Machine Learning Tutorial
Part 1
[Machine Learning Techniques]

Inha University

Prof. Sang-Jo Yoo

sjyoo@inha.ac.kr

Contents (Part 1)

- **Introduction**
- **Linear Regression**
  - Linear Regression
  - Polynomial Regression
  - Least Square Estimator
  - Gradient Decent Algorithm
- **Tree-based Classification and Regression**
  - Decision Tree
  - Random Forest
  - Ensemble Techniques
- **Unsupervised Clustering**
  - Clustering
  - K-means Algorithm
  - Density-based Clustering (DBSCAN)
  - Gaussian Mixture Model (GMM)
  - Expectation Maximization (EM)
  - Hierarchical Clustering (agglomerative)
- **Naïve Bayes Classifier**
  - Bayes Theorem
  - Naïve Bayes Classifier
- **Support Vector Machine**
  - SVM (linear)
  - Decision Rule
  - Soft Margin SVM
  - Kernel SVM (non-linear)

Contents (Part 2)

- **Neural Network Basics**
  - Linear Regression
  - Gradient Descent Algorithm
  - Logistic Regression (sigmoid)
  - SoftMax Regression
- **Deep Neural Network**
  - Multi-Layer Perceptron
  - Feed Forwarding
  - Backpropagation
  - Gradient Vanishing Problem
  - Tanh/ReLu/Maxout
- **Optimization for Training Deep Models**
  - Data Normalization
  - Parameter Updating (SGD, Mini-batch GD, Batch GD)
  - Gradient Descent with Momentum
  - Regularization
  - Parameter Initialization

Contents (Part 3)

- **Convolutional Neural Network**
  - Background
  - Convolution Layer
  - Padding
  - Pooling Layer
  - CNN Structures
- **Generative Adversarial Network**
  - Generative and Discriminative Models
  - Objective Function of GAN
  - Applications of GAN
- **Recurrent Neural Network**
  - Motivations
  - RNN Model Sequences
  - Vanilla RNN Architecture
  - Long Short Term Memory (LSTM)
- **Reinforcement Learning**
  - Reinforcement Learning Model
  - Q-Learning Algorithm
  - Deep Reinforcement Learning (DQN)

Contents (Part 4)

- **Cognitive Radio and Machine Learning**
  - Cognitive radio
  - Parameters for learning and optimization
  - Learning in CR
- **Wireless Sensor Network and Internet of Things using Machine Learning**
  - Wireless Sensor Network
  - WSN and Machine Learning
  - Clustering and Data Aggregation
  - Event Detection
  - Localization
  - Routing and MAC
- **Wireless Network Research Activities (Mnet, Inha Univ) Using Machine Learning**
  - WSN
  - Ad-hoc Network
  - Cognitive Radio

Contents (Part 5)

- **TensorFlow Install**
  - Anaconda
  - Jupyter Notebook
  - CUDA
  - TensorFlow
- **TensorFlow Basics**
  - TensorFlow Mechanism
  - Tensor Manupulation
  - NumPy
  - Python for Loops
- **Regression Programing**
  - Linear Regression
  - Cost Minimization
  - Multi-variable Linear Regression
- **Classification Programing**
  - Binary Classification (Sigmoid)
  - Multinomial Classification (Softmax)
- **Neural Network and MNIST Programing**
  - Learning Rate
  - Preprocessing
  - MNIST Case Study
- **Multi-layer (Deep) Neural Network Programing**
  - XOR Example
  - DNN MNIST Case Study
  - DNN Case Study with Dropout and tf.layers.dense()

Contents (Part 5)

- **Convolutional Neural Network (CNN) Programing**
  - CNN Functions
  - MNIST CNN Classifier
  - CNN Ensemble Programing
- **Recurrent Neural Network (RNN) Programing**
  - Implementing RNN in Tensorflow
  - RNN Programing #1
  - RNN Programing #2 (Multi Layer RNN)
- **Reinforcement Learning Programing**
  - Q-learning Programing
  - DQN Programing

Machine Learning Introduction

AI/Machine Learning/Deep Learning

- **Artificial Intelligence:**
  - AI is intelligence exhibited by machines. In computer science, an ideal "intelligent" machine is a flexible rational agent that *perceives its environment and takes actions that maximize its chance of success at some goal*.
  - goals) of AI research include reasoning, knowledge, planning, learning, natural language processing, perception and the ability to move and manipulate objects
- **Machine Learning**
  - Machine learning is a type of artificial intelligence (AI) that provides *computers with the ability to learn without being explicitly programmed*.

AI/Machine Learning/Deep Learning

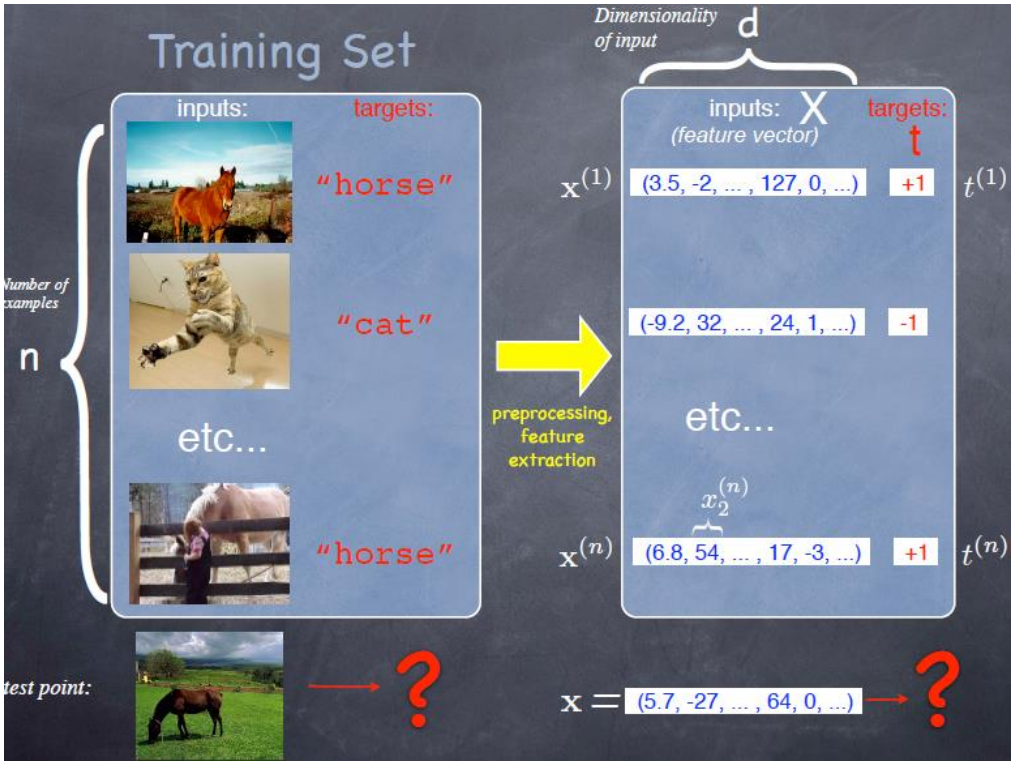
- **Deep Learning**
  - is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data by using a *deep graph with multiple processing layers, composed of multiple linear and non-linear transformations*.

Machine Learning Tasks

- **Supervised learning** = predict target **t** from input **x**
  - Learning with labeled (image labeling)
  - **t** represents a category or "class" → classification (binary or multiclass)
  - **t** is a real value → regression
- **Unsupervised learning**: no explicit target **t**
  - model the distribution of **x** → density estimation
  - capture underlying structure in **x** → dimensionality reduction, clustering, etc... (e.g., Google news grouping)
- **Reinforcement learning**
  - interacts with a dynamic environment in which it must perform a certain goal without explicitly telling it whether it is close to its goal.

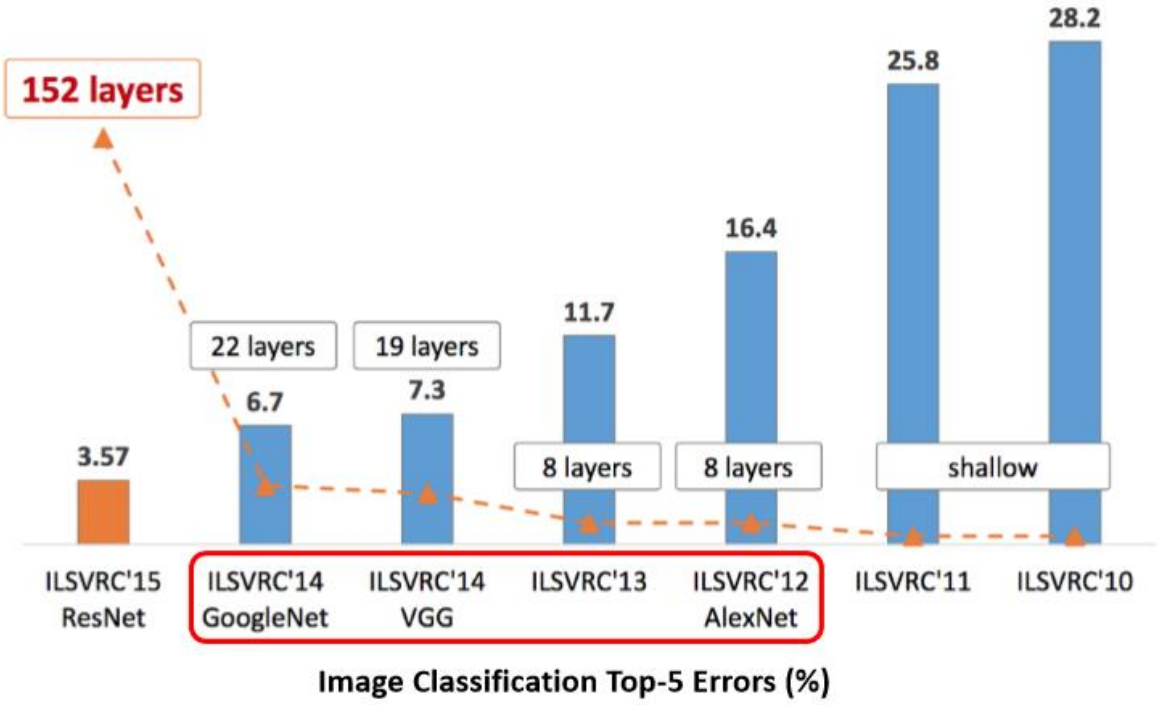
Deep Learning

- **'Deep Learning'**
  - means using a neural network with several layers of nodes between input and output.
  - The series of layers between input & output do feature identification and processing in a series of stages, just as our brains seem to.

