# Chapter 3

# Supervised Learning

# Supervised Learning

- Supervised learning: learning a function that maps an input to an output based on example **input-output** pairs.

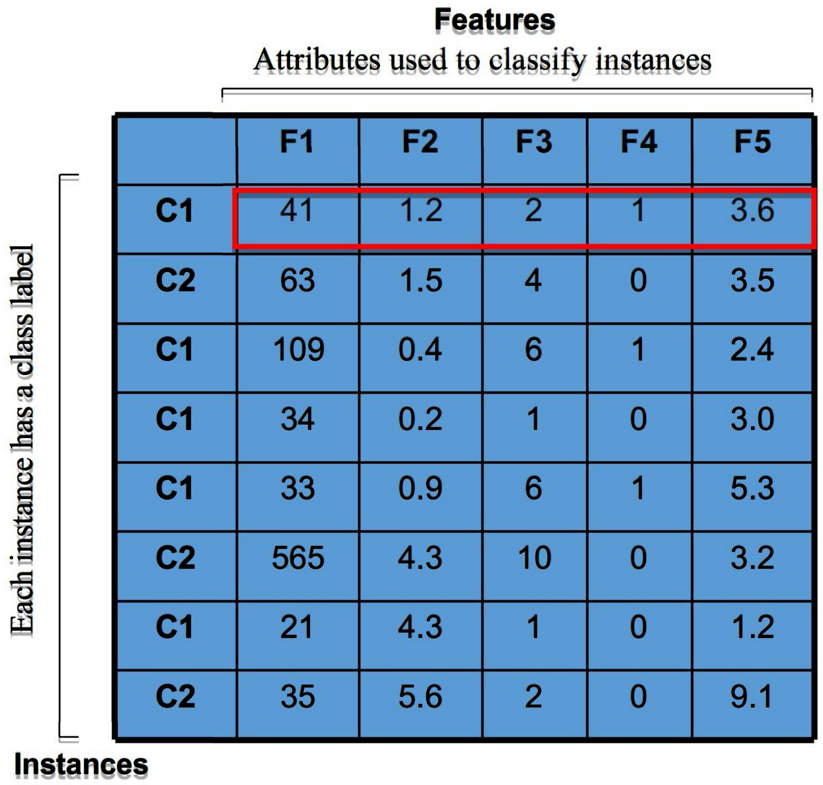
- Supervised learning:
  - **Regression** problem, if the target variable is **continuous**
    - Regression algorithms are used to **predict the continuous** values such as price, salary, age, etc.
    - You can estimate the selling price of your house  in your city.
  - **Classification** problem, if the target variable is categorical,
    - Classification algorithms are used to predict/Classify the discrete values such as Male or Female, True or False, Spam or Not Spam, etc.

# Background – Terminology

- Let’s review some common ML terms.

- Data is usually represented with a **feature matrix**.
  - **Features**
    - Attributes used for analysis
    - Represented by columns in feature matrix
  - **Instances/feature vector**
    - Entity with certain attribute values
    - Represented by rows in feature matrix

  - **Class labels**
    - Indicate category for each instance.
    - This example has two classes (C1 and C2).
    - Only used for supervised learning


Features
Attributes used to classify instances
Each instance has a class label
Instances

| | F1 | F2 | F3 | F4 | F5 |
|---|---|---|---|---|---|
| C1 | 41 | 1.2 | 2 | 1 | 3.6 |
| C2 | 63 | 1.5 | 4 | 0 | 3.5 |
| C1 | 109 | 0.4 | 6 | 1 | 2.4 |
| C1 | 34 | 0.2 | 1 | 0 | 3.0 |
| C1 | 33 | 0.9 | 6 | 1 | 5.3 |
| C2 | 565 | 4.3 | 10 | 0 | 3.2 |
| C1 | 21 | 4.3 | 1 | 0 | 1.2 |
| C2 | 35 | 5.6 | 2 | 0 | 9.1 |

**Fig. 1 Feature matrix**

# Classification

- The classification problem aims to predict group membership (i.e., labels or classes), for a set of observations.

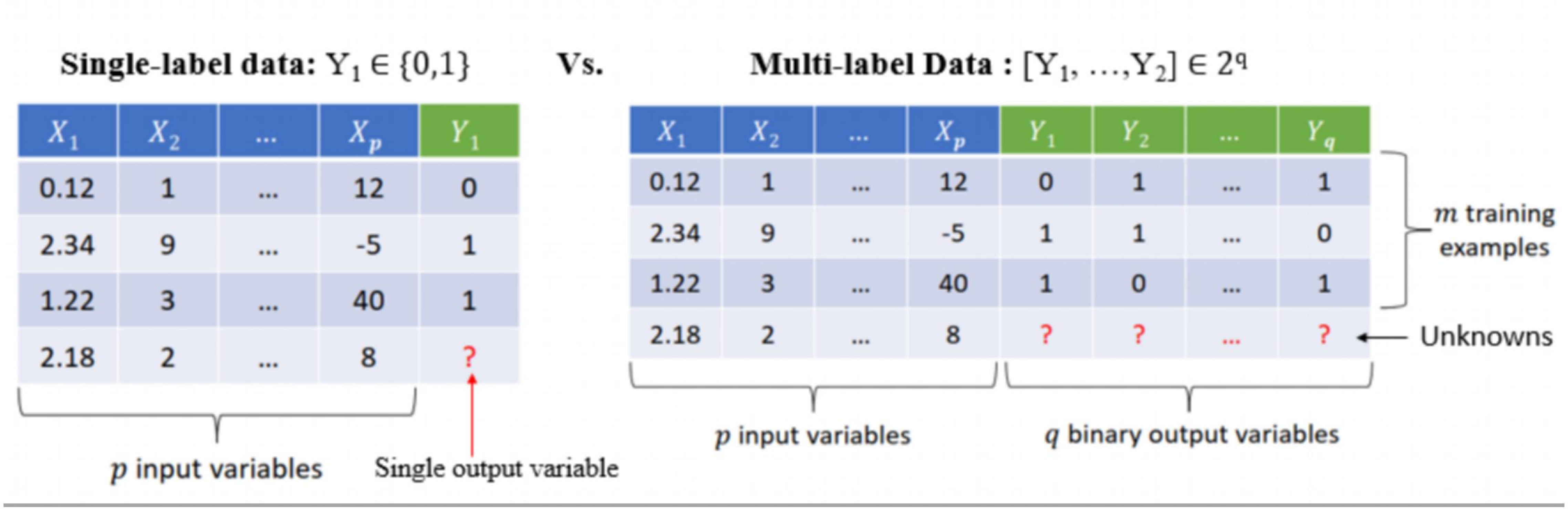
- Classification is a two-step process: a **model construction (learning)** phase, and a **model usage (applying)** phase. Sometimes model adjustment/by validation considered as 3rd step.

- **Formally:**

  Given a set of data-point $\mathbf{X= \{x_1, x_2, ..., x_n\}}$ and a finite set of targate classes $\mathbf{Y= \{y_1, y_2, ..., y_m\}}$, Classification problems is to define a mapping $\mathbf{f : X \rightarrow Y}$, where each $\mathbf{x_i}$ is assigned to one of class ($\mathbf{y_i}$).

# Single label vs multi-label Classification

- The classification problem can be further divided into two categories:
  - Single-label classification : *only one label*
  - Multi-label classification: *more than one label*



**Single-label data:** $Y_1 \in \{0,1\}$ **Vs.** **Multi-label Data :** $[Y_1, \ldots, Y_2] \in 2^q$

| $X_1$ | $X_2$ | ... | $X_p$ | $Y_1$ |
|---|---|---|---|---|
| 0.12 | 1 | ... | 12 | 0 |
| 2.34 | 9 | ... | -5 | 1 |
| 1.22 | 3 | ... | 40 | 1 |
| 2.18 | 2 | ... | 8 | ? |

| $X_1$ | $X_2$ | ... | $X_p$ | $Y_1$ | $Y_2$ | ... | $Y_q$ |
|---|---|---|---|---|---|---|---|
| 0.12 | 1 | ... | 12 | 0 | 1 | ... | 1 |
| 2.34 | 9 | ... | -5 | 1 | 1 | ... | 0 |
| 1.22 | 3 | ... | 40 | 1 | 0 | ... | 1 |
| 2.18 | 2 | ... | 8 | ? | ? | ... | ? |

$p$ input variables
Single output variable
$m$ training examples
Unknowns
$p$ input variables
$q$ binary output variables

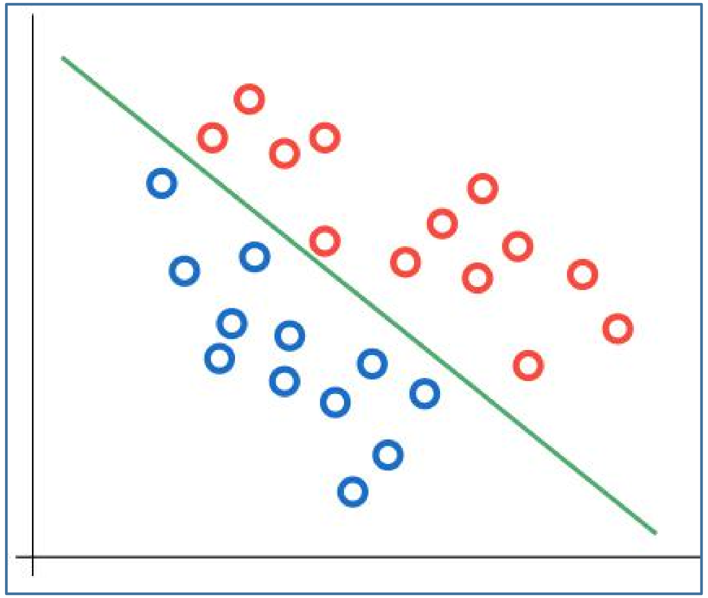
# Single label classification

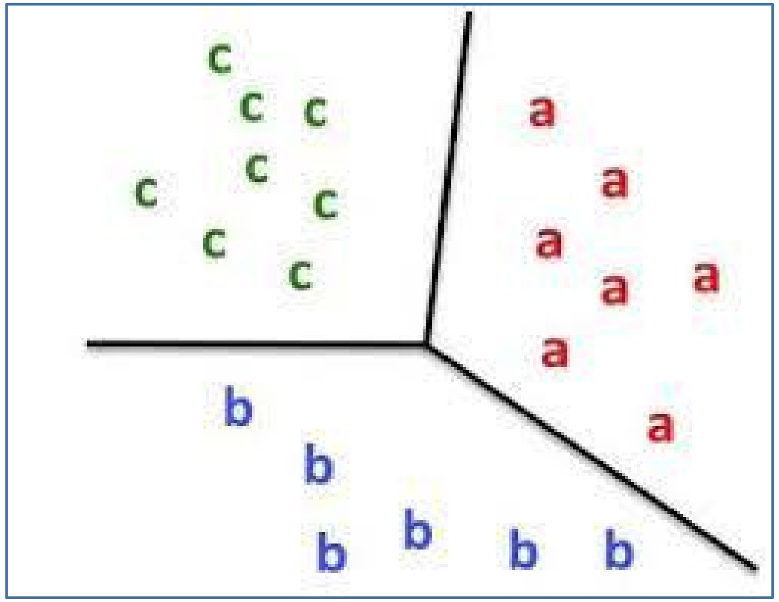
- **Single label classification:** learning from a **set of data points** that are associated with only one target label from a set of disjoint labels $Y$ , with $|Y| \geq 2$.
  - If |Y| = 2, the learning problem is called **binary classification**

    **e.g** disease diagnosis, spam, malware detection, etc.

  - if |Y | > 2, then it is called a **multi-class classification** problem

    **e.g** Language identification, an animal image classification, character recognition, face recognition, etc.



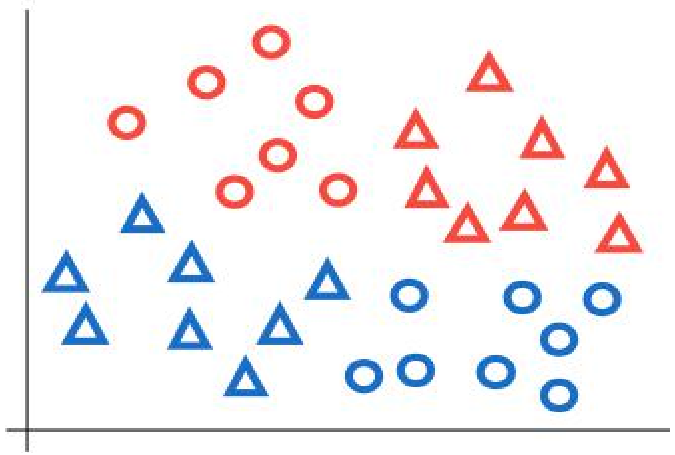


# Multi-label classification

- What is  the issue with Single-label classification?

