaboost':AdaBoostClassifier()}
for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=5, scoring='f1_weighted')
    print(name, scores, scores.mean())
```

```
Decision Tree [0.91 0.92 0.9  0.94 0.9 ] 0.914221621695664
Adaboost [0.94 0.97 0.99 0.98 0.96] 0.9701923679227485
Logistic Regression [0.93 0.94 0.97 0.95 0.96] 0.9505832605036615
Random Forest [0.95 0.97 0.97 0.96 0.97] 0.9648362233834484
```

References

Thanks to Mina Heidari and Shohreh Teymoori, members of our lab, for their assistance in preparing this slide.

- J. Han, M. Kamber, J. Pei, "Data Mining, Concepts and Techniques," 3rd edition, 2011.
- Tan, Steinbach, Kumar, '' Classification: Basic Concepts, Decision Trees, and Model Evaluation,'' Lecture Notes for Chapter 4.
- G. Wolf, ''Introduction to Data Mining: Classification & Decision Trees,'' Yale University Fall 2016.
- Slides of "Lesson9-Classification1", Unibz University, 2012.
- F. Chollet, "Deep Learning with Python," 2018.
- Data mining course by X. Wu, University of Vermont.

Exercises

1. Chapter 8 of Han's book, 3rd edition: 3, 4, 5, 7, 16 (read part 8.6.5 for this question).
2. Project:
   - Choose a dataset from http://archive.ics.uci.edu/ml/datasets.html and apply four arbitrary classification algorithms learned in this chapter.
   - Find the appropriate hyperparameters (such as K in KNN, C in Logistic Regression and SVM, etc.) using cross validation. Show the results as informative plots.
   - Prepare and submit a *scientific report*.
   - Deadline: 10 PM, 31th Tir 1397.

The End