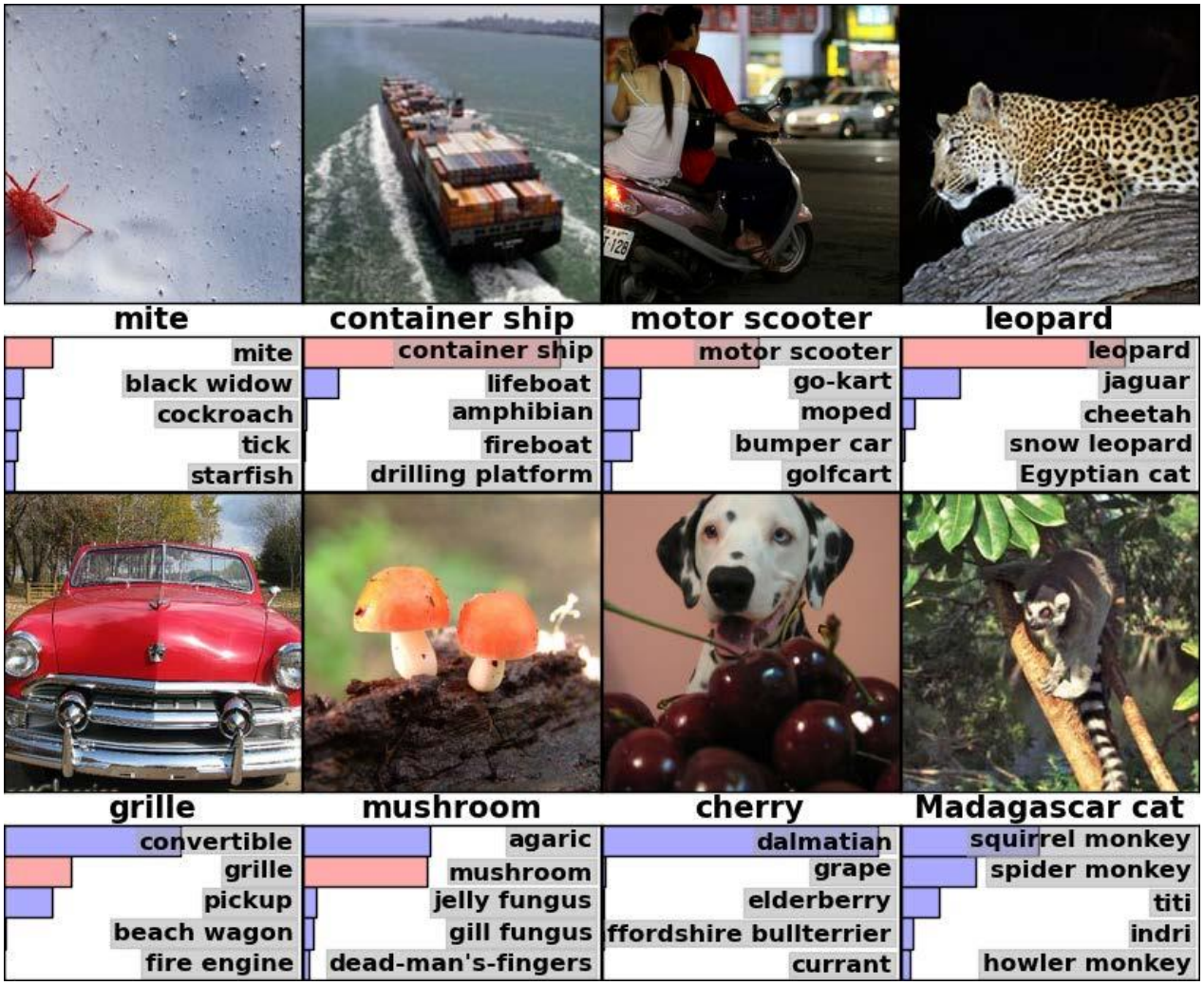


Deep Learning

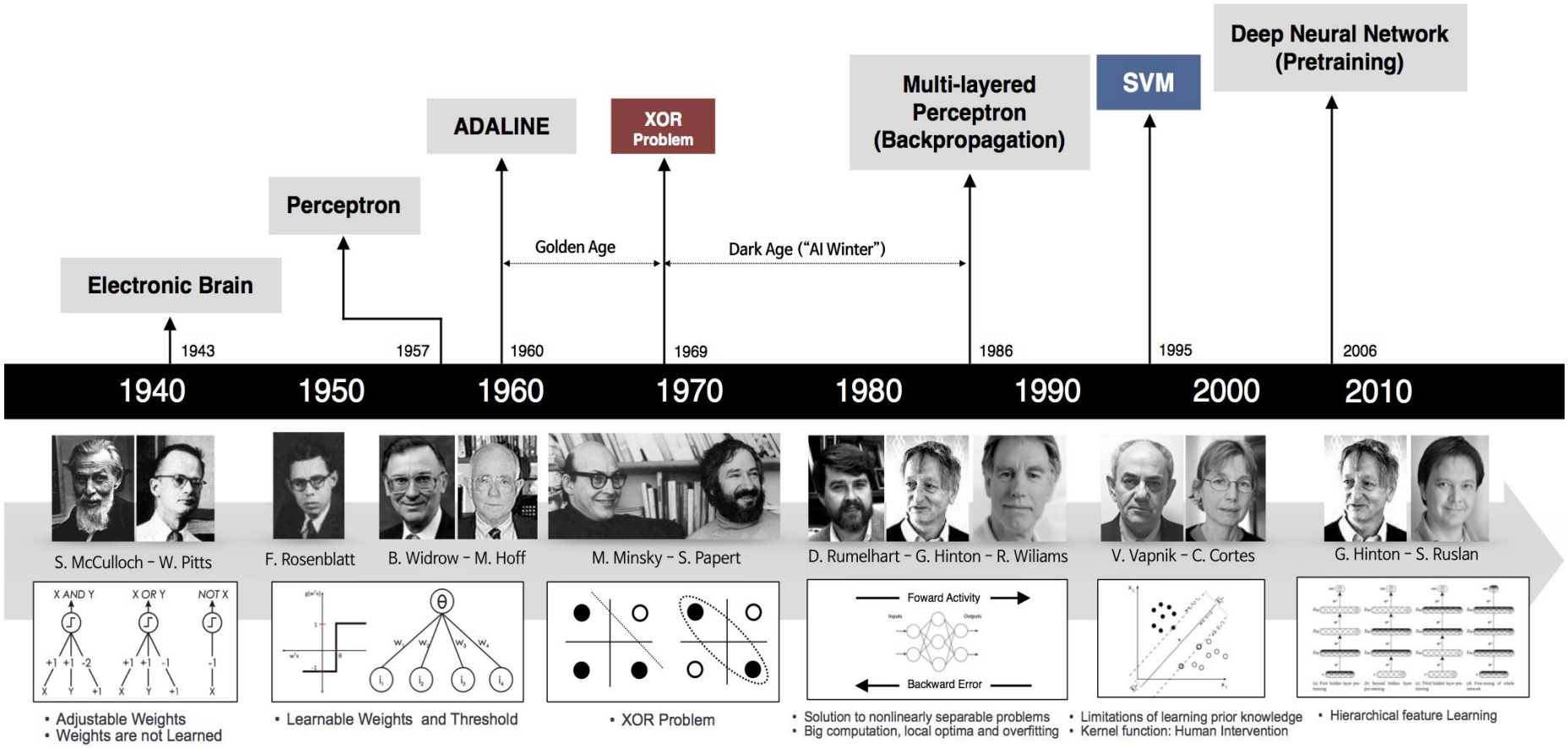
- **ImageNet Large Scale Visual Recognition Competition (ILSVRC)**
  - Human= about 5% error rate



Deep Learning



History of Artificial Intelligent



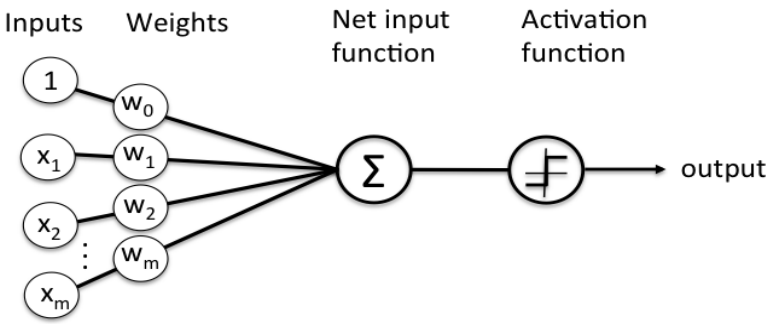
History of Artificial Intelligent

- **Frank Rosenblatt: "perceptron"1957**
  - An electronic device which was constructed in accordance with biological principles and showed an ability to learn.
  - Rosenblatt was so confident that the perceptron would lead to true AI.
  - Also provided a learning algorithm.

$$W = W + \eta(d(X) - y(X))X$$

  - $d(X)$: $output\ of\ perceptron$, $y(X)$: $true\ output\ of\ X$

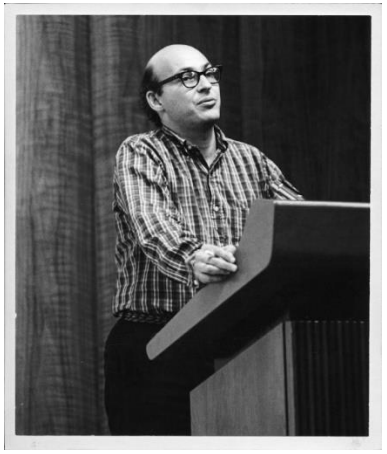
https://www.youtube.com/watch?v=cNxadbrN_aI



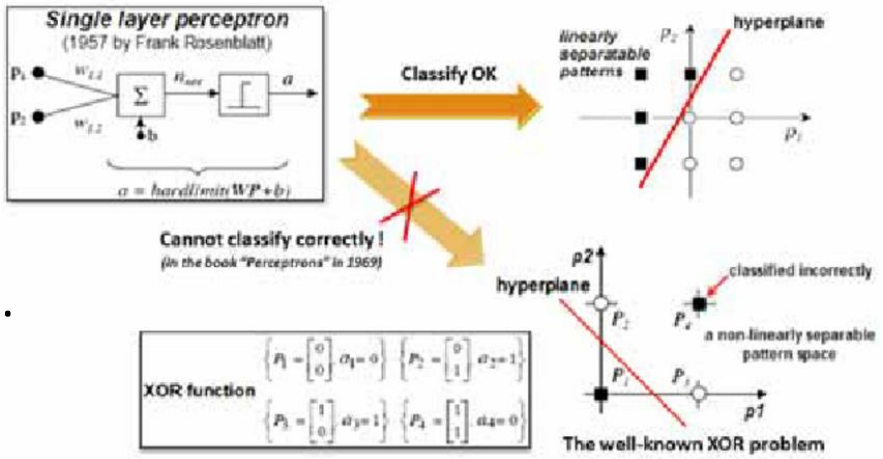
Schematic of Rosenblatt's perceptron.

History of Artificial Intelligent

- **In 1969 Marvin Minsky (The first AI winter!)**
  - published the book "Perceptrons" with a mathematical proof about the limitations of perceptrons as well as unproven claims about the difficulty of training multi-layer perceptrons: ***Perceptron is incapable of learning the simple exclusive-or (XOR) function***.
    - Perceptron is only simple linear classifier!



- The book effects on research funding and, consequently, the AI community.
- He could not provide a learning method for multi-layer perceptron.



History of Artificial Intelligent

Geoff Hinton (David Rumelhart and Ronald Williams) 1986

- With David Rumelhart and Ronald Williams, Hinton published a paper entitled "*Learning representations by back-propagating errors*". In this paper they showed that neural nets with many hidden layers could be effectively trained by a relatively simple procedure with the ability to learn nonlinear functions, 'Back-propagation learning rule'.
  - **Paul Werbos** first described the training artificial neural networks through back-propagation of errors in 1964.



- Since 2013, Geoff Hinton divides his time working for Google (Google Brain) and the University of Toronto. He is referred to by some as the "***Godfather of Deep Learning***".[

History of Artificial Intelligent

- **Yann LeCun: Convolutional Neural Network in 1986**
  - He is a founding father of convolutional nets (Convolutional Neural Network).
  - CNN was developed to recognize handwritten digits for a project of AT&T Bell Labs.
    - https://www.youtube.com/watch?v=FwFduRA_L6Q
  - He joined New York University (NYU) in 2003 and became the first director of Facebook AI Research in 2014.

The second AI winter~

Unfortunately, neural nets were in for another deep freeze. The approach didn't scale to larger problems, and by the 90s the support vector machine (SVM) became the method of choice, and neural nets were moved back into storage.

History of Artificial Intelligent

- **Vladimir Vapnik and Corinna Cortes (soft margin SVM in 1993)**
  - The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Chervonenkis in 1963. In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick. The current standard incarnation (*soft margin SVM*) was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995
  - Professor of Computer Science at Columbia University, New York City since 2003. In 2014, Vapnik joined Facebook AI Research.
  - Cortes currently serves as the Head of Google Research.

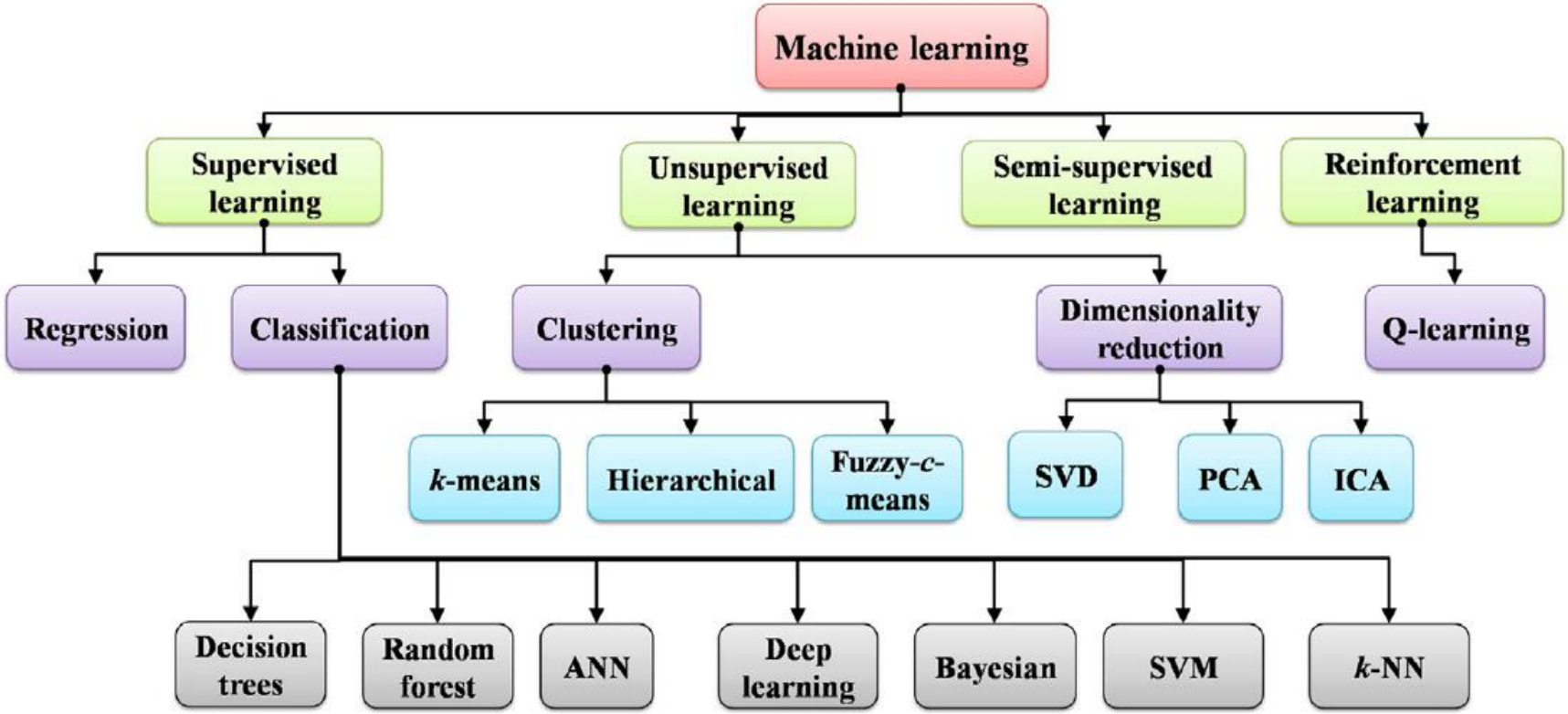
History of Artificial Intelligent

- **Chris Watkins (Q-learning) in 1989**
  - He is a reader in Artficial Intelligence at the Department of Computer Science, University of London
  - Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances.

- **Goodfellow (GAN) in 2014**
  - is best known for inventing generative adversarial networks (GAN)
  - He left Google to join the newly founded OpenAI institute. He returned to Google Research in March 2017.

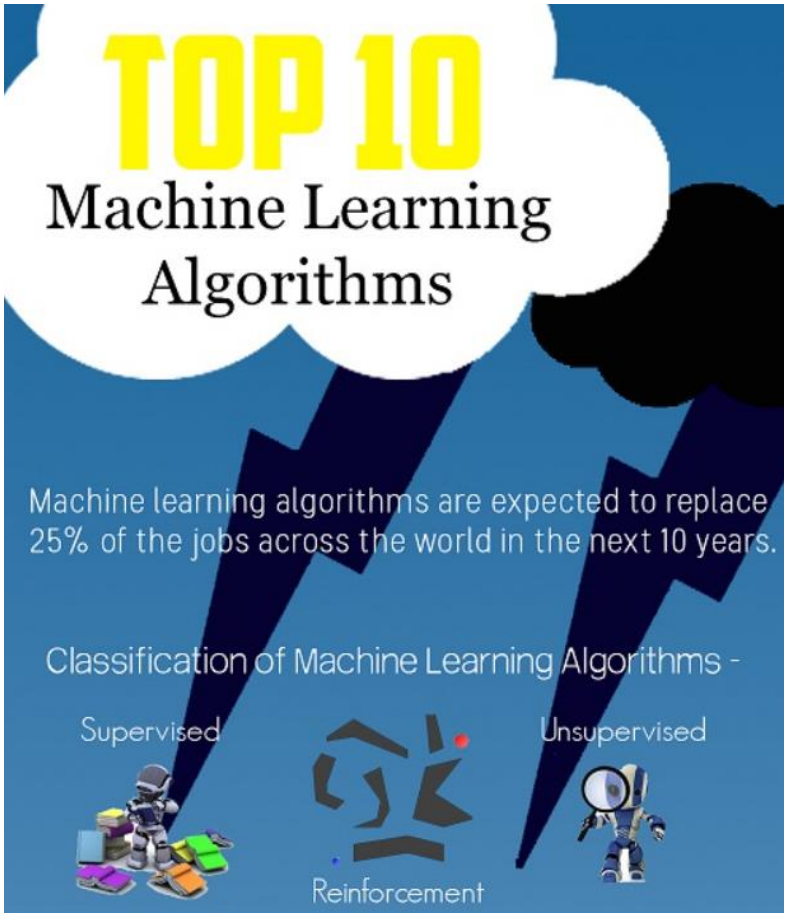
Machine Learning Techniques

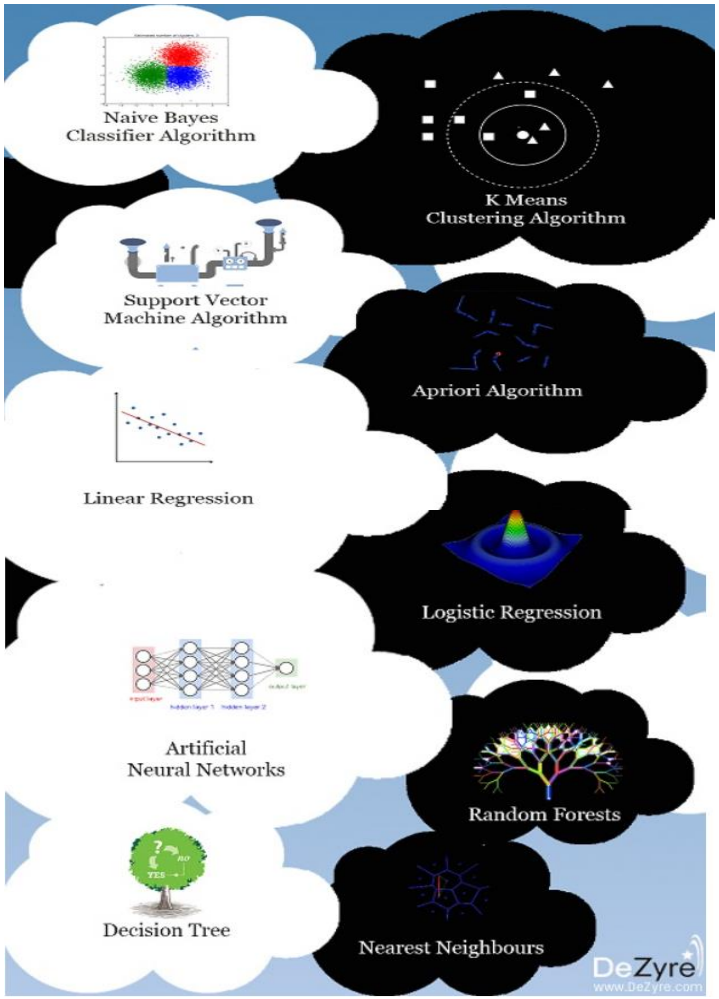
- **Classifications of machine learning techniques**



"Machine learning algorithms for wireless sensor networks: A survey", Information Fusion 2019
https://www.journals.elsevier.com/information-fusion

Top 10 Machine Learning Algorithms





Top 10 Machine Learning Algorithms

- **Naïve Bayes Classifier Algorithm**
  - works on the popular Bayes Theorem of Probability.
  - A Naïve Bayes classifier converges faster, requiring relatively little training data than other discriminative models.
- **K Means Clustering Algorithm**
  - operates on a given data set through pre-defined number of clusters, k.
  - popularly used unsupervised machine learning algorithm for cluster analysis.
- **Support Vector Machine**
  - tries to maximize the distance between the various classes that are involved and this is referred as margin maximization.
  - The best thing about SVM is that it does not make any strong assumptions on data and it does not over-fit the data.