Ans: In many real-world applications, each data instance can be associated with multiple class variables

- Examples:
  - A news article may cover multiple topics, such as politics, Agriculture, and economics



- Class1- Red Vs Blue
- Class2- Triangle Vs Oval

- So, how can we handle such multi-label problem?

# Approaches to solve multi-label classification problem

- **Problem transformation** : Transforms multi-label classification problem to multiple single-label classification problems which can be called **Adapt data to the algorithm**.

  - **Binary Relevance (BR)**: Multi-label into single-label problems.
  - **Label Power-set (LP)** : Multi-label into multi-class – consider each label-sets as class.
  - **Classifier Chains (CC)**: resolves the BR limitations, by making label correlation task.

**Other solutions [ will not coverd in this chapter]**

- **Adapted Algorithm:** Perform multi-label classification, rather than transforming the problem into different subsets of problem.

- **Ensemble Approaches:** learning multiple classifier systems train multiple hypotheses to solve the same problem.

# Binary Relevance

- Transform multi-label into single-label problems
- Suppose a training dataset D having five instances with an input feature vector X (x1 and x2) and output class vector Y (y1,y2,y3).

| X1 | X2 | Y1 | Y2 | Y3 |
|---|---|---|---|---|
| 0.3 | 0.7 | 1 | 1 | 0 |
| 0.9 | 0.3 | 1 | 1 | 0 |
| 0.6 | 0.6 | 0 | 0 | 1 |
| 0.4 | 0.1 | 0 | 1 | 0 |
| 0.5 | 0.2 | 1 | 0 | 1 |

To multiple singl-label problems

- X-->Y1
- X-->Y2
- X-->Y3

**Advantages**

- Computationally efficient

**Disadvantages**

- Does not capture the dependence relations among the class variables

# *Label Powerset Method*

- Transform each label combination to a class value and then learn a multi-class classifier with the new class values

| X1 | X2 | Y1 | Y2 | Y3 |
|---|---|---|---|---|
| 0.3 | 0.7 | 1 | 1 | 0 |
| 0.9 | 0.3 | 1 | 1 | 0 |
| 0.6 | 0.6 | 0 | 0 | 1 |
| 0.4 | 0.1 | 0 | 1 | 0 |
| 0.5 | 0.2 | 1 | 0 | 1 |

Combine label to a class value

| X1 | X2 | $Y_{comb}$ |
|---|---|---|
| 0.3 | 0.7 | 1 |
| 0.9 | 0.3 | 1 |
| 0.6 | 0.6 | 2 |
| 0.4 | 0.1 | 3 |
| 0.5 | 0.2 | 4 |

- X-->$Y_{comb}$

**Advantages**

- Learns the full joint of the class variables and each of the new class values maps to a label combination

**Disadvantages**

- The number of choices in the new class can be exponential ($|Y_{LP}| = O(2^d)$) and Learning a multi-class classifier on exponential choices is expensive

# Classifier Chains

- Similar to Binary relevance, doing multiple single-label problem but we use the previous prediction as input feature vector.

| X1 | X2 | Y1 |
|---|---|---|
| 0.3 | 0.7 | 1 |
| 0.9 | 0.3 | 1 |
| 0.6 | 0.6 | 0 |
| 0.4 | 0.1 | 0 |
| 0.5 | 0.2 | 1 |

Y1 as a class

| X1 | X2 | Y1 | Y2 |
|---|---|---|---|
| 0.3 | 0.7 | 1 | 1 |
| 0.9 | 0.3 | 1 | 1 |
| 0.6 | 0.6 | 0 | 0 |
| 0.4 | 0.1 | 0 | 1 |
| 0.5 | 0.2 | 1 | 0 |

Y1 as a feature and Y2 as a class

| X1 | X2 | Y1 | Y2 | Y3 |
|---|---|---|---|---|
| 0.3 | 0.7 | 1 | 1 | 0 |
| 0.9 | 0.3 | 1 | 1 | 0 |
| 0.6 | 0.6 | 0 | 0 | 1 |
| 0.4 | 0.1 | 0 | 1 | 0 |
| 0.5 | 0.2 | 1 | 0 | 1 |

Y1,Y2 as feature and Y3 as a class

Limitation: The result can vary for different order of chains Solution: ensemble

# Common Classification Methods

- K-Nearest-Neighbor
- Decision Tree
- Naïve Bayesian
- Support Vector Machine (SVM)
- Artificial Neural Network (ANN)

# k-Nearest Neigbhour (k-NN) Classification

In *k-nearest-neighbor* (*k*-NN) classification, the training dataset is used to classify each member of a — target dataset.

There is no model created during a  learning phase but the training set itself.

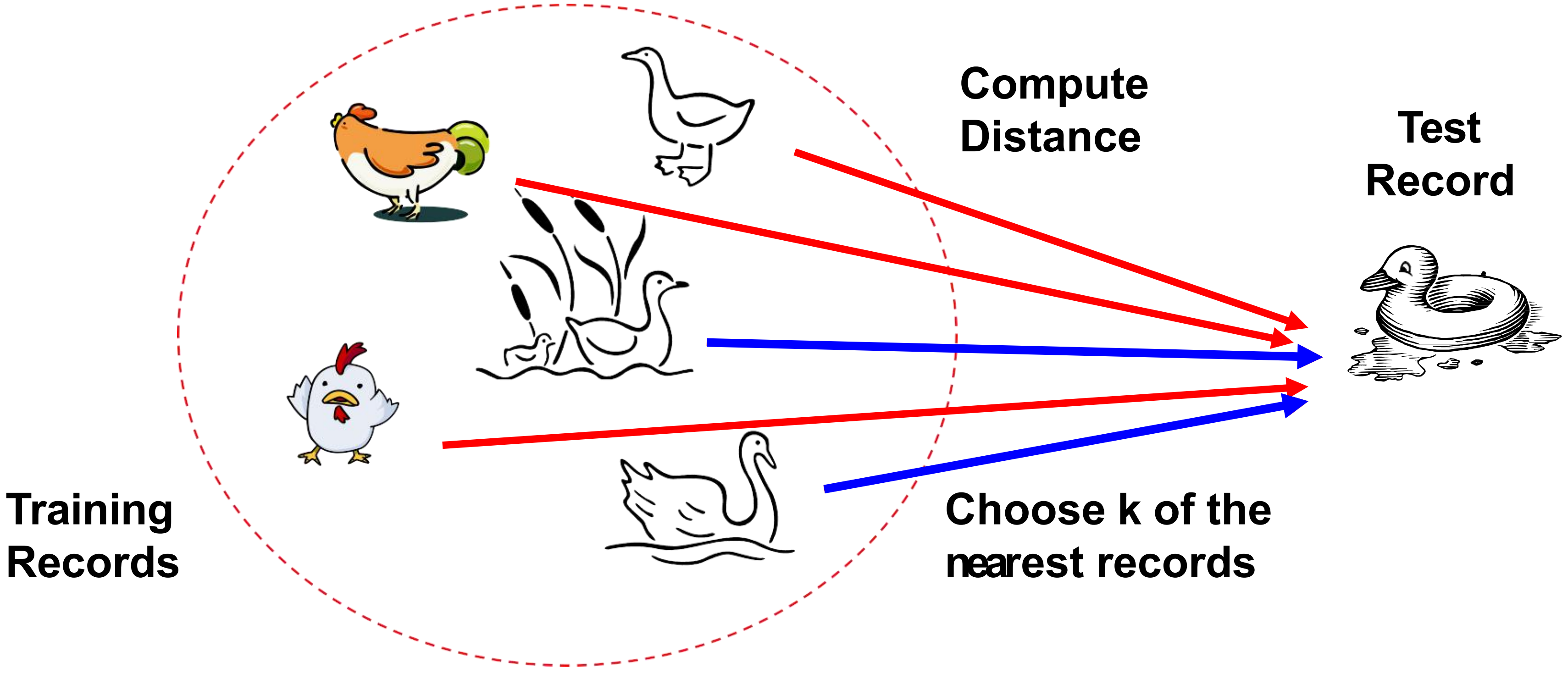
Is known as a *lazy-learning* method unlike *eager-learners* like DT, Naïve Bayes etc.

Rather than building a model and referring to it during the classification, *k*-NN directly refers to the training set for classification.

# Nearest Neighbor Classifiers

**Basic idea:** The basic idea of *nearest-neighbor* models is that the properties of any particular input **X** are likely to be similar to those of points in the neighborhood of **X**

- E.g., If the object walks like a duck, quacks like a duck, then it‘s probably a duck



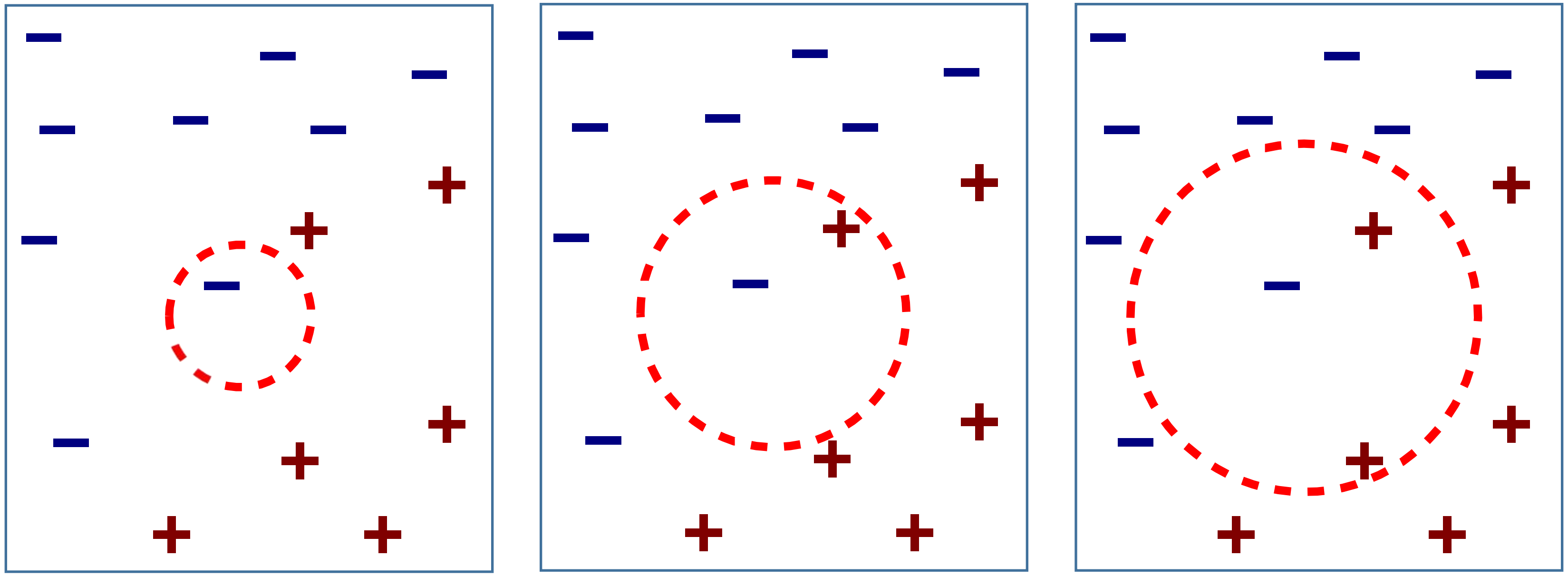
Compute Distance

Test Record

Training Records

Choose k of the nearest records

# Definition of Nearest Neighbor



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

Three points required to deal with KNN

The set of stored records

Distance Metric to compute distance between records

The value of *k*, the number of nearest neighbors to retrieve

# Nearest Neighbor Classification

- The most used distance function is the Euclidian distance:

$$d(X,Y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

- However, other measures are possible
  - the Manhattan distance: $d(X,Y)=\sum_{i=1}^{n}(x_i-y_i)$
  - the Chebychev: $d(X,Y)=\max_{i=1}^{n}(x_i-y_i)$
  - the cosine measure:

$$d(X,Y)=\frac{\sum_{i=1}^{n}(x_i.y_i)}{\sqrt{\sum_{i=1}^{n}x_i}\cdot\sqrt{\sum_{i=1}^{n}y_i}}$$

  - Pearson's correlation:

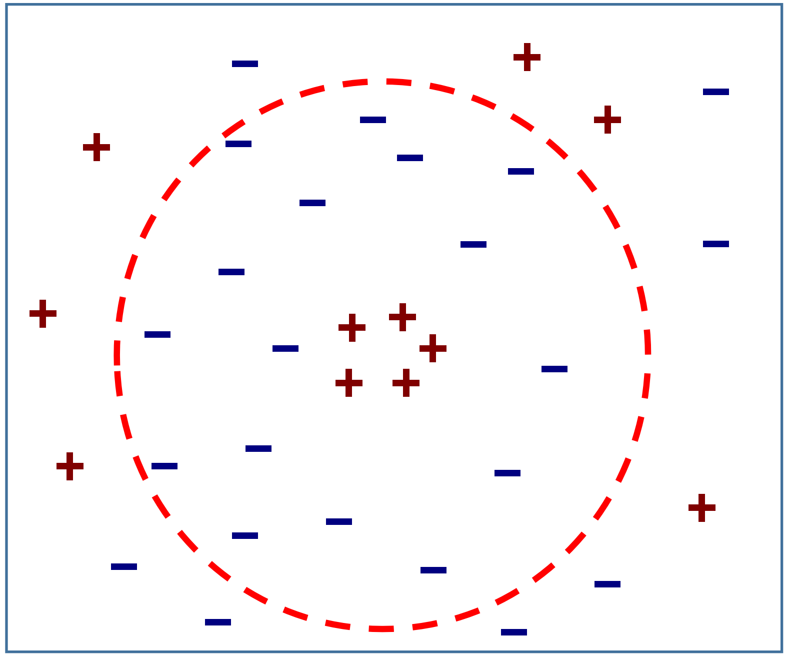
$$d(X,Y)=\frac{\sum_{i=1}^{n}(x_i-\overline{x}).(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\overline{y})^2}}$$

# Nearest Neighbor Classification

**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



# *Steps of K*-NN Algorithm

*k*-nearest neighbors algorithm steps.

**Step 1**. Determine parameter *k*= number of *nearest neighbors*

**Step 2**. Calculate the distance between the query-instance and all the training samples

**Step 3**. Sort the distance and determine *nearest neighbors* based on the *k*-th minimum distance

**Step 4**. Gather the category Y of the nearest neighbor

**Step 5**. Use simple majority of the category of *nearest neighbors* as the prediction value of the query instance

# *k*-NN Example

Question: A factory produces a new paper tissue that passes the laboratory test with $X_1 = 3$ and $X_2 = 7$. Classify this paper as *good* or *bad* using *k*-nearest neighbor method.

| Acid Durability ($X_1$) | Strength ($X_2$) | Classification (Y) |
|---|---|---|
| 7 | 7 | Bad |
| 7 | 4 | Bad |
| 3 | 4 | Good |
| 1 | 4 | Good |

## Solution

**Step1.** Determine the nearest neighbor parameter *k*. In this example we assume *k*=3

.

# *k*-NN Example

**Step2.** Calculate the distance between the query-instance and all the training examples

| Acid Durability ($X_1$ ) | Strength ($X_2$ ) | Square distance to query instance (3,7) |
|---|---|---|
| 7 | 7 | $(7-3)^2 + (7-7)^2 = 16$ |
| 7 | 4 | $(7-3)^2 + (4-7)^2 =25$ |
| 3 | 4 | $(3-3)^2 + (4-7)^2 =9$ |
| 1 | 4 | $(1-3)^2 + (4-7)^2 =13$ |

# *k*-NN Example

**Step 3**. Sort the distance and determine nearest neighbors based on the *k*-th minimum distance

| Acid Durability ($X_1$ ) | Strength ($X_2$) | Square distance to query instance (3,7) | Rank minimum distance | Is it included in 3-nearest neighbors? |
|---|---|---|---|---|
| 7 | 7 | $(7-3)^2 + (7-7)^2 = 16$ | 3 | Yes |
| 7 | 4 | $(7-3)^2 + (4-7)^2 = 25$ | 4 | No |
| 3 | 4 | $(3-3)^2 + (4-7)^2 = 9$ | 1 | Yes |
| 1 | 4 | $(1-3)^2 + (4-7)^2 = 13$ | 2 | Yes |

# *k*-NN Example

**Step 4.** Gather the category Y of the nearest neighbors.
Notice that the second row last column that the category of nearest neighbor (Y) is not included because the rank of this data is more than 3 (=*k*).

| Acid Durability ($X_1$) | Strength ($X_2$) | Square distance to query instance (3,7) | Rank minimum distance | Is it included in 3-nearest neighbors? | Y= Category of nearest neighbor |
|---|---|---|---|---|---|
| 7 | 7 | $(7-3)^2 + (7-7)^2 = 16$ | 3 | Yes | Bad |
| 7 | 4 | $(7-3)^2 + (4-7)^2 = 25$ | 4 | No | - |
| 3 | 4 | $(3-3)^2 + (4-7)^2 = 9$ | 1 | Yes | Good |
| 1 | 4 | $(1-3)^2 + (4-7)^2 = 13$ | 2 | Yes | Good |

# *k*-NN Example

**Step 5.** Use simple majority of the category of nearest neighbors as the prediction value of the query instance.

In this example we have 2 good and 1 bad, since 2>1, we conclude that a new paper tissue that pass lab test with $X_1 = 3$ and $X_2 = 7$ is classified as GOOD.

* Please try ($x_1$=2, $x_2$=6)

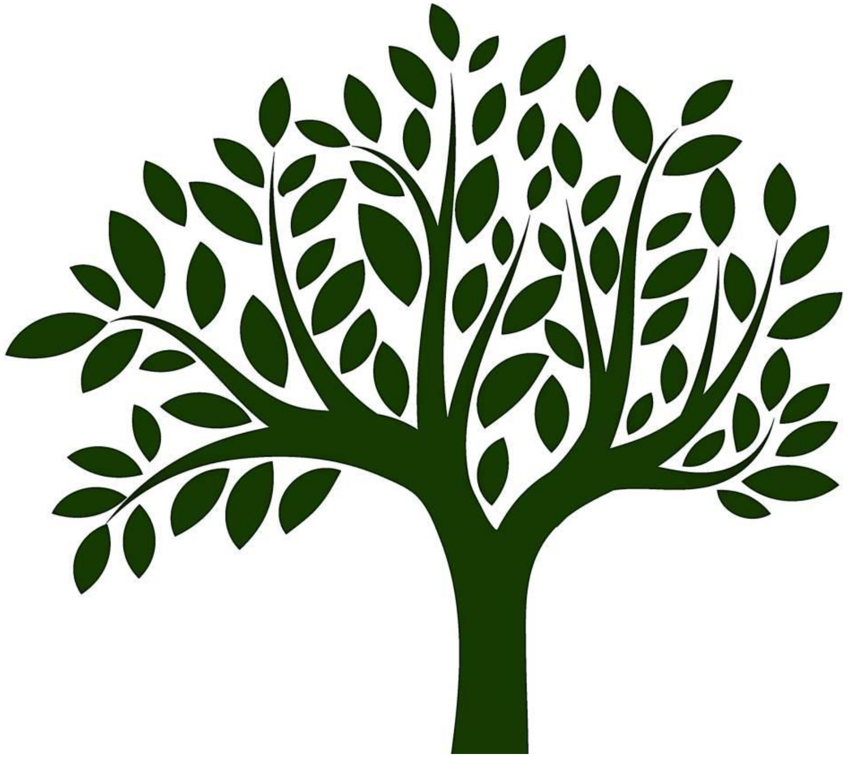
# Nearest Neighbor Classification

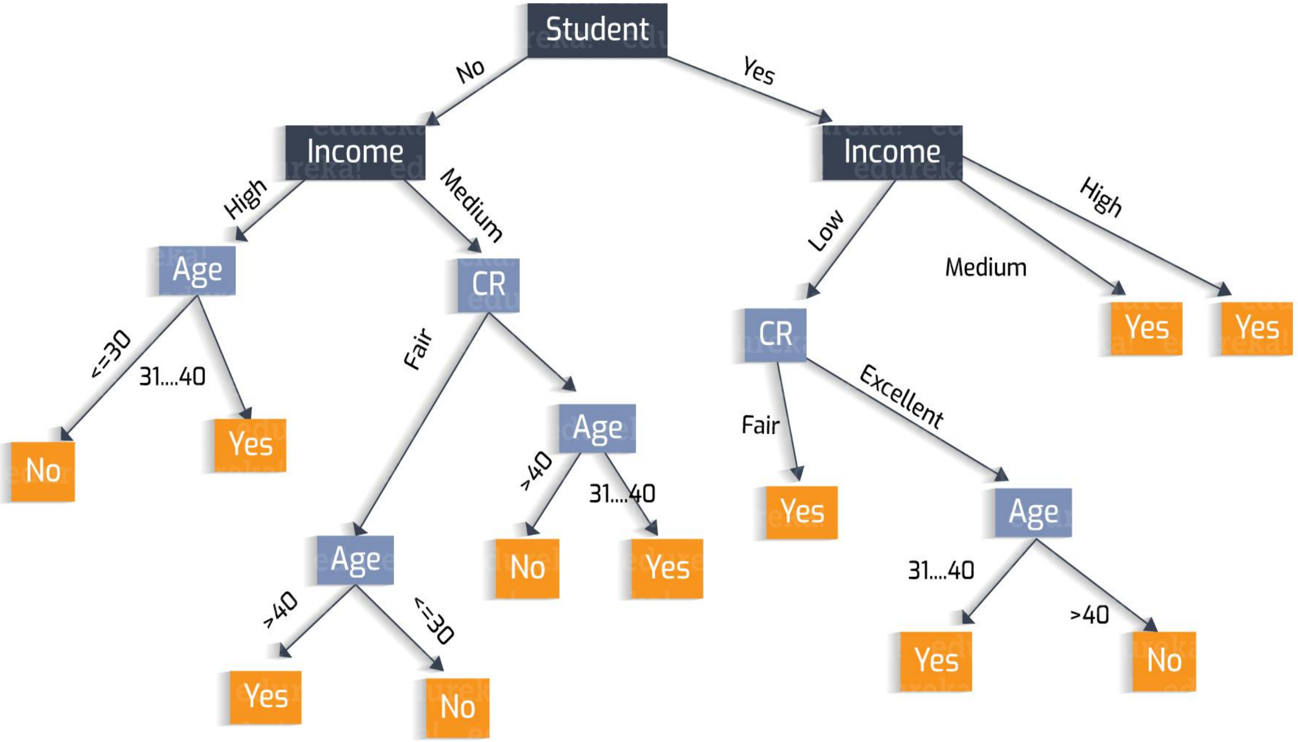
- **Advantages of KNN**
  - It is extremely easy to implement
  - Requires no training prior to making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g SVM, linear regression.
  - There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)
- **Disadvantages of KNN**
  - The KNN algorithm doesn't work well with **high dimensional data** because with large number of dimensions, it becomes difficult for the algorithm to **calculate distance in each dimension**.
  - The KNN algorithm doesn't work well with **categorical features** since it is difficult to find the distance between dimensions with categorical features.

# Common Classification Methods

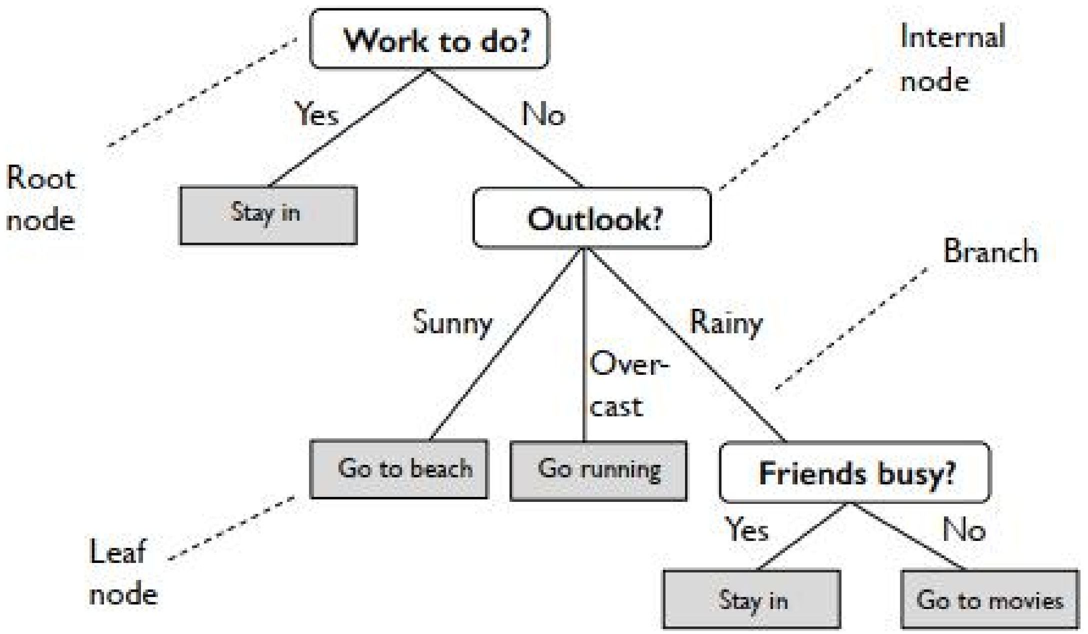
- K-Nearest-Neighbor
- Decision Tree
- Naïve Bayesian
- Support Vector Machine (SVM)
- Artificial Neural Network (ANN)

# Decision Tree





# Decision Tree



Example of a non-binary decision tree with categorical features.

## Decision tree structure

- **Root node:** beginning of a tree and represents **entire population** being analyzed.
- **Internal node:** denotes a test on an **attribute**
- **Branch:** represents an **outcome** of the test
- **Leaf nodes:** represent **class labels** or class distribution

- Tree is constructed in a top-down recursive divide-and-conquer manner

# Decision Tree

- **Root node:** has **no incoming edges** and zero or more outgoing edges.
- **Internal nodes:** each of which has exactly **one incoming edge** and two or more outgoing edges.
- **Leaf** or **terminal node**, each of which has exactly one incoming and no outgoing edges.

Solving the classification problem using DT is a two-step process:

- Decision Tree Induction- Construct a DT using training data/Induction
- For each $t_i$ in D, apply the DT to determine its class/ Deduction

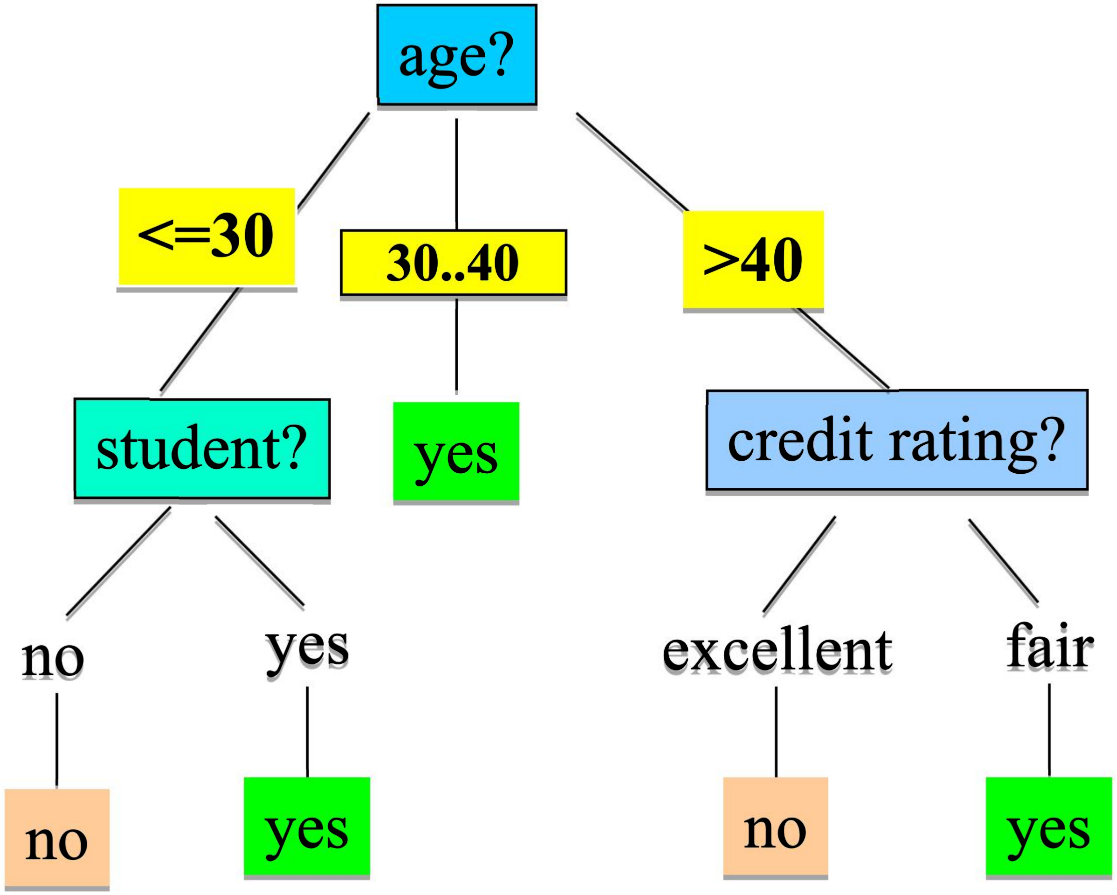
# Decision Tree-Example

## Output: A Decision Tree for “*buys_computer*”

**Training Dataset**

| 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 |

**Decision Tree**



# Decision Tree

- If there are so many possible trees, can we actually search this space? (solution: greedy search).
- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree.

- A **greedy algorithm** is a simple, intuitive **algorithm** that is used in optimization problems. The **algorithm** makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem.

# Decision Tree

## Algorithm:

- The algorithm is a greedy algorithm that doesn’t warranty optimality
- The algorithm for decision tree involves three general phases as stated below:
- **Phase I:** Find Splitting Criteria based on all the sample set at the splitting point (node) ***(attribute selection measure)***
- **Phase II:** Split all the sample data based on the splitting criteria and form branches and each successor nodes will have the samples
- **Phase III:** Do phase one and phase two iteratively until stopping criterion get fulfilled

# Decision Tree

## Attribute Selection Measure

1. Information Gain (ID3 algorithm)
2. Gain Ratio (C4.5 algorithm) [will not be discussed]
3. Gini Index (CART algorithm) [will not be discussed]

- ID3, C4.5, and CART adopt a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner using Information gain attribute selection approach.

# Decision Tree

## Information Gain (ID3)

- Developed by J. Ross Quinlan in late 1970s
- All attributes are assumed to be categorical
- Uses information gain to split node into branches
- Details of the algorithm is given below
    - Select the **attribute** with the highest **information gain**
    - This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions
    - Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.
    - The amount of information need becomes maximum if all the classes has the same number of tuples (i.e. I(D)=1)
    - The amount of information need becomes minimal if all the tuples belongs to one class (i.e I(D)= 0)

# Decision Tree

## Information Gain (ID3)

- Let $p_i$ be the probability that an arbitrary tuple in D belongs to class $C_i$, estimated by $|C_{i,\,D}|/|D_j|$, and $|D_j|$ is those tuples in D that have outcome $a_j$ of A for $i=1$ *to m, and* $j=1$ *to v.*

Expected information (entropy) needed to classify a tuple in D:

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

- info(D) is average amount of information needed to identify the class label of a tuple in D. Info(D) is also known as entropy of D.