Top 10 Machine Learning Algorithms

- **Apriori Machine Learning**
  - If an item set occurs frequently then all the subsets of the item set, also occur frequently
- **Linear Regression**
  - The algorithm shows the impact on the dependent variable on changing the independent variables.
  - It is easy of use as it requires minimal tuning.
- **Decision Tree**
  - graphical representation that makes use of branching methodology to exemplify all possible outcomes of a decision, based on certain conditions.
  - Decision trees are easy to use but creating large decision trees that contain several branches is a complex and time consuming task.
  - consider only one attribute at a time and might not be best suited for actual data in the decision space.

Top 10 Machine Learning Algorithms

Random Forest

- uses a bagging approach to create a bunch of decision trees with random subset of the data.
- Overfitting is less of an issue with Random Forests, unlike decision tree machine learning algorithms. There is no need of pruning the random forest.
- Large number of decision trees in the random forest can slow down the algorithm in making real-time predictions.

Logistic Regression

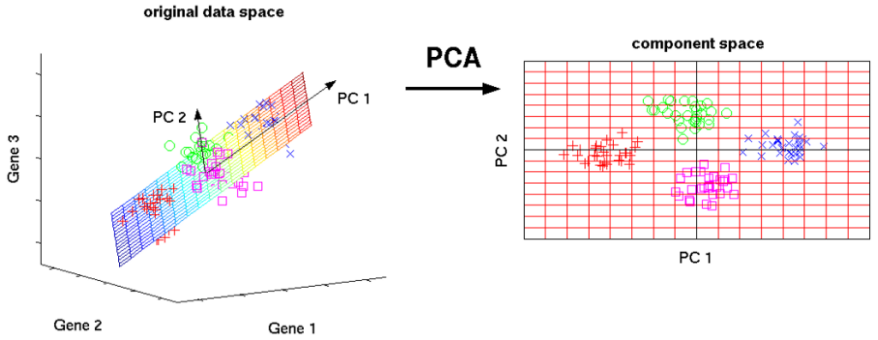
- This algorithm applies a logistic function to a linear combination of features to predict the outcome for classification.
- Logistic regression algorithms require more data to achieve stability and meaningful results.

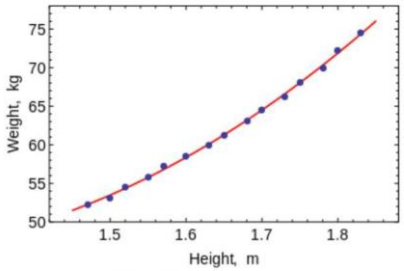
Artificial Neural Networks

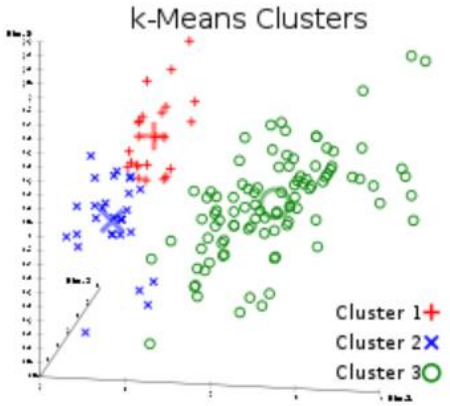
Top 10 Machine Learning Algorithms

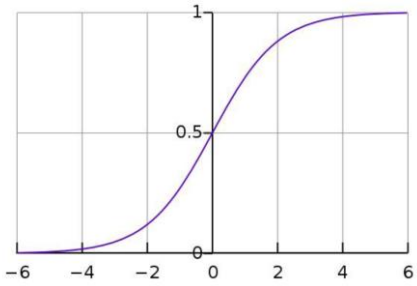
- **Top 10 Machine Learning Algorithms for Data Science**
  : https://towardsdatascience.com

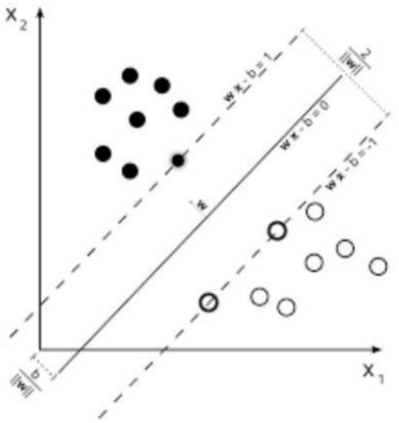
1. **Principal Component Analysis (PCA)**
2. **Least Squares and Polynomial Fitting**
3. **K-Means Clustering**
4. **Logistic Regression**
5. **Support Vector Machines (SVM)**





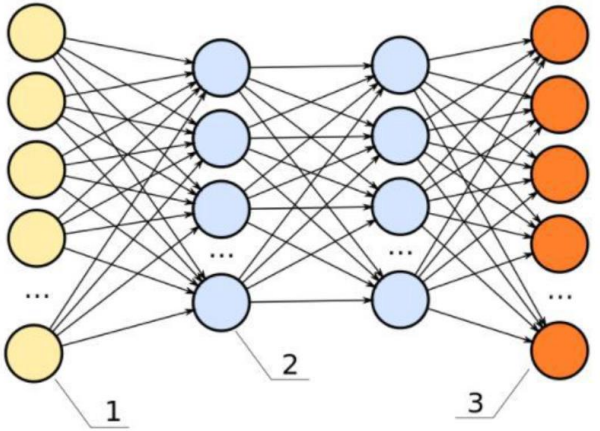


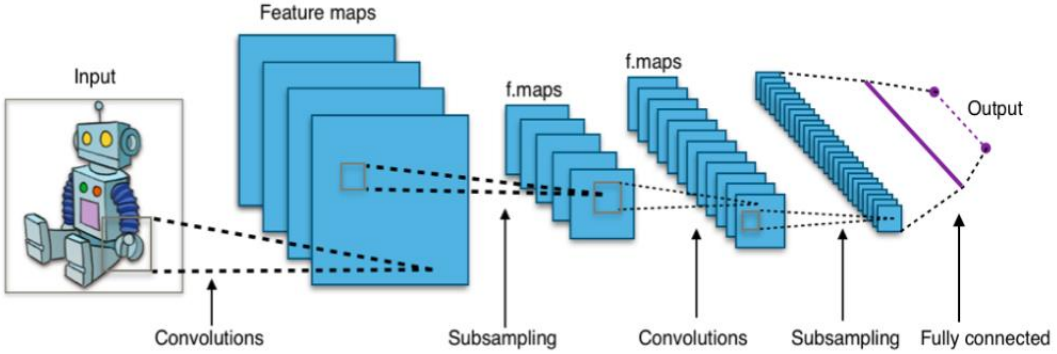


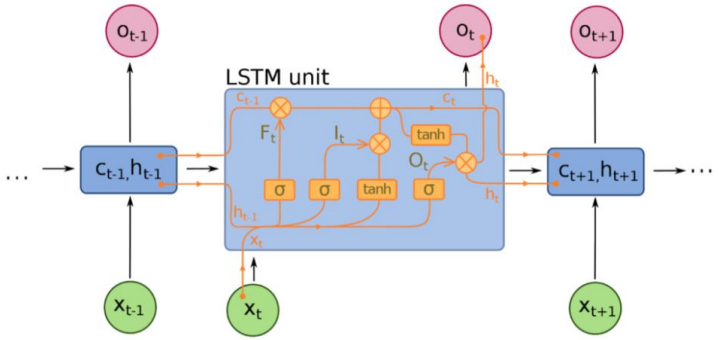


Top 10 Machine Learning Algorithms

6. Feed-Forward Neural Networks
7. Convolutional Neural Networks
8. Recurrent Neural Networks (RNNs)
9. Conditional Random Fields (CRFs)
10. Decision Trees





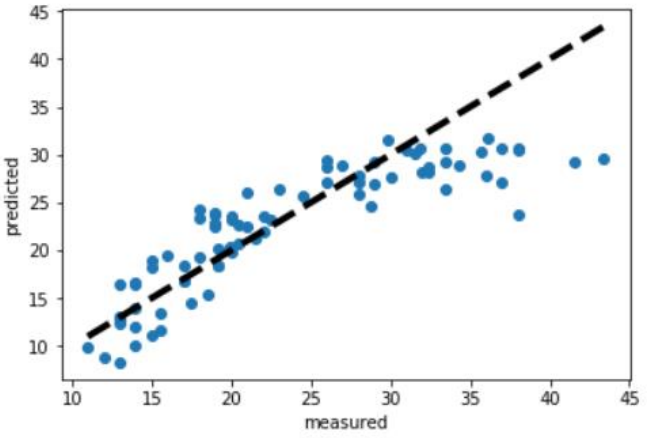


Linear Regression

Linear Regression
Polynomial Regression
Least Square Estimator
Gradient Decent Algorithm

Definition

- **Regression and Classification**
  - Classification is the task of predicting a discrete class label.
  - Regression is the task of predicting a continuous quantity.
- **Linear regression**
  - Linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables).
    - simple linear regression: one explanatory variable
    - multiple linear regression: more than one explanatory variable
  - Practical uses:
    - prediction, or forecasting, or error reduction



Linear Regression

Linear model

$$y_i = \theta_0 + \theta_1 x_{i1} + \cdots + \theta_p x_{ip} + \varepsilon_i \quad (i = 1, \cdots, n)$$

$$\underbrace{\boldsymbol{Y}}_{n \times 1} = \underbrace{\boldsymbol{X}}_{n \times (p+1)} \underbrace{\boldsymbol{\theta}}_{(p+1) \times 1} + \underbrace{\boldsymbol{\varepsilon}}_{(n \times 1)}$$

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}$$

- $(y_i; i = 1, \cdots, n)$ form the vector $\boldsymbol{Y}$ of realizations of the dependent variable (also known as the “target variable" or “response").
- $(x_{ij}; j = 1, \cdots, p)$ form the vector $\boldsymbol{x}_i$ of explanatory variables of the i -th observation $(i = 1, \cdots, n)$.
- $(\theta_j; j = 0, \cdots, p)$ form the unknown parameter vector. θ_0: *bias parameter*
- $(\varepsilon_i; i = 1, \cdots, n)$ form the vector $\boldsymbol{\varepsilon}$ of errors, which we shall assume to be random.

Polynomial Regression

Example

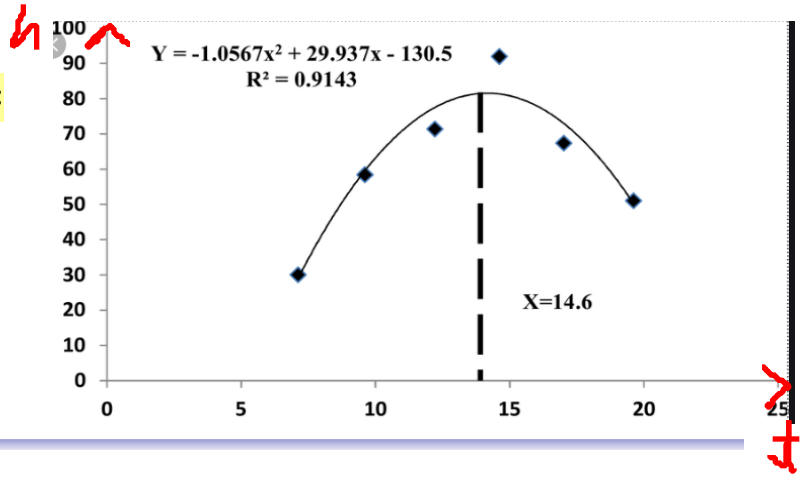
- Consider a situation where a small ball is being tossed up in the air and then we measure its heights of ascent h_i at various moments in time t_i. The relationship can be modeled as

$$h_i = \theta_0 + \theta_1 t_i + \theta_2 t_i^2 + \cdots + \theta_p t_i^p + \varepsilon_i$$

- Linear regression can be used to estimate the values of $\boldsymbol{\theta}$ from the measured data. This model is non-linear in the time variable, but it is linear in the parameters $\boldsymbol{\theta}$

– if we take regressors $\boldsymbol{x}_i = (1, x_{i1}, \cdots, x_{ip}) = (1, t_i, t_i^2, \cdots, t_i^p)$, the model takes on the standard form

$$h_i = \boldsymbol{x}_i \boldsymbol{\theta} + \varepsilon_i$$



Least Square Estimator

- **Consider the following model**

$$\boldsymbol{y} = \boldsymbol{X\theta} + \boldsymbol{\varepsilon}$$

  - We would like the "best possible" estimate of $\boldsymbol{\theta}$. The least squares estimate $\widehat{\boldsymbol{\theta}}$ is defined as the quantity that minimizes the L_2 norm of the error:

$$\|\mathbf{y} - X\widehat{\boldsymbol{\theta}}\| = \min_{\boldsymbol{\theta}} \|\mathbf{y} - X\boldsymbol{\theta}\|$$

    - Thus we minimize the Euclidean distance of the error $\boldsymbol{y} - \boldsymbol{X\theta}$ from the vector zero.
  - To compute the least squares estimate, we calculate the partial derivatives of $\|\boldsymbol{y} - \boldsymbol{X\theta}\|^2$ by $\boldsymbol{\theta}$ (which form a vector) and require them to be zero.