- Information after split (after using A to split D into *v* partitions) to classify D:

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

- Information gained by branching on attribute A

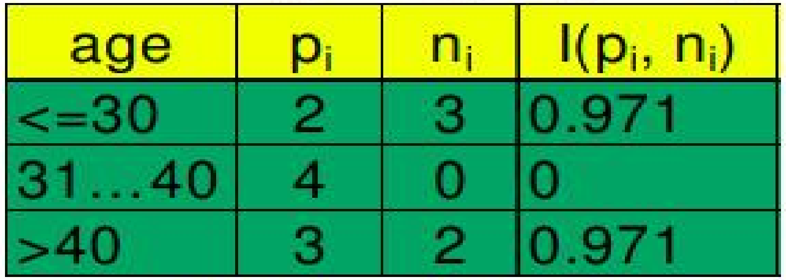
$$Gain(A) = Info(D) - Info_A(D)$$

# Decision Tree

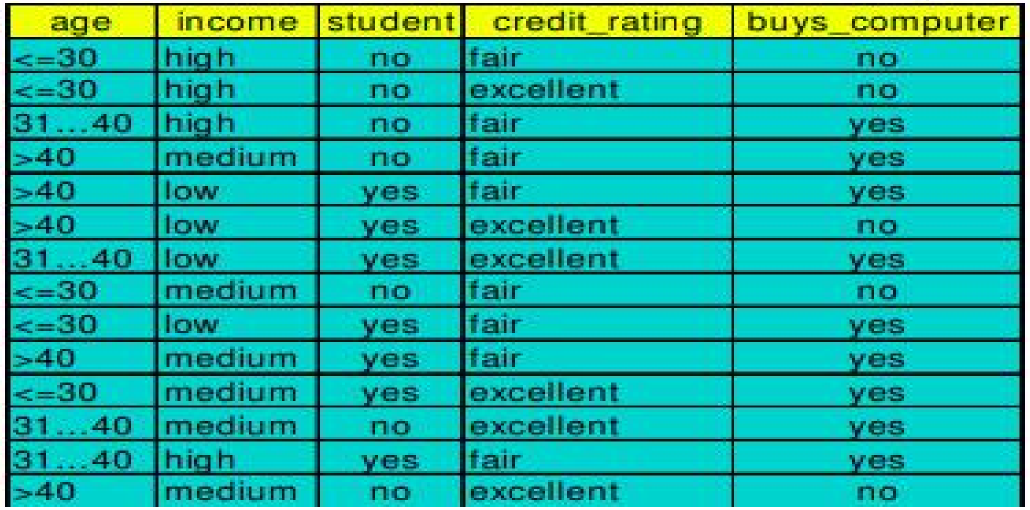
## Information Gain (ID3)-Example

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

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



| age | $p_i$ | $n_i$ | $I(p_i, n_i)$ |
|---|---|---|---|
| <=30 | 2 | 3 | 0.971 |
| 31…40 | 4 | 0 | 0 |
| >40 | 3 | 2 | 0.971 |



| 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 |

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

# Decision Tree

## Stopping Criteria

- Some conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning
  - There are no samples left in the branching direction
  - Some user defined stopping  criterion met such as
    - Depth threshold
    - Number of samples in the node becomes bellow some value
    - Or others

# Decision Tree

## Classification Rule Extraction

- Represent the knowledge in the form of IF-THEN rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

IF *age* = "<=30" AND *student* = "*no*" THEN *buys_computer* = "*no*"
IF *age* = "<=30" AND *student* = "*yes*" THEN *buys_computer* = "*yes*"
IF *age* = "31…40" THEN *buys_computer* = "*yes*"
IF *age* = ">40" AND *credit_rating* = "*excellent*" THEN *buys_computer* = "*yes*"
IF *age* = ">40" AND *credit_rating* = "*fair*" THEN *buys_computer* = "*no*"

# Decision Tree

## Strength

- In practice: One of the most popular methods. Why?
    - Very comprehensible – the tree structure specifies the entire decision structure
        - Easy for decision makers to understand model’s rational
    - Relatively easy to implement
    - Very fast to run (to classify examples) with large data sets

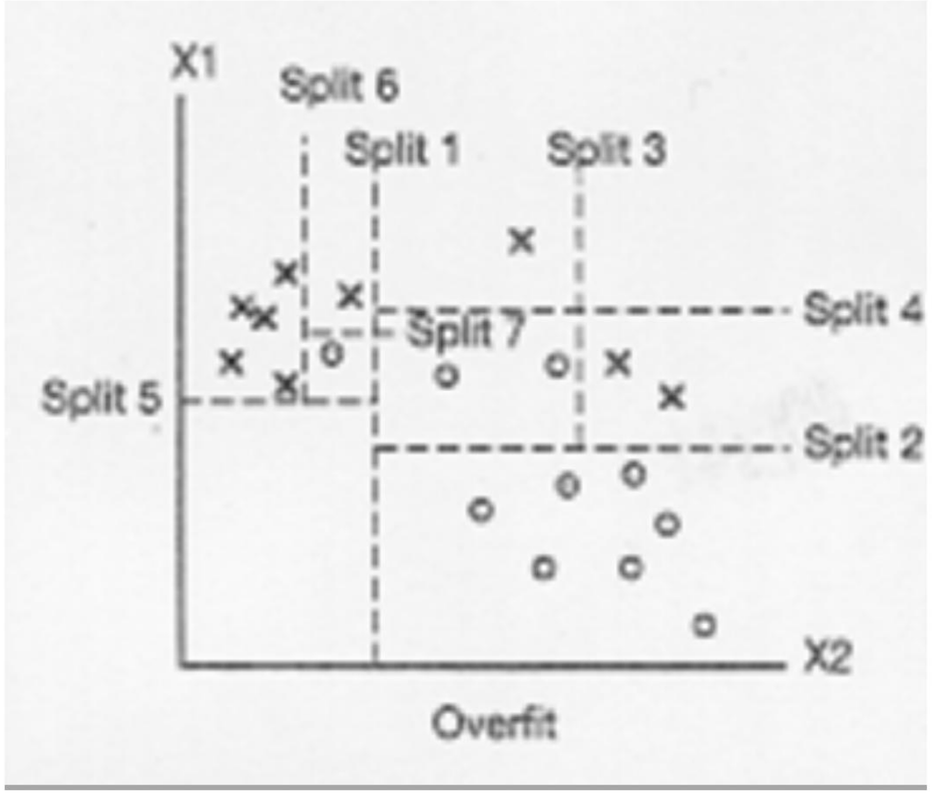
## Weakness

- Splits are simple and not necessarily optimal
- Overfitting (may not have the same performance when deployed, complex and more specific conditions)
- Underfitting (may over generalize during training, less coniditions)

# Decision Tree

## Weaknesses: Overfitting

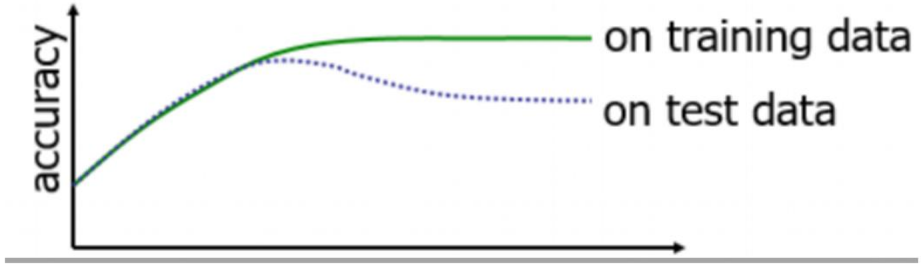
Overfitting means that the model performs poorly on new examples (e.g. *testing* examples) as it is too highly trained to the specific (non-general) nuances of the *training* examples.



# Decision Tree

## Weaknesses: Overfitting

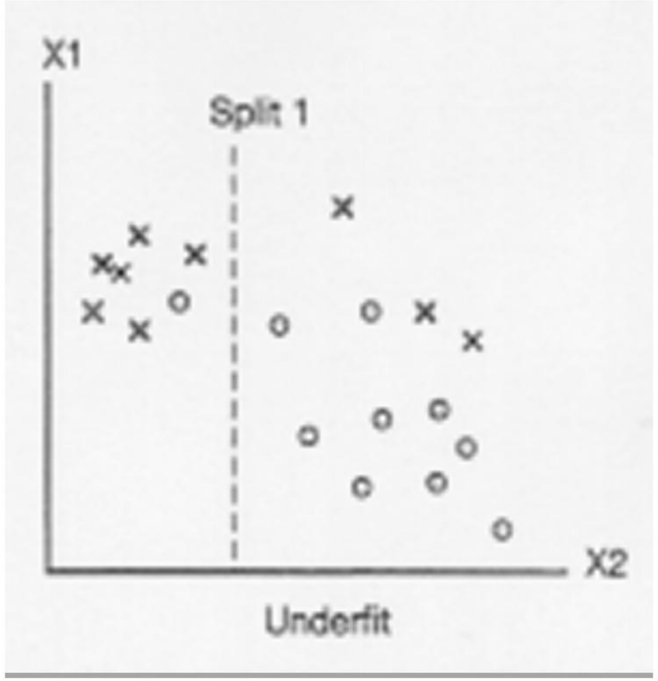
- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  - There may be noise in the training data that the tree is erroneously fitting.
  - The algorithm may be making poor decisions towards the leaves of the tree that are based on very little data and may not reflect reliable trends.
- A hypothesis, *h*, is said to overfit the training data if there exists another hypothesis *h´* such that *h* has less error than *h´* on the training data but greater error on independent test data.



# Decision Tree

## Weaknesses: Underfitting

Underfitting means that the model performs poorly on new examples as it is too simplistic to distinguish between them (i.e. has not picked up the important patterns from the training examples)



# Bayesian Classification

$$P(h \mid X) = \frac{P(X \mid h)P(h)}{P(X)}$$

Thomas Bayes 1702-1761

# Bayesian Classification

- Bayesian classifiers are statistical classifiers.
- They can predict class membership probabilities, such as t**he probability** that a **given tuple belongs** to a particular class.

- Bayesian classification is based on **Bayes' theorem**

$$P(h|X) = \frac{P(X|h)P(h)}{P(X)}$$

where h is hypothesis, X is a training data

- Compared to Decision tree, Byesian classifiers have also exhibited high accuracy and speed when applied to large databases.

# Bayesian Classification

## Baye’s Theroem

- Let **X** be an observation (vector of values of the independent variables) whose class label (dependent variable) is not given
- Let **h** be a hypothesis that says the class of **X** is **C**
- The classification problem require to accept or reject the hypothesis

- This can be done in Bayesian classification by finding the *posteriori probability of a hypothesis* $P(h|X)$

- Bayes theorem says $P(h|X)=\frac{P(X|h)P(h)}{P(X)}$

# Naïve Bayesian Classification

1) Let D be a training set of tuples and their associated class labels, and each tuple is represented by an *n*-dimensional feature vector $X = (x_1, x_2, \ldots, x_n)$, depicting *n* measurements made on the tuple from *n* attributes, $A_1, A_2, \ldots A_n$, respectively.
2) And suppose there are *m* classes $C_1, C_2, \ldots, C_m$. Now, given a tuple X, the classifier predict that X belongs to the class having highest posterior probability ( conditioned on X.)
3) That is, classifier assigns an unknown sample X to the class $C_i$, if and only if $P(C_i|X) > P(C_j|X)$ *for* $1 \leq j \leq m, j \neq i$ . The class $C_i$ for which $P(C_i|X)$ is maximized is called the ***maximum posteriori hypothesis.***

This can be derived from Bayes" theorem:

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

4) Since P(X) is constant for all classes, only $P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$ needs to be maximized. If prior probability of X unknown, classes are assumed to be equally likely, i.e., $P(C_1) = P(C_2) = P(C_3\ ) =\ldots = P(C_m\ )$ and would maximize $P(X|C_i\ )$, where $P(C_i\ ) = |\ C_{i,D}\ |/\ |D|$.

# Naïve Bayesian Classification

5) A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

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

6) This greatly reduces the computation cost:-only counts the class distribution

7) If $A_k$ is categorical, $P(x_k|C_i) = s_{ik}/s_{i,}$ is the # of tuples of $C_i$ in D having value $x_k$ for $A_k$ divided by $|C_{i, D}|$ (# of tuples of $C_i$ in D)

s

8) If $A_k$ is continuous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

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

and $P(x_k|C_i)$ is $P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$

9) In order to classify an unknown sample *X*, $P(X|C_i)P(C_i)$ is evaluated for each class $C_i$. X is assigned to the class $C_i$ iff, $P(X|C_i)P(C_i) > P(X|C_j)\ P(C_j)$ *for* $1 \le j \le m, j \ne i$.

# Naïve Bayesian Classification

## Example:

Class:

C1:buys_computer ="yes‘

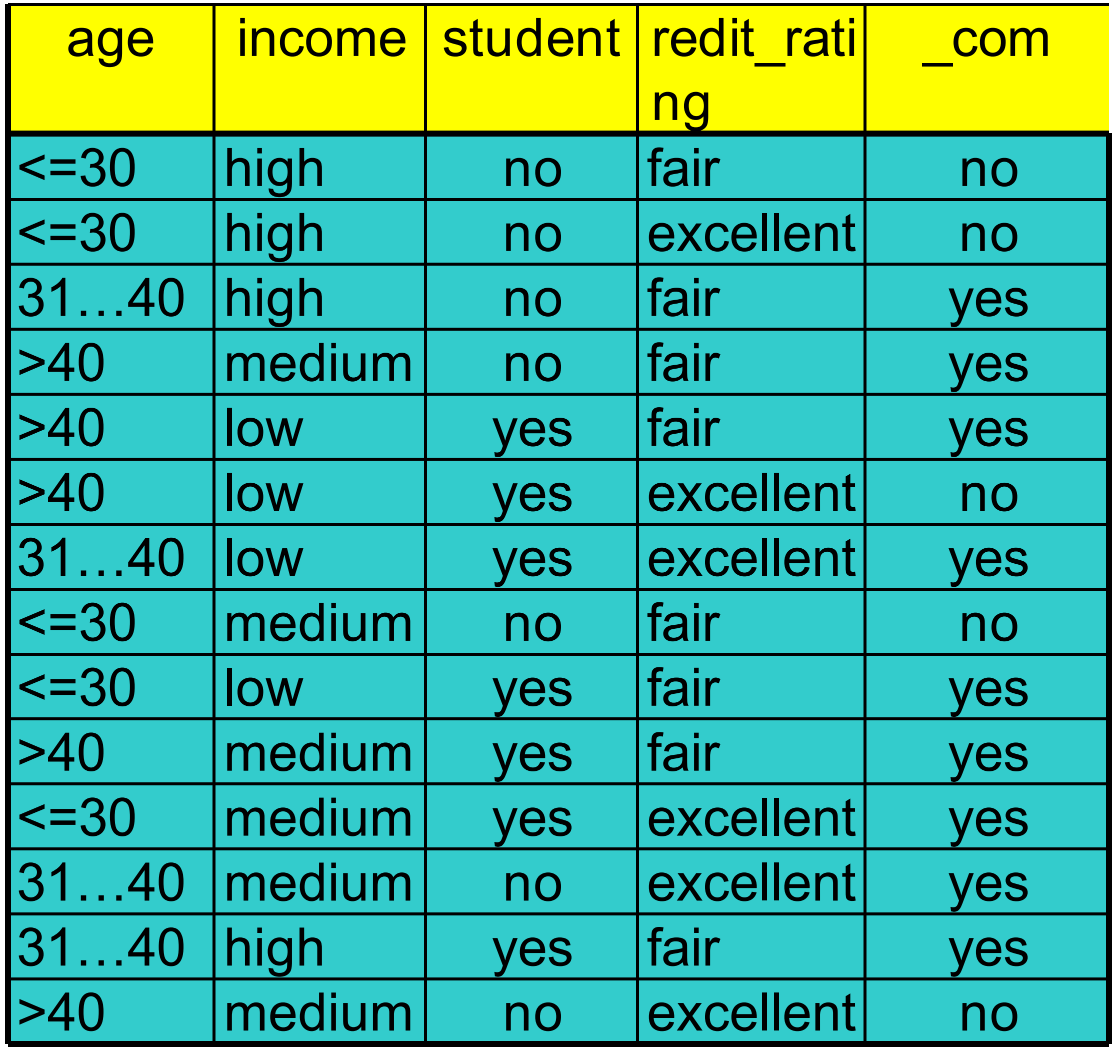
C2:buys_computer = "no‘

Given a data sample X:

X = (age <=30, Income = medium, Student = yes, Credit_rating = Fair)

Task:

Classify X using Bayesian classifier



| age | income | student | redit_rati ng | _com |
|---|---|---|---|---|
| <=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 |

# Naïve Bayesian Classification

**Example:**

We need to maximize *P(X|Ci)P(Ci)*, for i=1,2.

P($C_i$): P(buys_computer = “yes”) = 9/14 = 0.643
P(buys_computer = “no”) = 5/14= 0.357

Compute P(X|$C_i$) for each class:

P(age = “<=30” | buys_computer = “yes”) = 2/9 = 0.222
P(age = “<= 30” | buys_computer = “no”) = 3/5 = 0.6

P(income = “medium” | buys_computer = “yes”) = 4/9 = 0.444
P(income = “medium” | buys_computer = “no”) = 2/5 = 0.4

P(student = “yes” | buys_computer = “yes) = 6/9 = 0.667
P(student = “yes” | buys_computer = “no”) = 1/5 = 0.2

P(credit_rating = “fair” | buys_computer = “yes”) = 6/9 = 0.667
P(credit_rating = “fair” | buys_computer = “no”) = 2/5 = 0.4

| age | income | student | redit_rating | _com |
|---|---|---|---|---|
| <=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 |

# Naïve Bayesian Classification

**Example:**

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

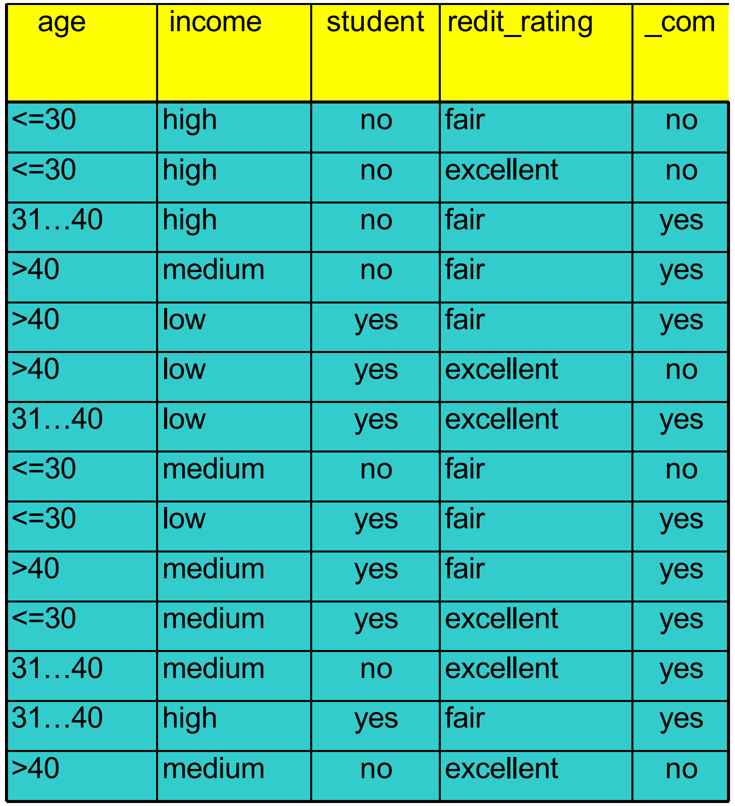
- $P(X|C_i)$ : P(X|buys_computer = “yes”)
  = 0.222 x 0.444 x 0.667 x 0.667
  = 0.044

- P(X|buys_computer = “no”)
  = 0.6 x 0.4 x 0.2 x 0.4
  = 0.019

- $P(X|C_i)*P(C_i)$ : P(X|buys_computer = “yes”) * P(buys_computer = “yes”)
  = 0.044 * 0.643
  =0.028

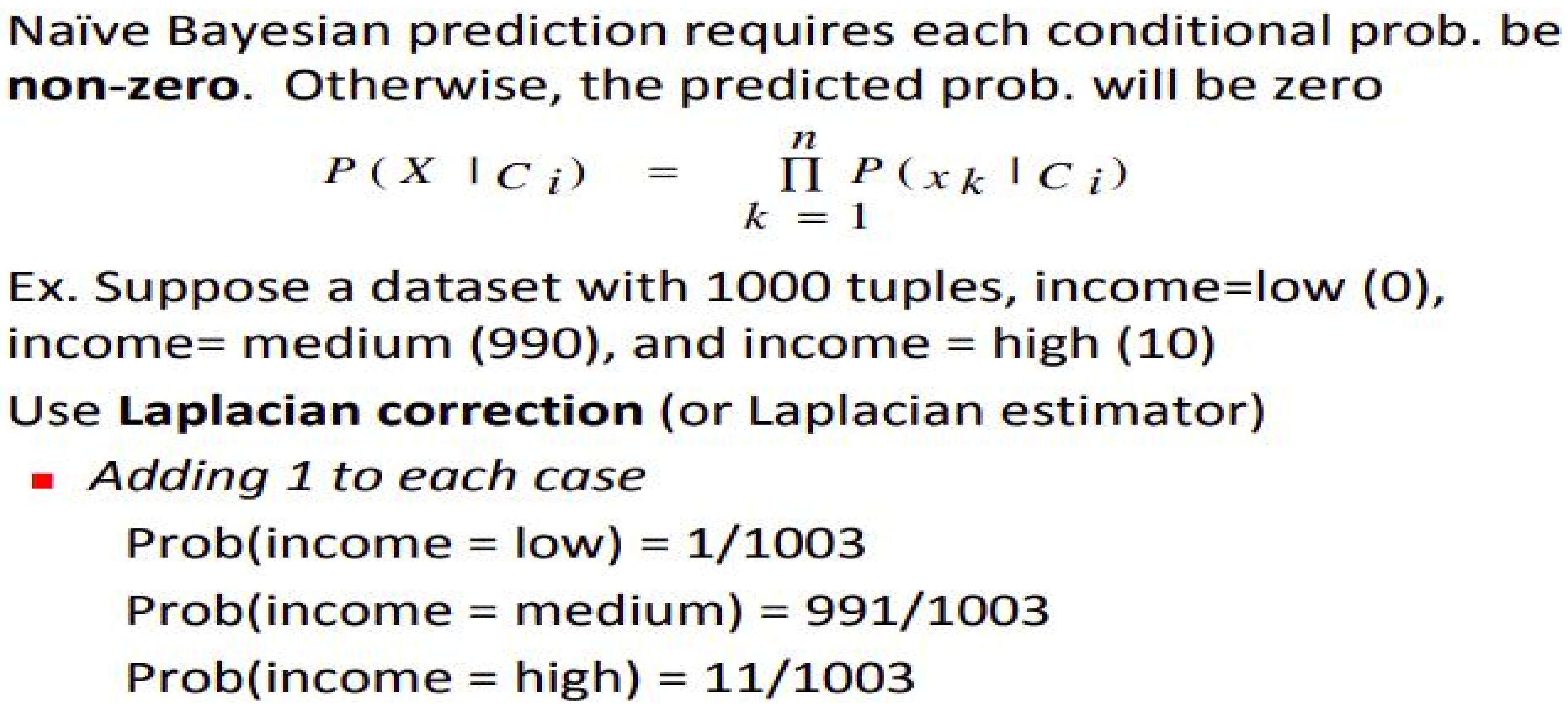
- P(X|buys_computer = “no”) * P(buys_computer = “no”)
  = 0.019 * 0.357
  =0.007

- Since 0.028 > 0.007, X belongs to class (“buys_computer = yes”)



| age | income | student | redit_rating | _com |
|---|---|---|---|---|
| <=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 |

# Naïve Bayesian Classification



Naïve Bayesian prediction requires each conditional prob. be **non-zero**. Otherwise, the predicted prob. will be zero

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

Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)

Use **Laplacian correction** (or Laplacian estimator)