$$(-2)\ X^T(\mathbf{y} - X\widehat{\boldsymbol{\theta}}) = \mathbf{0}$$

Least Square Estimator

- Which is the same as

$$(-2)\ X^T(\mathbf{y} - X\hat{\boldsymbol{\theta}}) = \mathbf{0} \quad \rightarrow \quad X^T X\hat{\boldsymbol{\theta}} = X^T\mathbf{y}$$

  - These are the **normal equations**: we have (p+1) linear equations for (p+1) unknowns
  - If we now assume that the matrix X has full rank (which means rank p+1), then $X^T X$ is invertible

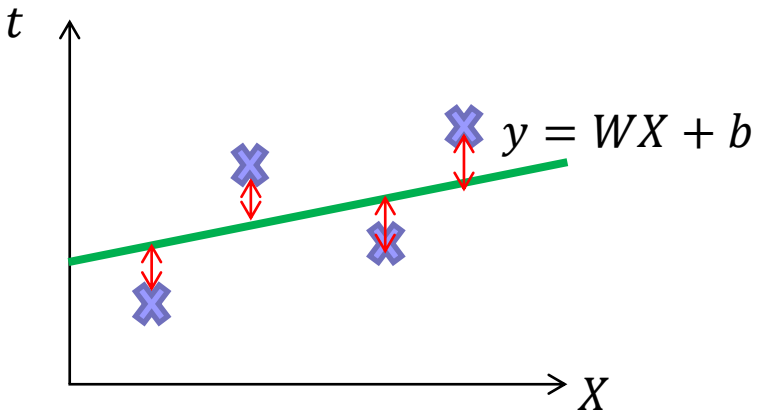
- The lease squares solution is unique and can be written as

$$\hat{\boldsymbol{\theta}} = (X^T X)^{-1} X^T \mathbf{y}$$

Gradient Descent Algorithm

- **Linear Hypothesis**

$$H_W(X) = WX + b$$



- **Cost function : L2 loss function**
  - For m training examples

$$cost = \frac{1}{m}\sum_{i=1}^{m}\left[H\left(X^{(i)}\right) - t^{(i)}\right]^2$$

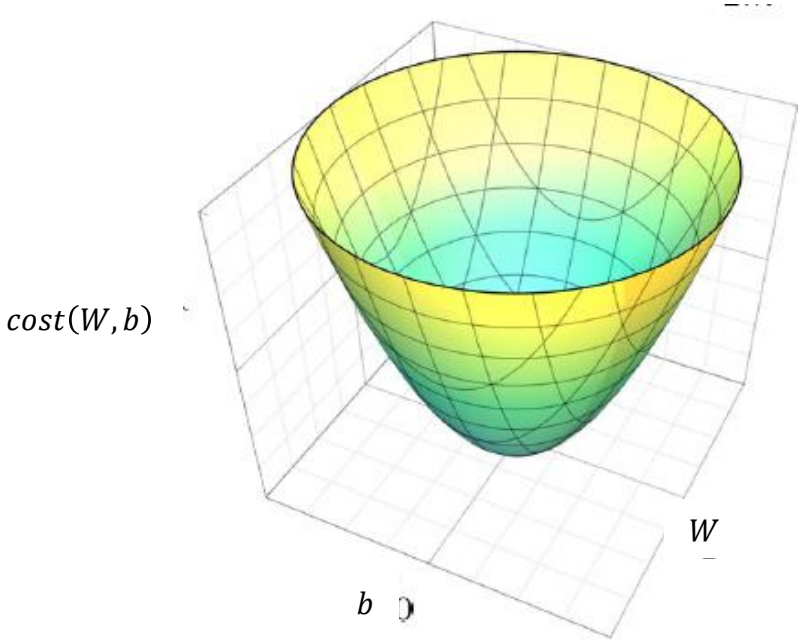
$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left[WX^{(i)} + b - t^{(i)}\right]^2$$

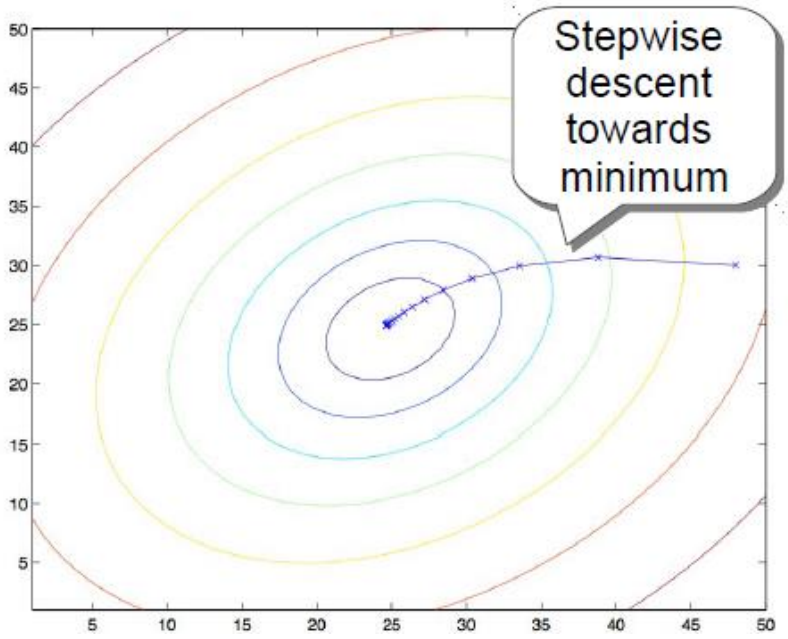
- **Learning objective: Finding optimal parameters (W, b)**

$$(W^*, b^*) = \min_{W,b} cost(w,b)$$

Gradient Descent Algorithm

- **How to minimize cost?**
  - Little modified cost function
    - $cost(W,b) = \frac{1}{2m}\sum_{i=1}^{m}\left[WX^{(i)} + b - t^{(i)}\right]^2$





Gradient Descent Algorithm

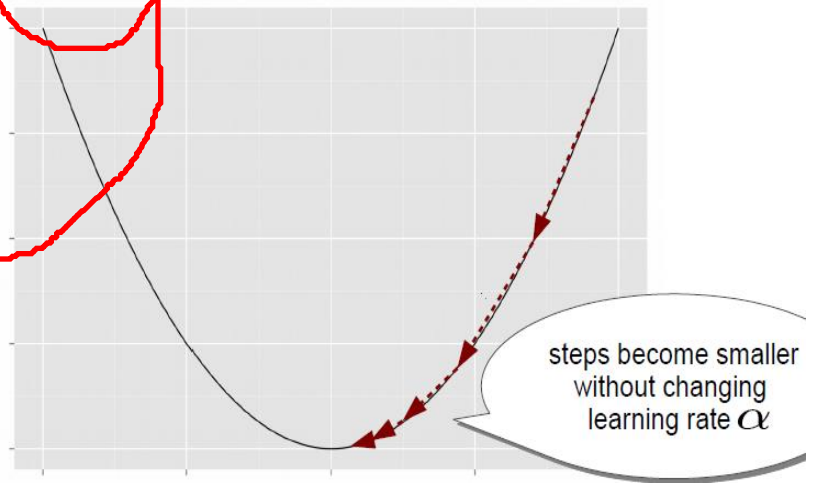
$$WX + b = \overbrace{[b \;\; w_1 \; w_2 \;\; \cdots w_d]}^{W} \overbrace{\begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}^{X} = WX$$

$$W := W - \alpha \frac{\partial}{\partial W} cost(W) = W - \alpha \frac{\partial}{\partial W} \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left[WX^{(i)} - t^{(i)} \right]^2 \right\}$$

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[WX^{(i)} - t^{(i)} \right] X^{(i)}$$

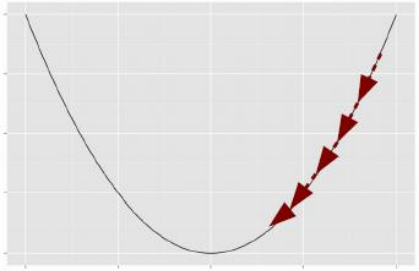
α: learning rate

The most commonly used rates are :
0.001, 0.003, 0.01, 0.03, 0.1, 0.3.

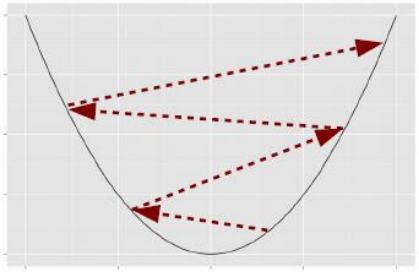


Gradient Descent Algorithm

- **Learning rate considerations**
  - Small learning rate leads to slow convergence

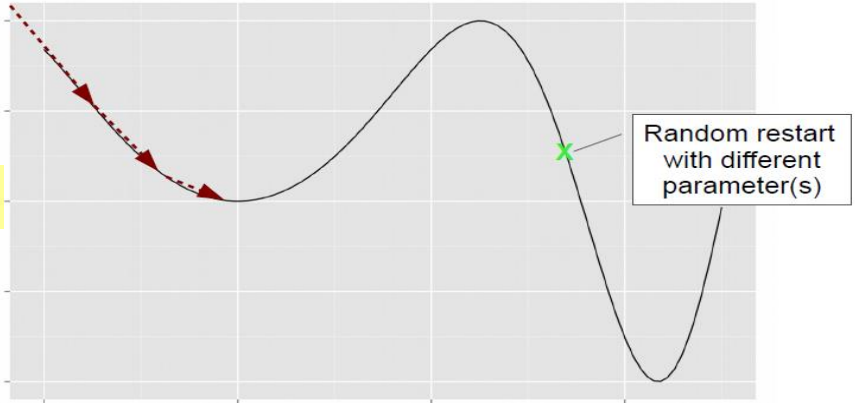


  - Overly large learning rate may not lead to converge or divergence



  - Open $\alpha \in [0.001,1]$

- **Gradient descent can get stuck at local minima.**

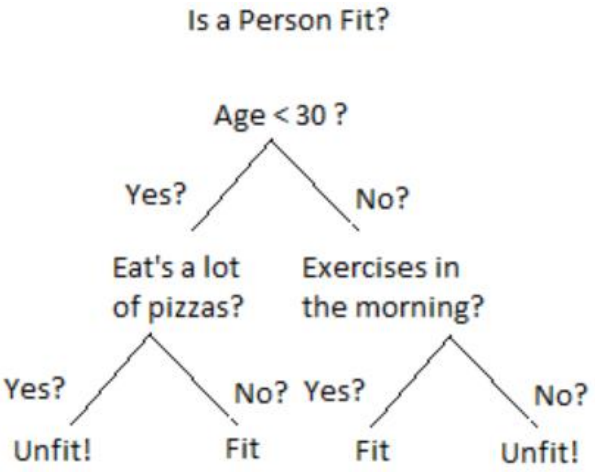


Tree-based Classification and Regression

Decision Tree
Random Forest

Decision Tree

- **A decision tree consists of**
  - Nodes: test for the value of a certain attribute
  - Edges:
    - correspond to the outcome of a test
    - connect to the next node or leaf
  - Leaves: (leaf)
    - terminal nodes that predict the outcome



to classifiy an example:

1. start at the root
2. perform the test
3. follow the edge corresponding to outcome
4. goto 2. unless leaf
5. predict that outcome associated with the leaf

ID3 Algorithm

ID3 Algorithm (Iterative Dichotomiser 3)

이분법

불확실성

- Calculate the entropy of every attribute *a* of the data set *S*.
- Partition ("split") the set *S* into subsets using the attribute for which the resulting entropy after splitting is minimized; or, equivalently, information gain is maximum.
- Make a decision tree node containing that attribute.
- Recurse on subsets using the remaining attributes.

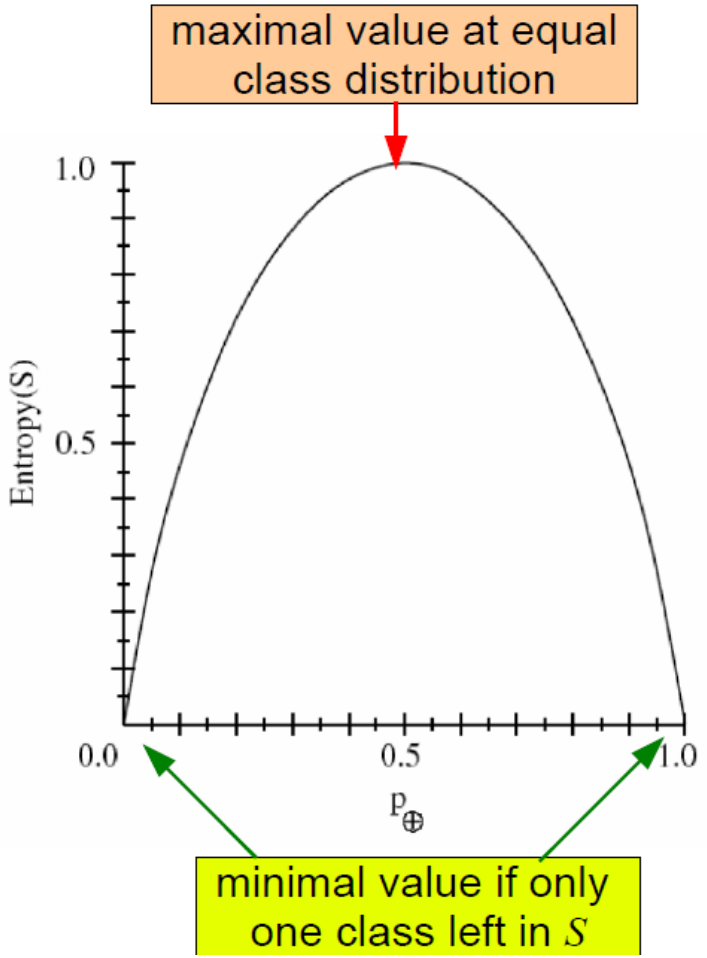
- We want to grow a simple tree
  - a good attribute prefers attributes that split the data so that each successor node is as pure as possible.
  - i.e., the distribution of examples in each node is so that it mostly contains examples of a single class
- Entropy is the amount of information that is contained all examples of the same class → no information

Entropy

- For simple two classes
  - S is a set of examples
  - $p_{\oplus}$ is the proportion of examples in class $\oplus$
  - $p_{\ominus} = 1 - p_{\oplus}$ is the proportion of examples in class $\ominus$

Entropy:

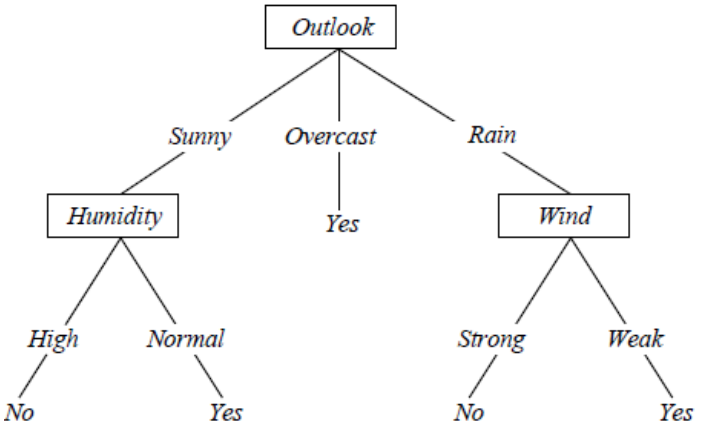
$$E(S) = -p_{\oplus} \cdot \log_2 p_{\oplus} - p_{\ominus} \cdot \log_2 p_{\ominus}$$