- *Adding 1 to each case*

  Prob(income = low) = 1/1003

  Prob(income = medium) = 991/1003

  Prob(income = high) = 11/1003

# Naïve Bayesian Classification

## Naïve Bayes Classifier: Strength

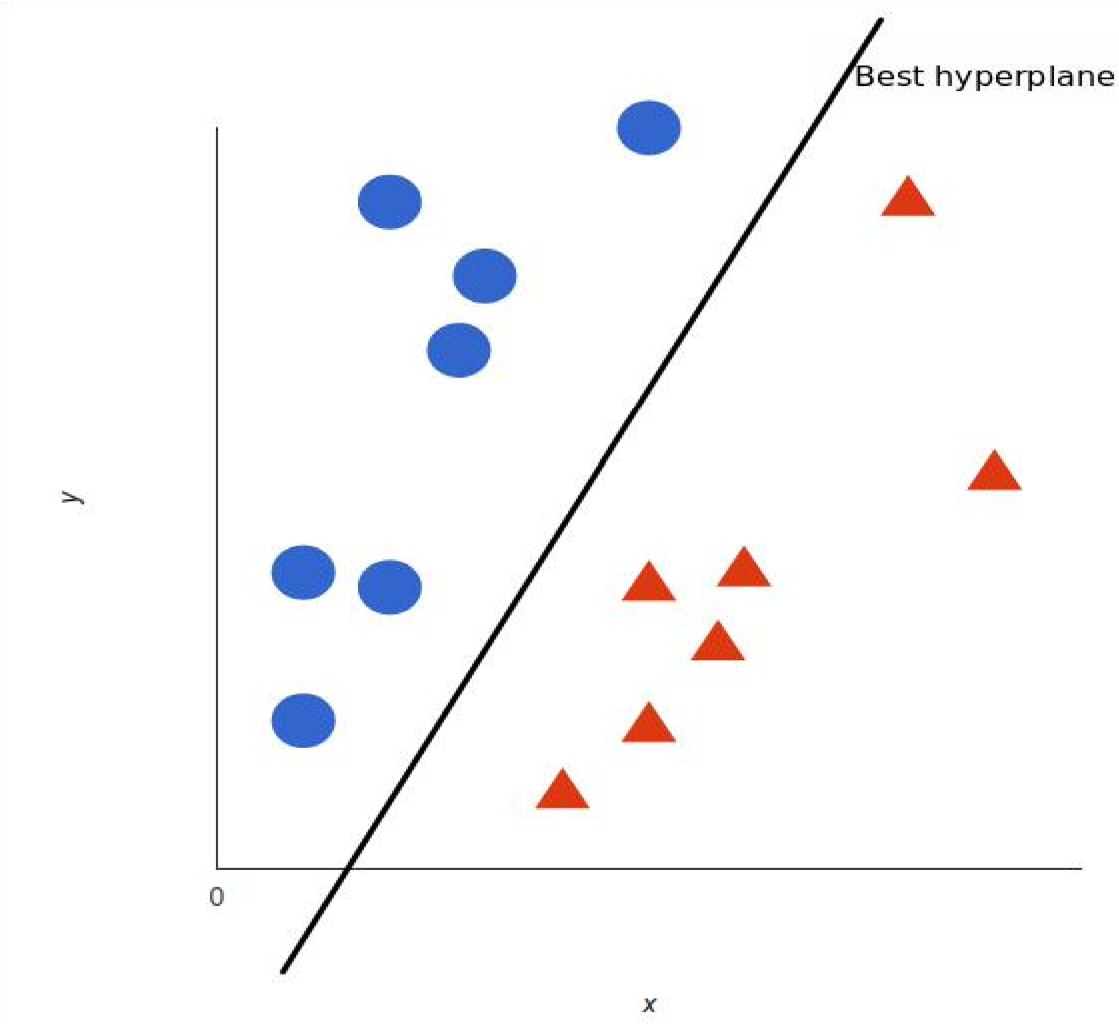
- Simple
- Incremental learning
- Naturally a probability estimator
- Easily handles missing values

# Naïve Bayesian Classification

## Naïve Bayes Classifier: Weakness

- Independence assumption
- Categorical/discrete attributes
- Sensitive to missing values

# Support Vector Machine



# Support Vector Machine

- SVM is a geometric model that views the input data as **two sets of vectors** in an *n*-dimensional space.

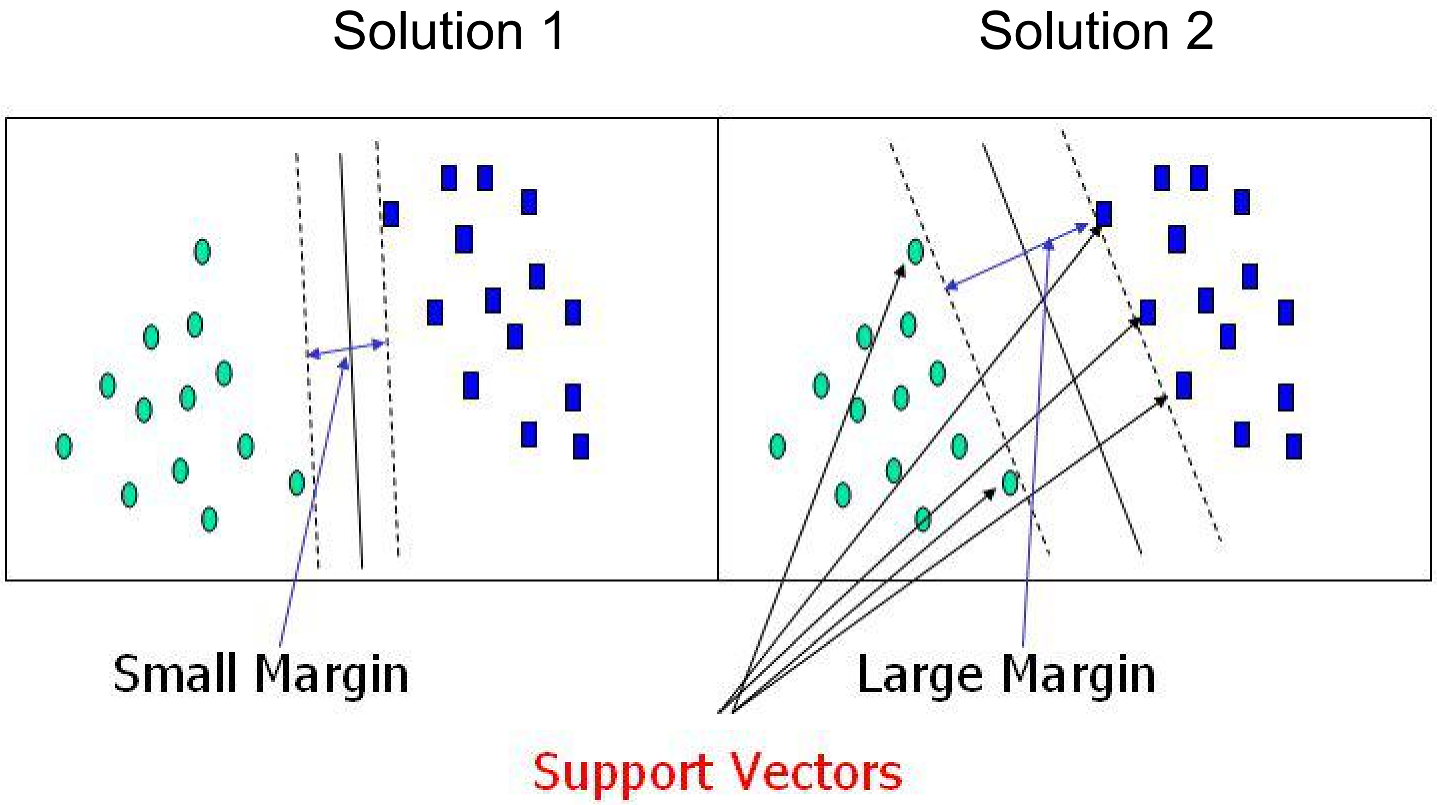
- It constructs a **separating hyperplane** in that space, one which maximizes the *margin* between the two data sets.

- To calculate the margin, **two parallel hyperplanes** are constructed, one on each side of the separating hyperplane.

- A good separation is achieved by the hyperplane that has the **largest distance** to the neighboring data points of both classes.

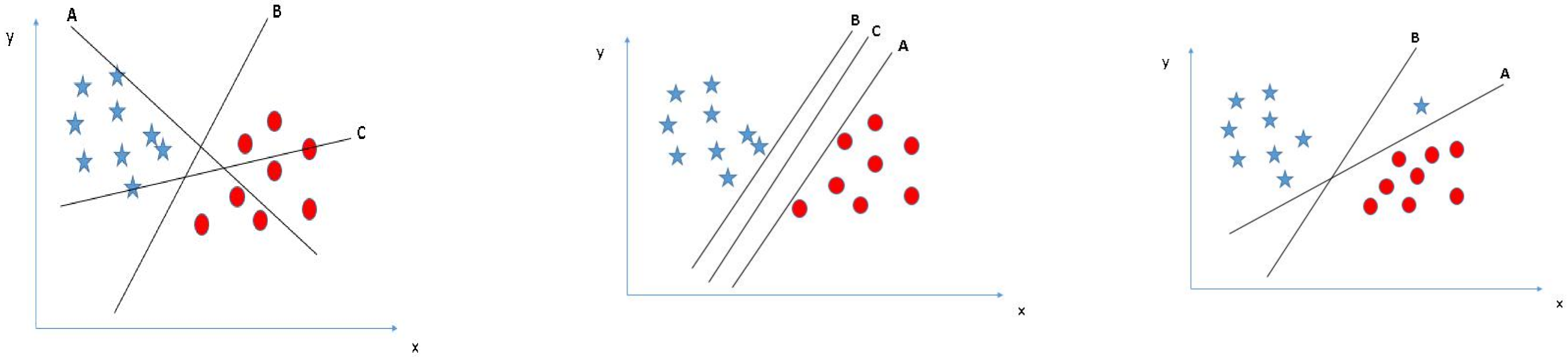
- The vectors (points) that constrain the width of the margin are the ***support vectors*.**

# Support Vector Machine



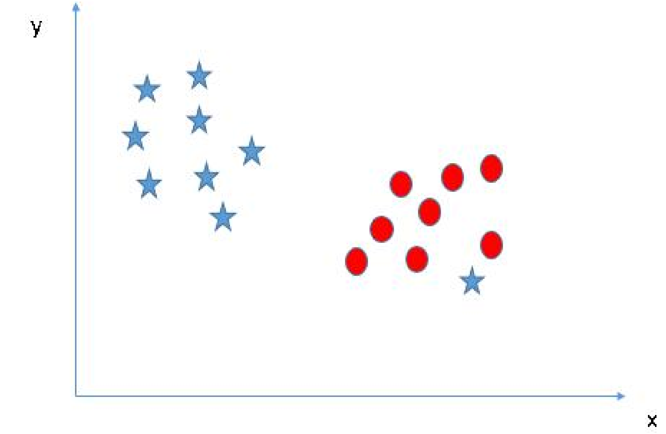
An SVM analysis finds the line (or, in general, hyperplane) that is oriented so that the margin between the support vectors is maximized. In the figure above, Solution 2 is superior to Solution 1 because it has a larger margin.