Example case

- **Given the data**
  - Each internal node tests an attribute
  - Each branch corresponds to attribute value
  - Each leaf node assigns a classification

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No



- **Predict the value of PlayTennis for**

  <outlook=sunny, Temp=cool, Humidity=high, Wind=strong>

Entropy of Attribute

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- Outlook = sunny: 3 examples yes, 2 examples no

$$E(\text{Outlook}=\text{sunny})=-\frac{2}{5}\log\left(\frac{2}{5}\right)-\frac{3}{5}\log\left(\frac{3}{5}\right)=0.971$$

- Outlook = overcast: 4 examples yes, 0 examples no

$$E(\text{Outlook}=\text{overcast})=-1\log(1)-0\log(0)=0$$

- Outlook = rainy : 2 examples yes, 3 examples no

$$E(\text{Outlook}=\text{rainy})=-\frac{3}{5}\log\left(\frac{3}{5}\right)-\frac{2}{5}\log\left(\frac{2}{5}\right)=0.971$$

General case (multinomial classes)

- Entropy can be easily generalized for $n > 2$ classes
  - p_i is the proportion of examples in S that belong to the i-th class

$$E(S)=-p_1\log p_1-p_2\log p_2\ldots-p_n\log p_n=-\sum_{i=1}^{n}p_i\log p_i$$

Average Entropy of Attribute

- **Problem**
  - Entropy only computes the quality of a single (sub-)set of examples
  - corresponds to a single value
  - How can we compute the quality of the entire split?
- **Solution:**
  - Compute the weighted average over all sets resulting from the split: weighted by their size

$$I(S,A)=\sum_i \frac{|S_i|}{|S|}\cdot E(S_i)$$

  - Average entropy for attribute *Outlook*:

$$I(\text{Outlook})=\frac{5}{14}\cdot 0.971+\frac{4}{14}\cdot 0+\frac{5}{14}\cdot 0.971=0.693$$

Information Gain for Attribute

- When an attribute ***A*** splits the set ***S*** into subsets S_i
  - We compute the average entropy and compare the sum to the entropy of the original set *S*.

Information Gain for Attribute A

alleachE averageE

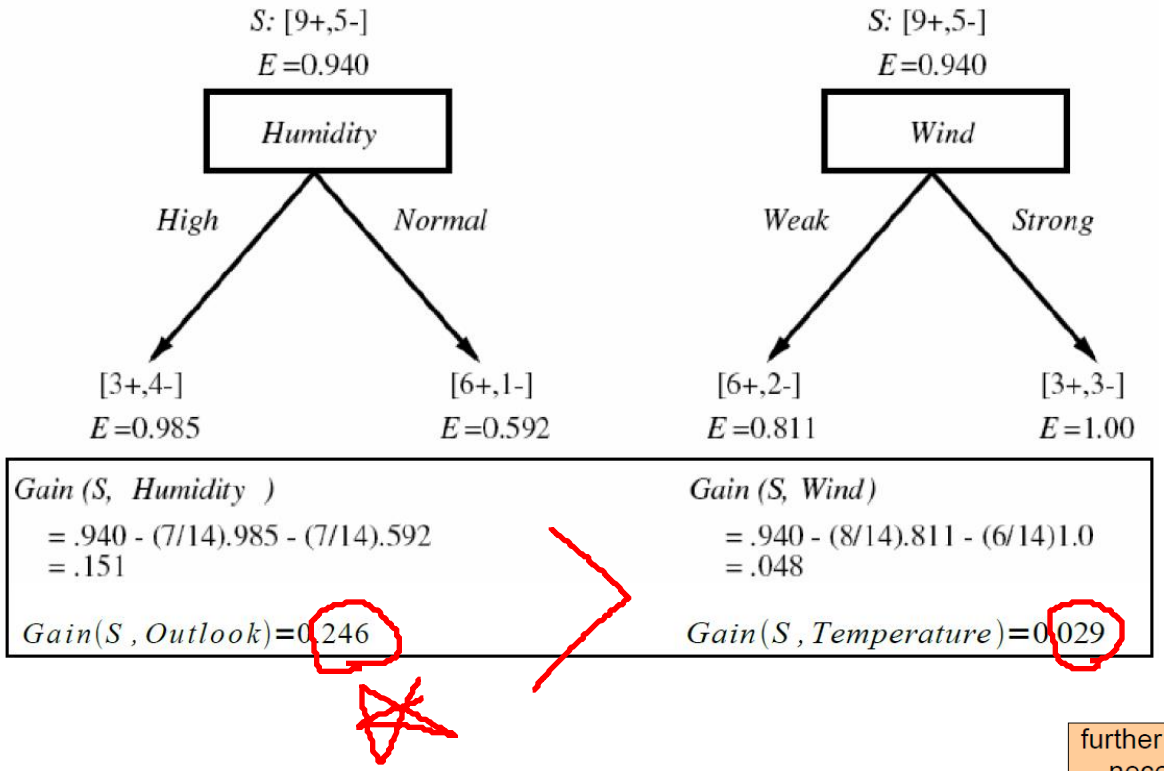
$$Gain(S, A) = E(S) - I(S, A) = E(S) - \sum_i \frac{|S_i|}{|S|} \cdot E(S_i)$$

  - For the PlayTennis example at the first node, $S = all\ days$
    - NO: five days, YES: nine days

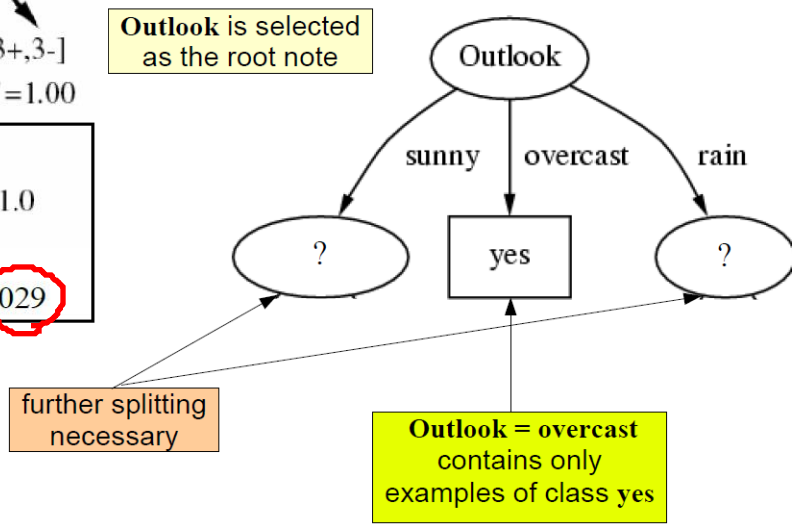
$$E(S) = -\left(\frac{5}{14}\right) log_2 \left(\frac{5}{14}\right) - \left(\frac{9}{14}\right) log_2 \left(\frac{9}{14}\right) = 0.940$$

- The attribute that maximizes the difference is selected.
  - Maximizing information gain is equivalent to minimizing average entropy, because *E*(*S*) is constant for all attributes *A*

Example



Gain Big
-> Select Outlook



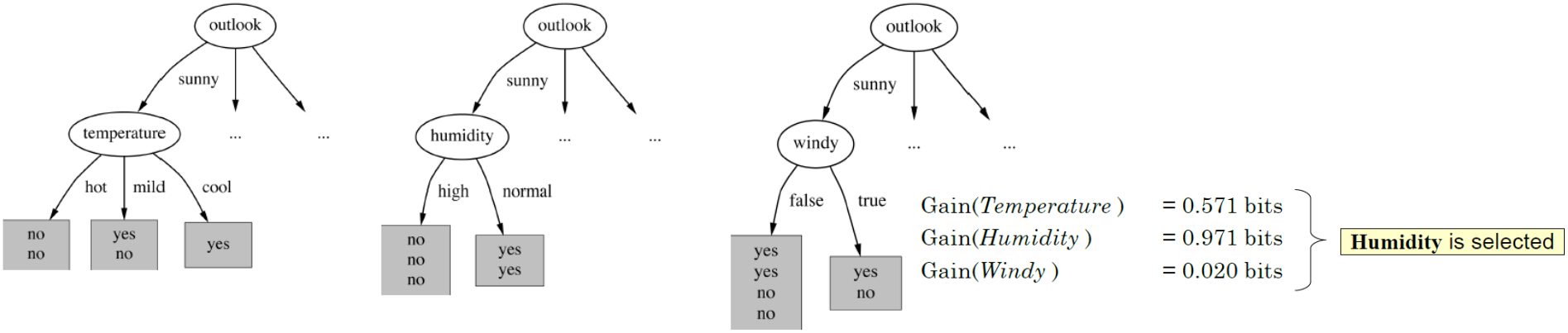
Example

- Outlook→sunny blanch, $S1 = \{D1, D2, D8, D9, D11\}$
- NO: three days, YES: two days

$$E(S1) = -\left(\frac{3}{5}\right) log_2 \left(\frac{3}{5}\right) - \left(\frac{2}{5}\right) log_2 \left(\frac{2}{5}\right) = 0.971$$

$$I(S1, Temp)$$
$$= \left(\frac{2}{5}\right)\left(-\left(\frac{2}{2}\right) log_2 \left(\frac{2}{2}\right) - \left(\frac{0}{2}\right) log_2 \left(\frac{0}{2}\right)\right) + \left(\frac{2}{5}\right)\left(-\left(\frac{1}{2}\right) log_2 \left(\frac{1}{2}\right) - \left(\frac{1}{2}\right) log_2 \left(\frac{1}{2}\right)\right)$$
$$+ \left(\frac{1}{5}\right)\left(-\left(\frac{0}{1}\right) log_2 \left(\frac{0}{1}\right) - \left(\frac{1}{1}\right) log_2 \left(\frac{1}{1}\right)\right) = 0.4$$

$$Gain(S1, Temp) = E(S1) - I(S1, Temp) = 0.971 - 0.4 = 0.571$$



Decision Tree

- **Advantages:**
  - Simple to understand and interpret. Help determine worst, best and expected values for different scenarios.
  - Can be combined with other decision techniques.

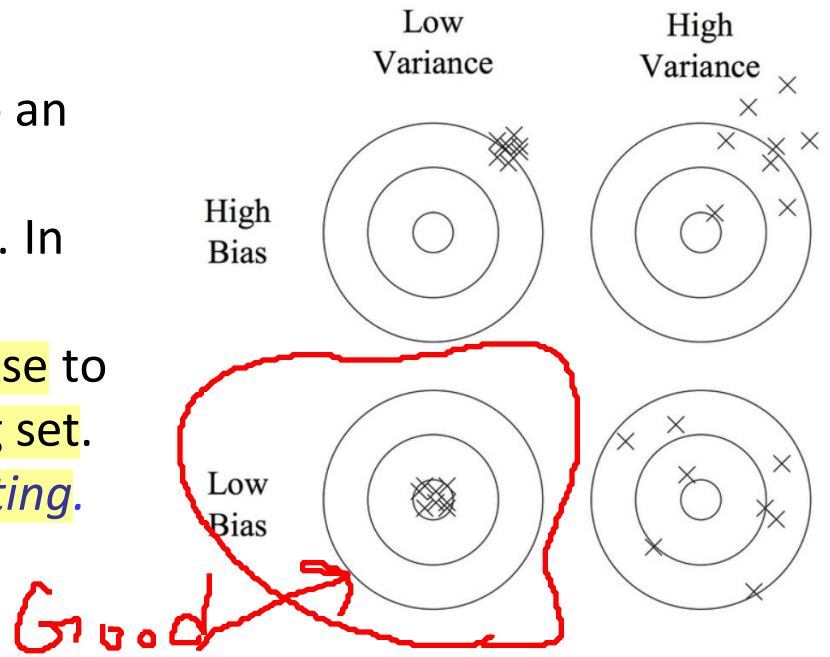
- **Disadvantages:**
  - They are unstable, a small change in the data can lead to a large change in the structure of the optimal decision tree.
  - They are often relatively inaccurate. Many other predictors perform better with similar data. This can be remedied by replacing a single decision tree with a random forest of decision trees.
  - For data including categorical variables with different number of levels, information gain in decision trees is biased in favor of those attributes with more levels.

Decision Tree Pruning

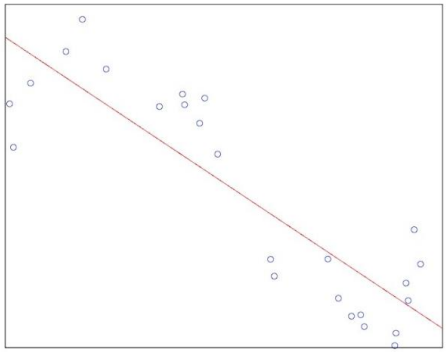
- **A tree that is too large risks overfitting the training data and poorly generalizing to new samples.**
  - Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.
  - Pre-Pruning:
    - stop growing a branch when information becomes unreliable
  - Post-Pruning:
    - grow a decision tree that correctly classifies all training data
    - simplify it later by replacing some nodes with leafs
  - Postpruning preferred in practice—prepruning can "stop early"

Bias-Variance Tradeoff

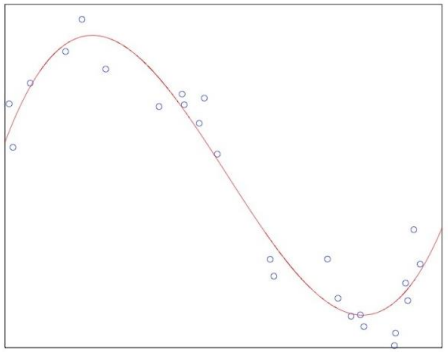
- **The bias variance trade-off is one of the most important aspect of error handling in supervised learning.**
  - Bias refers to the error due to *overly-simplistic assumptions* or *faulty assumptions* in the learning algorithm. *Bias results in under-fitting the data*. A high bias means our learning algorithm is missing important trends amongst the features.
  - Variance refers to the error due to an *overly-complex* that tries to fit the training data as closely as possible. In high variance cases the model's predicted values are extremely close to the actual values from the training set. *High variance gives rise to over-fitting.*



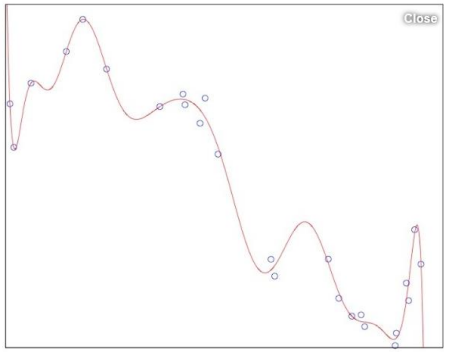
Bias-Variance Tradeoff



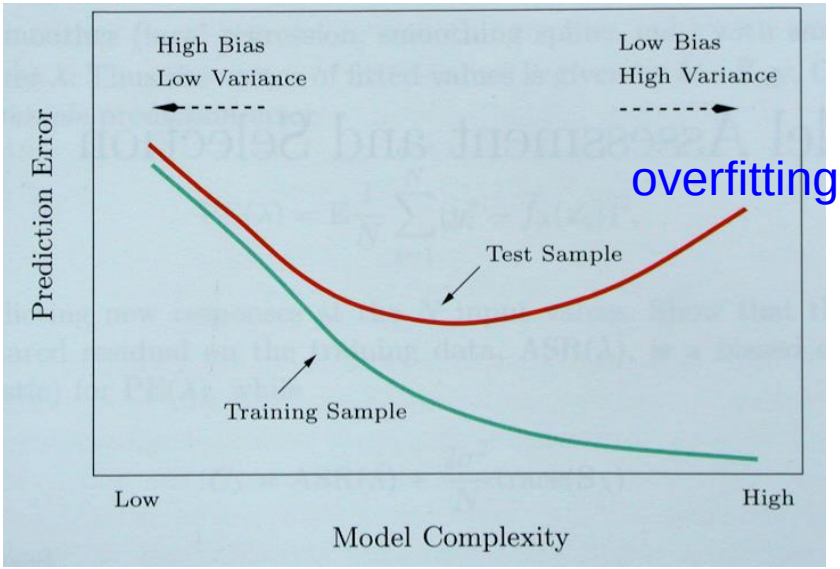
underfit
(degree = 1)



ideal fit
(degree = 3)



overfit
(degree = 20)



High Variance ML Algorithms

- **Decision Tree can be a high variance machine learning model.**
  - Different training samples lead different results.
  - Different samples make different trees.