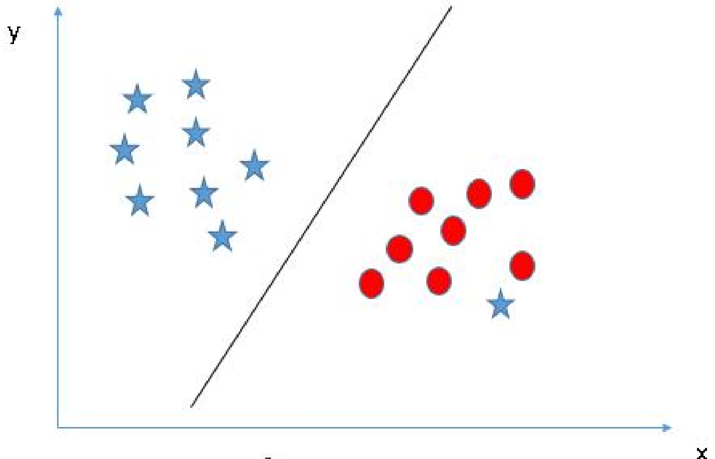
# Support Vector Machine



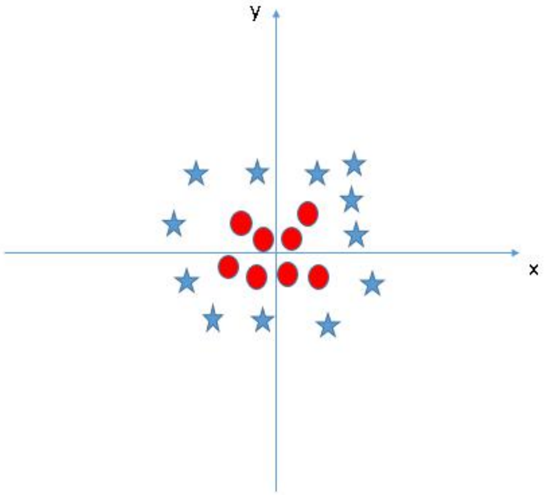
- To separate the two classes of data points:
  - There are many possible hyperplanes that could be chosen.
  - Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes.
  - Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

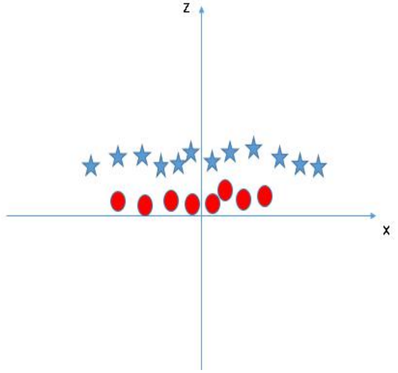
# Support Vector Machine



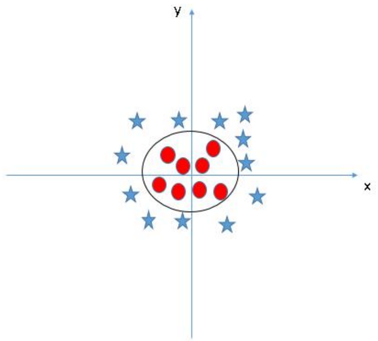


SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin





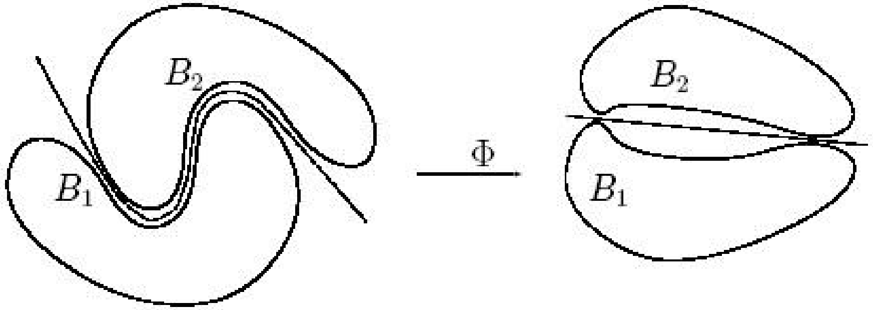
Add a new feature z=x$^2$+y$^2$ and then plot the data points on axis x and z



**kernel trick**. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem

# Support Vector Machine – Kernel Functions

- Kernel function Φ: map data into a different space to enable linear separation.



- Convert linear learning machines (such as linear SVM) into non-linear ones (such as kernel SVM) by doing an inner product between datapoints.
- Kernel functions are very powerful. They allow SVM models to perform separations even with very complex boundaries.
  - Some popular kernel functions are linear, polynomial, and radial basis.
  - While you can construct your own kernel functions according to the data structure, ML tool provides a variety of built-in kernels. e.g Scikit-learn

# Support Vector Machine – Multi-class classification

- Suppose we have a bunch of source code which are going to be classified in 3 programming languages. Do you think the binary classifier can solve this problem?

```
for (int i = 0; i < 5; i++)
{
  cout << i << "\n";
}
```

1

```
for x in range(0,5):

   print(x)
```

2

```
for (int i = 0; i < 5; i++)
{
  cout << i ;
}
```

3

```
for (int i = 0; i < 5; i++)
{
  System.out.println(i);
}
```

4

- One-Vs-Rest for Multi-Class Classification: distinguishing between some label and all the others, where the class prediction with highest probability wins.
- One-Vs-One for Multi-Class Classification: a classifier is trained for every pair of classes, making continuous comparisons. The class prediction with highest quantity of predictions wins.

# Support Vector Machine – Multi-class classification

- Suppose we have a bunch of source code which are going to be classified in 3 programming languages. Do you think the binary classifier can solve this problem?

```
for (int i = 0; i < 5; i++)
{
  cout << i << "\n";
}                                   1
```

```
for x in range(0,5):

   print(x)
                                   2
```

```
for (int i = 0; i < 5; i++)
{
  cout << i ;
}                                   3
```

```
for (int i = 0; i < 5; i++)
{
  System.out.println(i);
}                                   4
```

- In case of OvR: creating  C binary classifiers for C number of class, where each binary classifier should predict a class probability as p1, ... and $p_C$. Then the prediction will be argmax(p1, p2, p3)
  - binary classifier 1: java vs {c++, python}
  - binary classifier 2: c++ vs {java, python}
  - binary classifier 3: python vs {c++, java}

- if the probability that source code #4 is Java vs c++ or python is 0.6, c++ vs java or python is 0.15, python vs c++ or java is 0.25 then then argmax (0.6, 0.15, 0.25)=0 which is JAVA

# Support Vector Machine – Multi-class classification

- Suppose we have a bunch of source code which are going to be classified in 3 programming languages. Do you think the binary classifier can solve this problem?

```
for (int i = 0; i < 5; i++)
{
  cout << i << "\n";
}
```

1

```
for x in range(0,5):

   print(x)
```

2

```
for (int i = 0; i < 5; i++)
{
  cout << i ;
}
```

3

```
for (int i = 0; i < 5; i++)
{
  System.out.println(i);
}
```

4

- In case of OvO: creating set of binary classifiers in (C(C-1)/2) for C number of class, each representing one of the pairs.  Then the class that is picked the most is the winner
  - binary classifier 1: java vs c++
  - binary classifier 2: java vs  python
  - binary classifier 3: python vs c++
- if source code #4 is  classified as Java  in the first and the scond case, it is automatically JAVA

- Practical Examples

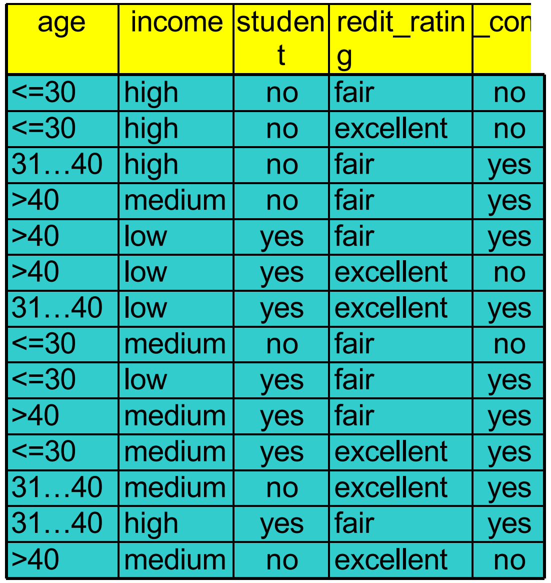
# Support Vector Machine – Example

```
from sklearn import svm
import numpy as np
x_train=np.array([[4],[5],[6],[7],[8]])
y_train=np.array([4,5,6,7,8])
est = svm.LinearSVC()
  #training
Model=est.fit(x_train, y_train)
  #prediction
pred = est.decision_function([[-5], [6], [7],[8]])
```

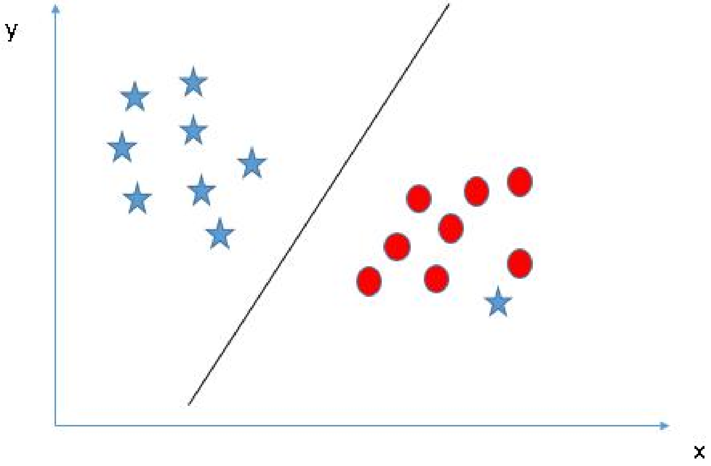
* If none of the predictions get more than half of the votes, we may say that the ensemble method could not make a stable prediction for this instance.

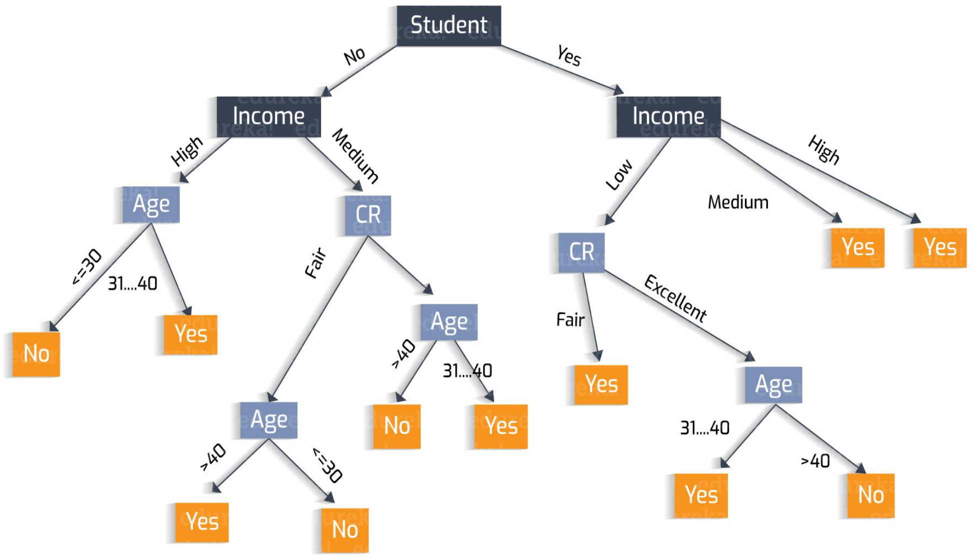
# Classification Algorithm: Others

- Please review the following methods
  - **Classification**
    - Bayesian Belief Networks
    - case-based reasoning
    - Genetic algorithm and programming
    - Fuzzy set approaches
  - **Regression**
    - Linear regression
    - Ridge regression



| age | income | student | redit_rating | _con |
|---|---|---|---|---|
| <=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 |





$$P(h|X) = \frac{P(X|h)P(h)}{P(X)}$$

```
for (int i = 0; i < 5; i++)
{
  cout << i << "\n";
}                                   1
```

```
for x in range(0,5):

   print(x)
                                    2
```

```
for (int i = 0; i < 5; i++)
{
  cout << i ;
}                                   3
```

```
for (int i = 0; i < 5; i++)
{
  System.out.println(i);
}                                   4
```