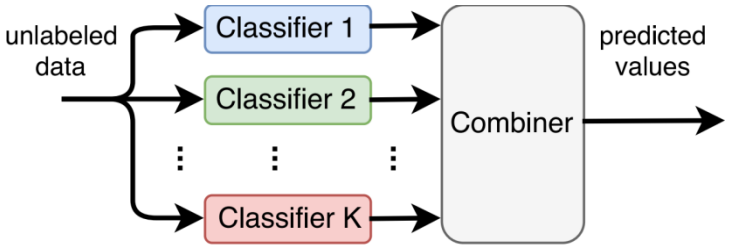
- **Reduce variance without increasing bias**
  - Averaging reduces variance:

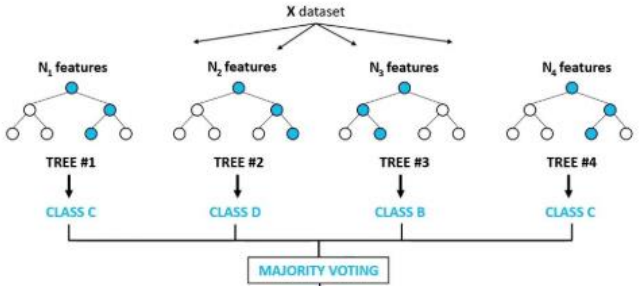
$$VAR(\bar{X}) = \frac{VAR(X)}{N}$$

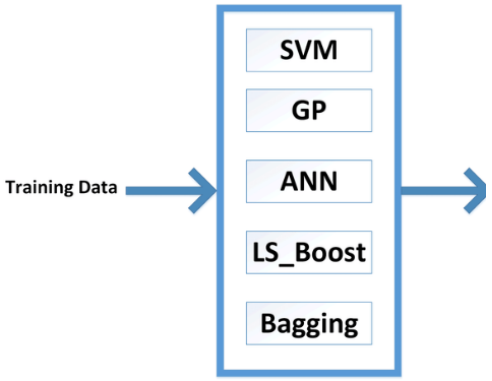
  - Average models to reduce model variance
  - One problem:
    - only one training set
    - where do multiple models come from?

Ensemble Learning

- **Ensemble learning**
  - uses multiple learning models to obtain better prediction or classification performance than could be obtained from a single learning model.
  - Use different 'Combiners'
    - Majority voting, averager,..
- **How to make different classifiers**
  - Use the same learning algorithm, but train on different parts of the training data.
    - Each classifier is only training on a small amount of data.
  - Use the same training data, but use different learning algorithms
    - Often, classifiers are not independent, that is, they make the same mistakes.



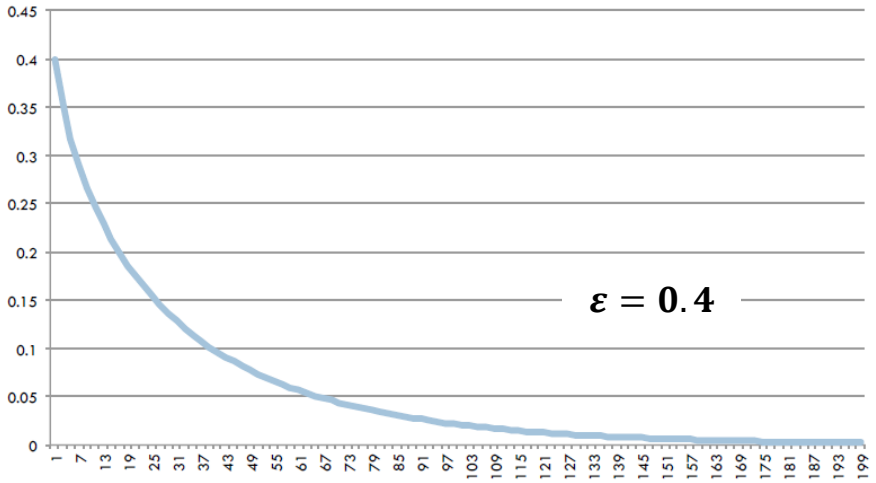




Ensemble Learning

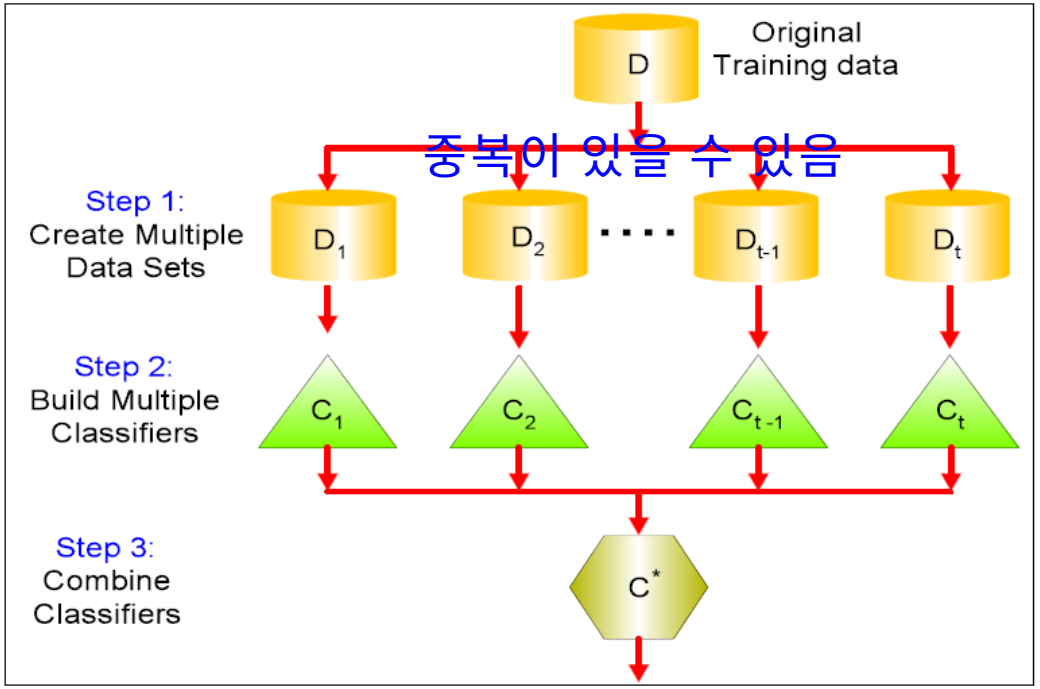
- **Benefits of ensemble learning**
  - Suppose we have ***m*** classifiers and **majority voting** is used.
  - ε is the classification error probability for individual classifier. Assume that classifiers are independent.
  - Then the classification error of the ensemble learning is,

$$P(error) = \sum_{i=(m+1)/2}^{m} \binom{m}{i} \varepsilon^i (1-\varepsilon)^{m-i}$$



Bagging: Bootstrap Aggregation

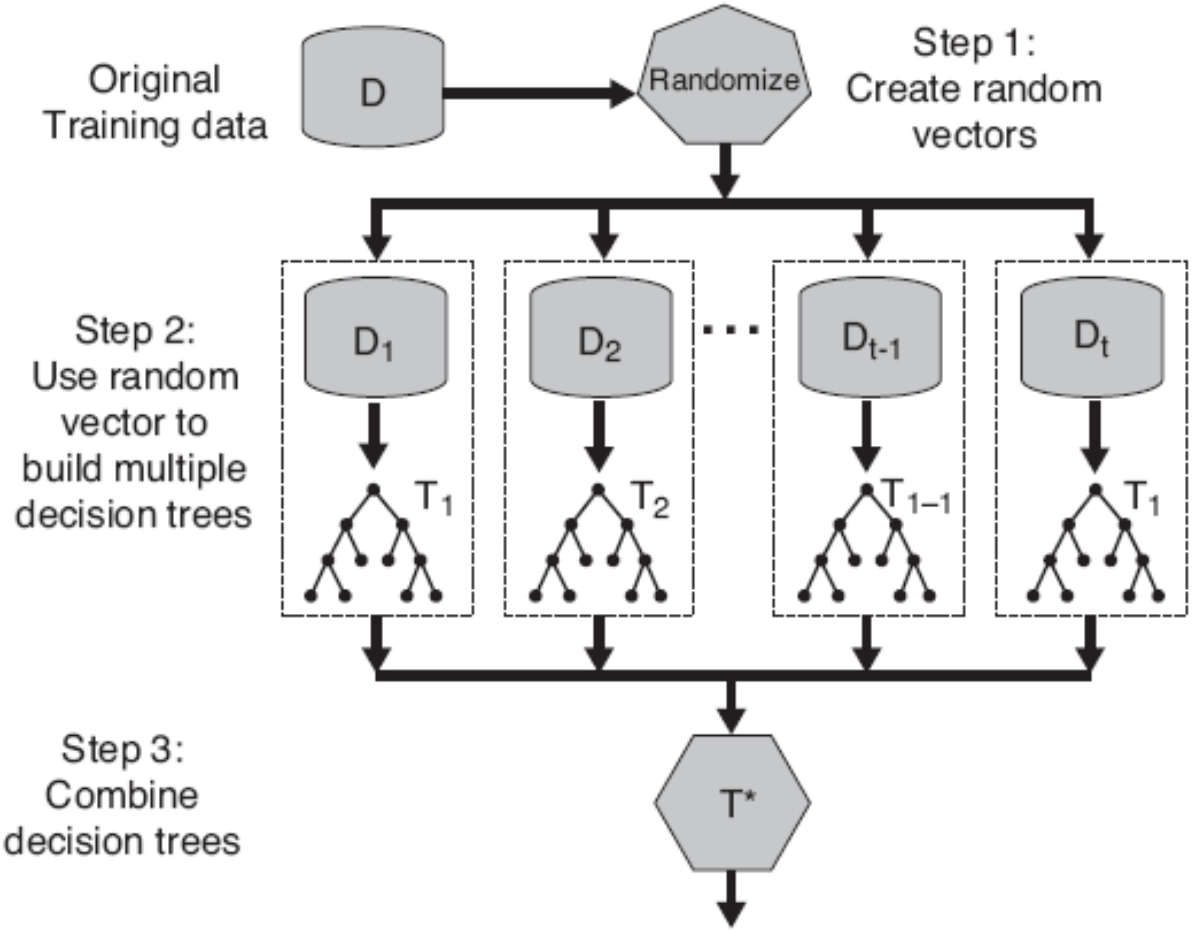
- **Take repeated *bootstrap samples* from training set *D***
- **Bootstrap sampling:**
  - Given set *D* containing *N* training examples, create *D'* by drawing *M* examples at random with replacement from *D*.
- **Bagging:**
  - Create *k* bootstrap samples $D_1, \cdots, D_k$.
  - Train distinct classifier on each D_i.
  - Classify new instance by majority vote / average.



Random Forest

- **Ensemble method specifically designed for decision tree classifiers**
- **Introduce two sources of randomness: "Bagging' and 'Random vector'**
  - Bagging method : each tree is grown using a bootstrap sample of training data.
  - Random vector method : at each node, best split is chosen from a random sample of *m* attributes instead of all attributes.

Random Forest



Random Forest

number of Classifiers

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

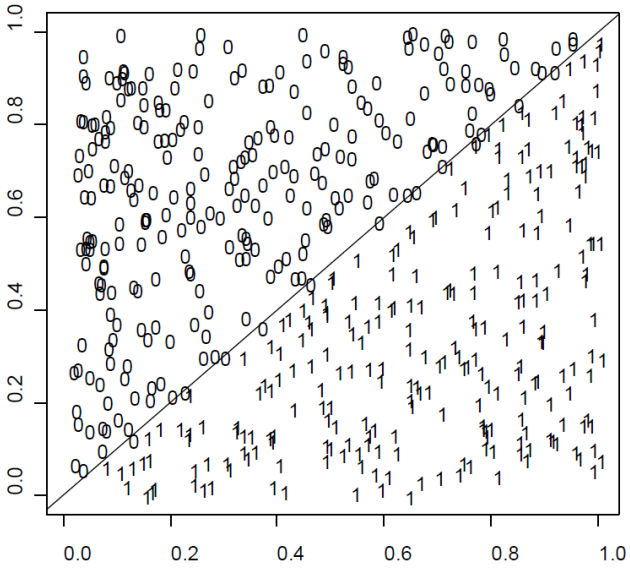
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

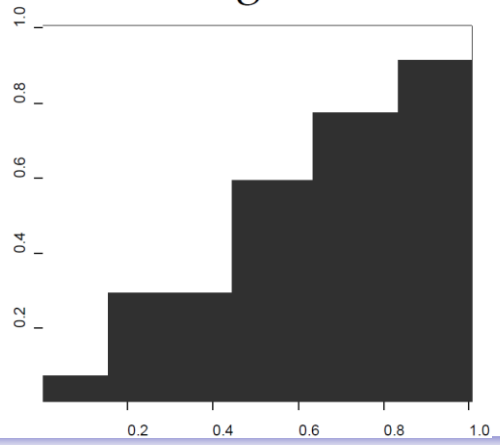
Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

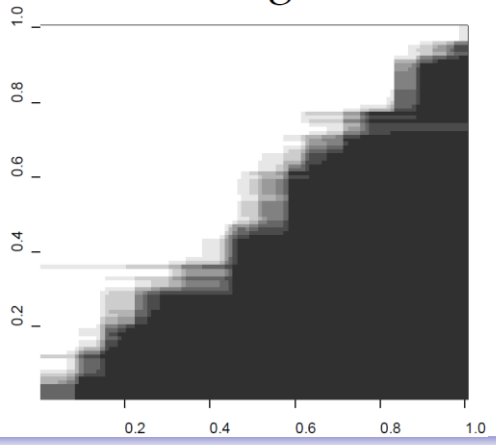
Random Forest



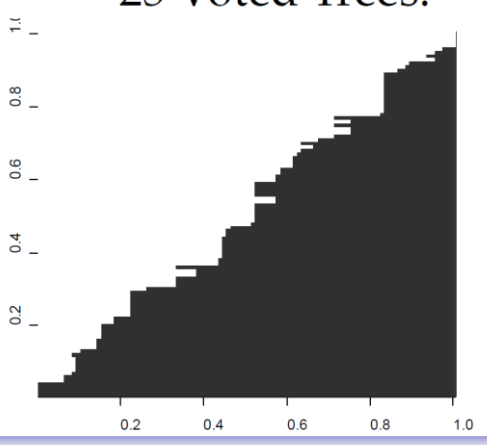
Single tree:



25 Averaged Trees:

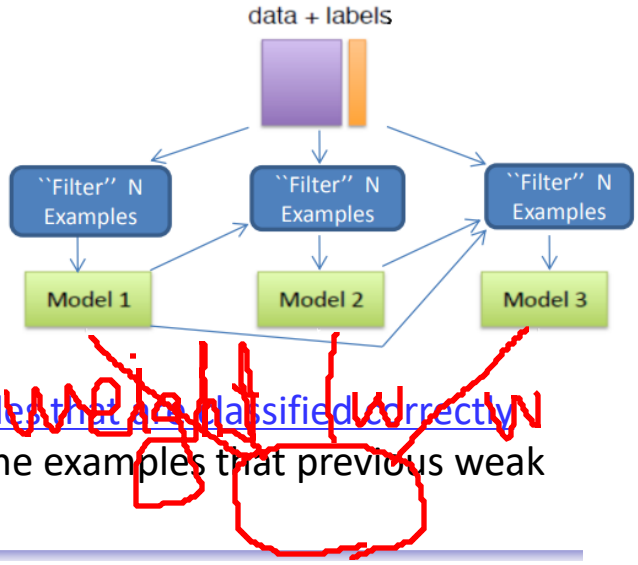


25 Voted Trees:



Boosting

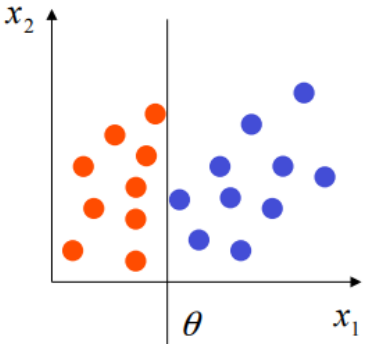
- **Boosting is**
  - an ensemble meta-algorithm for primarily *reducing bias, and also variance* in supervised learning.
  - based on the question "Can a set of weak learners create a single strong learner?"
    - A weak learner is a classifier that is only slightly correlated with the true classification (it can label examples better than random guessing).
    - A strong learner is a classifier that is arbitrarily well-correlated with the true classification.
- **Most boosting algorithms**
  - consist of iteratively learning weak classifiers with respect to a distribution and adding them to a final strong classifier.
  - When they are added, they are weighted in a way that is related to the weak learners' accuracy.
    - Misclassified input data gain a higher weight and examples that are classified correctly lose weight. Thus, future weak learners focus more on the examples that previous weak learners misclassified.



AdaBoost

- **Adaptive Boosting: Construct strong model sequentially by combining multiple weak models**
- **Weak classifier: Decision stump (example)**
  - Simplest type of decision tree
    - Equivalent to linear classifier defined by affine hyper-plane that is orthogonal to axis with which it intersects in threshold θ.
    - Formally, (x_i is the *i*-th training sample (*d*-dimension), *j* is dimension)

$$h(x_i; j, \theta) = \begin{cases} +1 & x_{i(j)} > \theta \\ -1 & else \end{cases}$$



  - Train a decision stump on weighted data

$$(j^*, \theta^*) = argmin_{j,\theta} \left\{ \sum_{i=1}^{N} w_i Error(h(x_i; j, \theta), y_i) \right\}$$

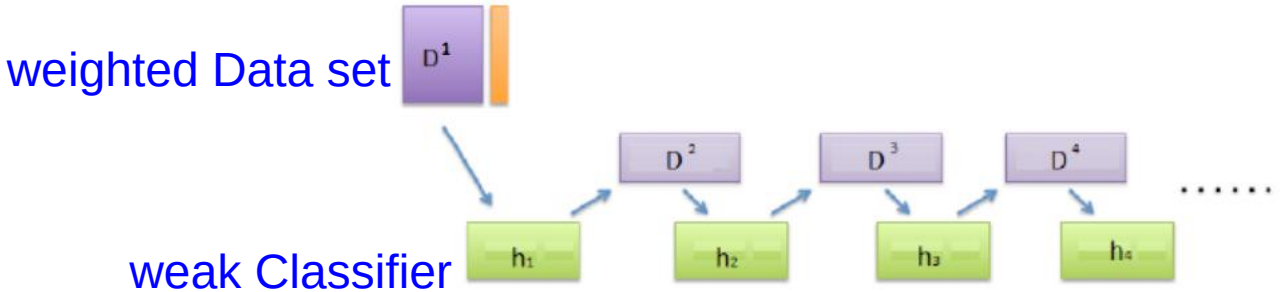
N=number of samples, y_i=label of x_i

AdaBoost

Algorithm Outline

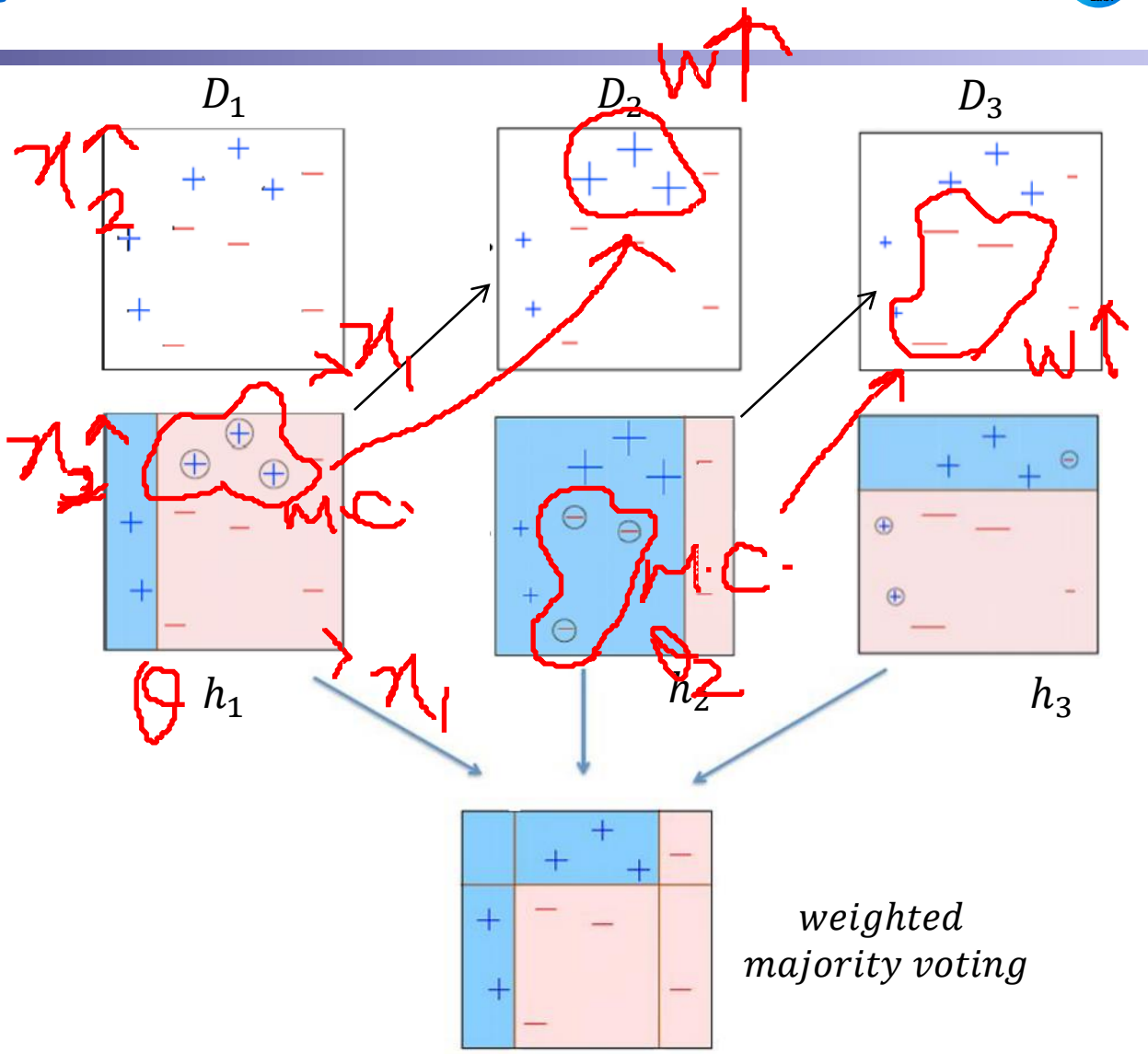
Algorithm 1: AdaBoost Sketch

Input: Training Data $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, Number of rounds M.
Training:
Define a weight distribution over the examples $D_i^1 = \frac{1}{N}$, for $i = 1, 2, \ldots, N$.
for round $j = 1$ to M **do**
  Build a model h_j from the training set using distribution D^j.
  Update D^{j+1} from D^j:
    Increase weights of examples misclasified by h_j.
    Decrease weights of examples correctly classified by h_j.
end for
Prediction: For a new example x', output the weighted (confidence-rated) majority vote of the models $\{h_1, h_2, \ldots, h_M\}$.



AdaBoost

- **Example**



AdaBoost

Algorithm 2 AdaBoost

Input: Training Data $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, Number of rounds M.

Training:

$D_i^1 = \frac{1}{N}$, for $i = 1, 2, \ldots, N$.

for $j = 1$ to M **do**

Define $\epsilon_j = \sum_{i:h_j(x_i) \neq y_i} D_i^j$.

ϵ_j: weighted error of the j-th model

Obtain a hypothesis h_j that minimizes ϵ_j and satisfies the condition $\epsilon_j < \frac{1}{2}$.

$\alpha_j = \frac{1}{2} \log\left(\frac{1-\epsilon_j}{\epsilon_j}\right)$.

α_j: ``confidence'' of the j-th model

$D_i^{j+1} = e^{-y_i h_j(x_i) \alpha_j} D_i^j$.

$D_i^{j+1} = \frac{D_i^{j+1}}{\sum_{i=1}^N D_i^{j+1}}$.

Update weights for next iteration

After normalization we have: $\sum_{i=1}^{N} D_i^{j+1} = 1$

end for

Prediction: $H(\boldsymbol{x}') = sign\left[\sum_{j=1}^{M} \alpha_j h_j(\boldsymbol{x}')\right]$.

$y_i = \{-1, +1\}$

Bagging vs AdaBoost

- **Both train many models on different versions of initial dataset and the aggregate, but …**

Bagging	AdaBoost
Resample dataset	Resample or reweight dataset
Builds base models in parallel	Build base models sequentially
Reduces variance (doesn't work well with e.g. decision stumps)	Reduces variance and bias (work well with decision stumps)

Unsupervised Clustering

Clustering
K-means Algorithm
Density-based Clustering (DBSCAN)
Gaussian Mixture Model (GMM)
Expectation Maximization
Hierarchical Clustering

Clustering

- **Clustering ?**
  - Clustering algorithms divide a data set into natural groups (clusters).
  - Instances in the same cluster are similar to each other, they share certain properties.

- **k-means clustering**
  - aims to partition *n* observations into *k* clusters in which *each observation belongs to the cluster with the nearest mean*. This results in a partitioning of the data space into *Voronoi cells*.
  - unsupervised learning

K-means Clustering

Objective function

- Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$), where each observation is a *d*-dimensional real vector, *k*-means clustering aims to partition the *n* observations into *k* (≤ *n*) sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS) (i.e. variance).

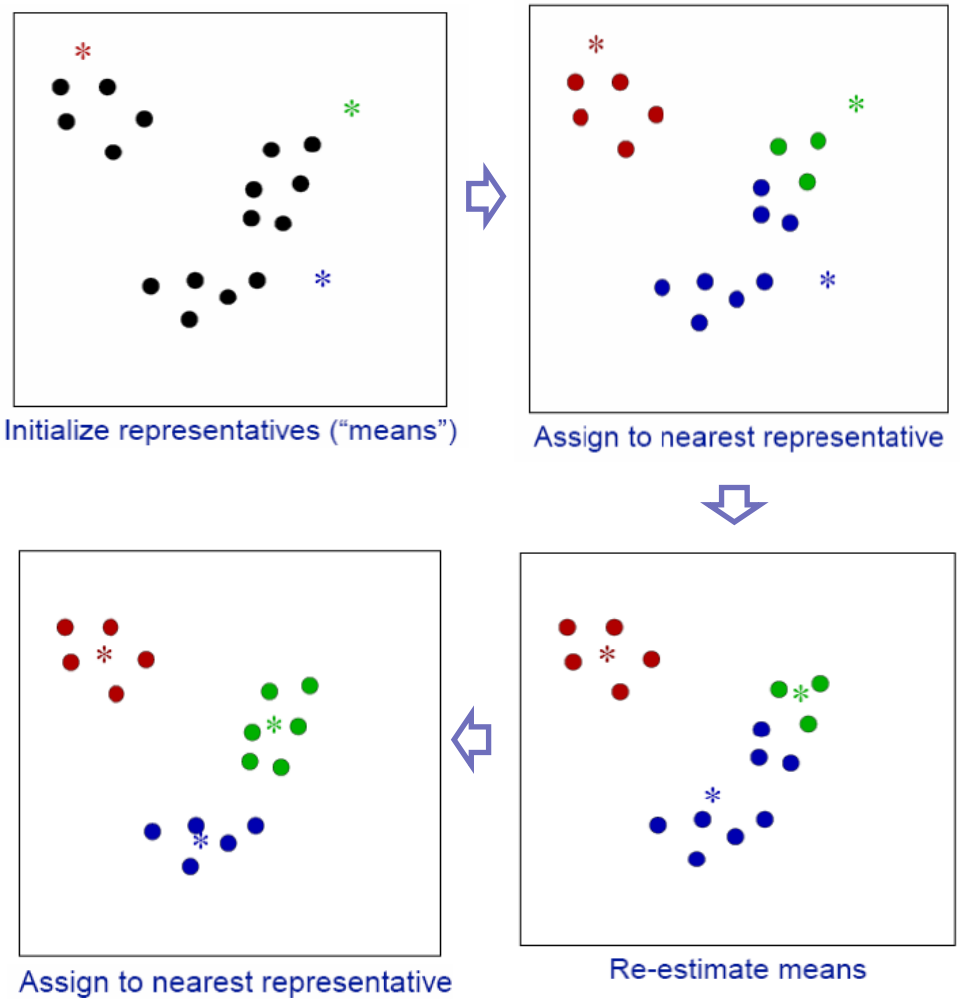
$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

  - where $\boldsymbol{\mu}_i$ is the mean of points in S_i

Iterative Algorithm

- Step 0 :
  - Select K objects as initial centroids.
- Step 1 : (Assignment)
  - For each object, compute distances to k centroids.
  - Assign each object to the cluster to which it is the closest.
- Step 2 : (New Centroids)
  - Compute a new centroid for each cluster.
- Step 3: (Converge)
  - Stop if the change in the centroids is less than the selected covergence criterion.
  - Otherwise repeat Step 1.

Iterative Algorithm

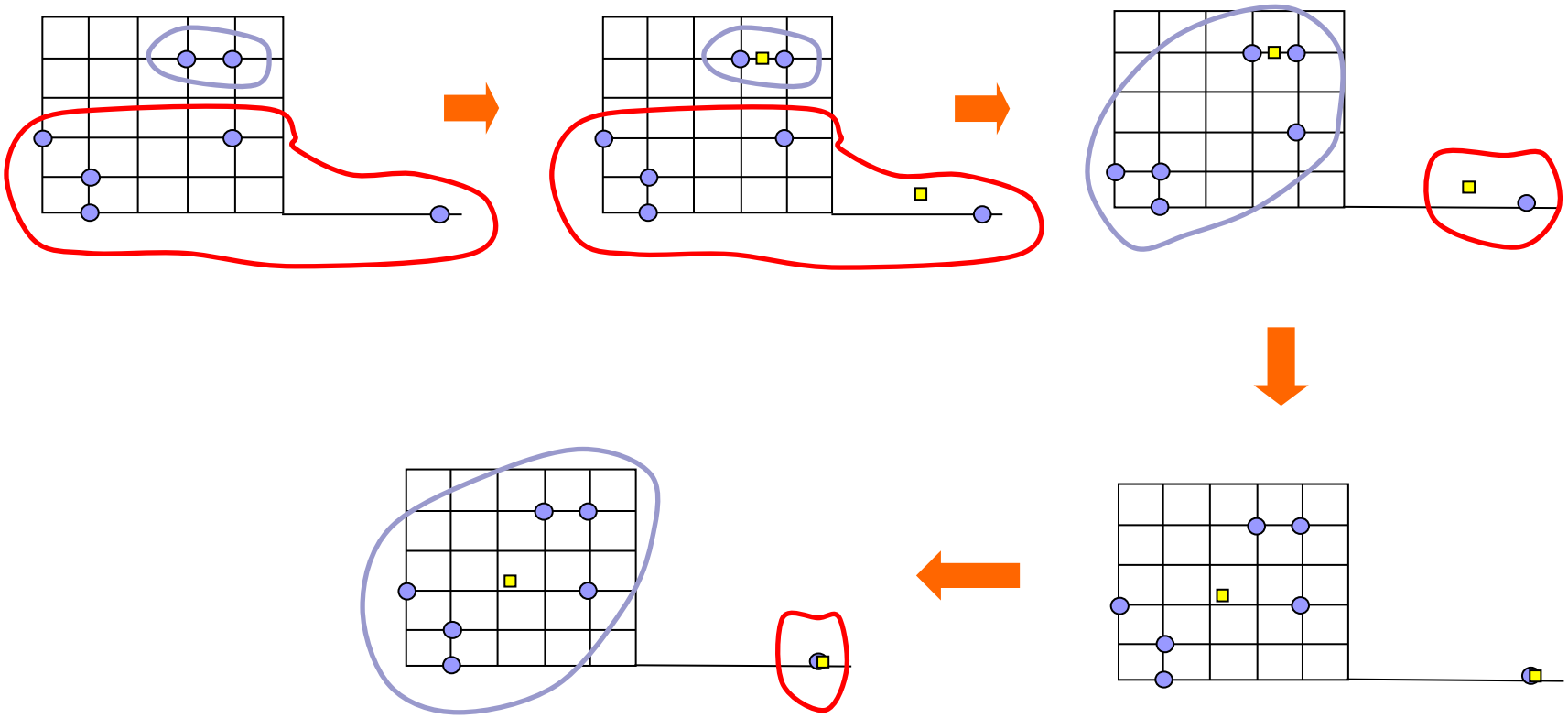


- Easy to use
- Need to know K
- May need to scale data
- Good for initial method

- No guarantee of optimal solution (local optimum)

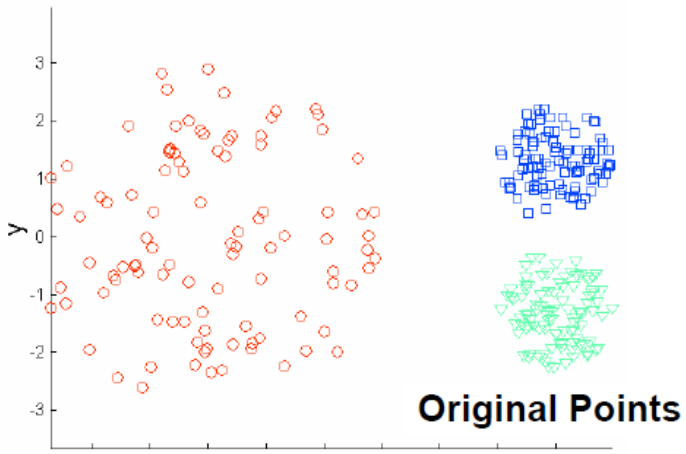
Problems of K-means Clustering

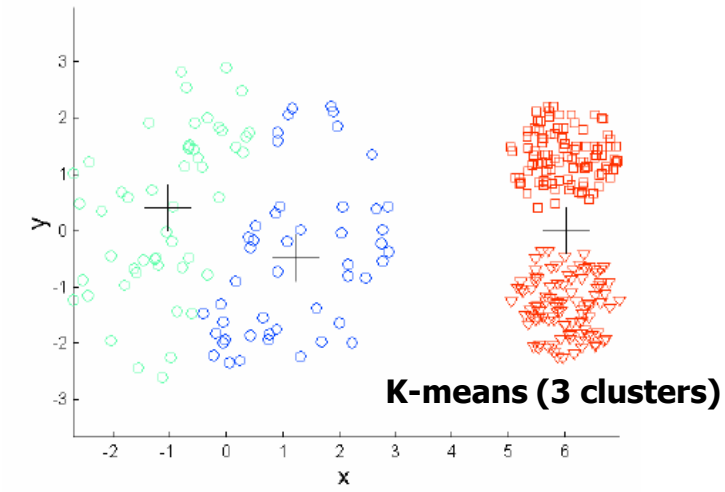
- weakness on outlier (noise)



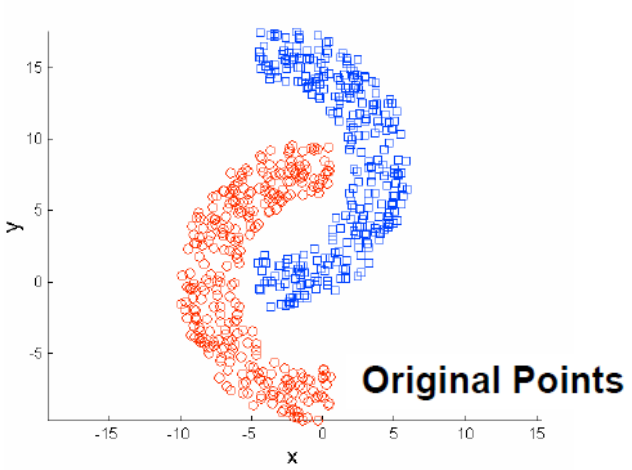
Problems of K-means Clustering

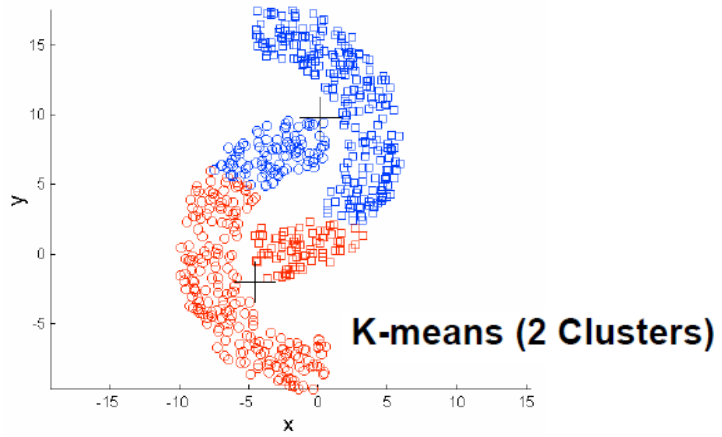
Different density





Non-globular Shapes





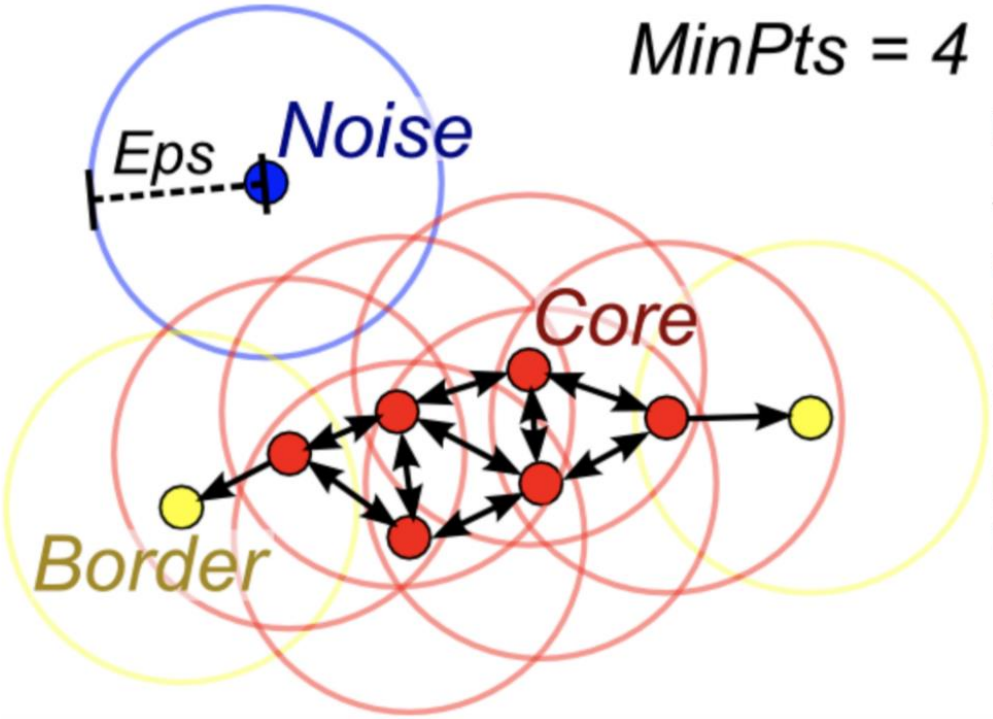
Density-based Clustering Algorithm

- **Density-based spatial clustering of applications with noise (DBSCAN)**
  - proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996.
  - It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

DBSCAN

- Let ε be a parameter specifying the radius of a neighborhood with respect to some point. For the purpose of DBSCAN clustering, the points are classified as *core points*, (*density*-)*reachable points* and *outliers*, as follows:
- A point ***p is a core point*** if at least minPts points are within distance ε of it (including p).
- A point ***q is directly reachable from p*** if point q is within distance ε from core point p.
- A point ***q is reachable from p*** if there is a path $p_1, \ldots, p_n$ with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i. Note that this implies that all points on the path must be core points, with the possible exception of q.
- All points ***not reachable*** from any other point are *outliers* or *noise points*.
- Now if p is a core point, then it forms a *cluster* together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge", since they cannot be used to reach more points.

DBSCAN



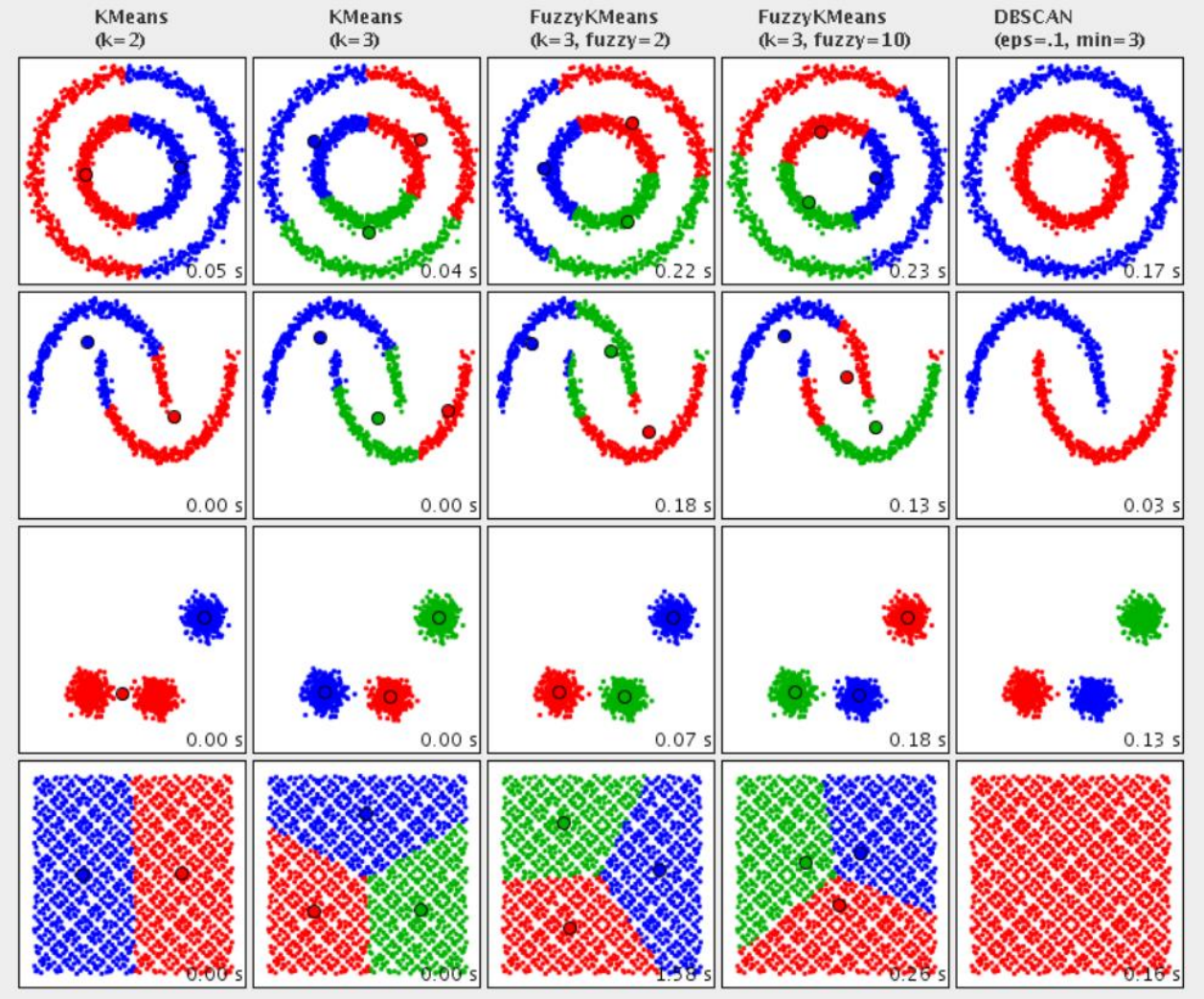
Red: Core Points

Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria

Blue: Noise point. Not assigned to a cluster

- **Requires two parameters**
  - ε : radius of a neighborhood
  - *minPts*: minPts points are within distance ε *to be a core*

DBSCAN


KMeans
(k=2)
KMeans
(k=3)
FuzzyKMeans
(k=3, fuzzy=2)
FuzzyKMeans
(k=3, fuzzy=10)
DBSCAN
(eps=.1, min=3)
0.05 s
0.04 s
0.22 s
0.23 s
0.17 s
0.00 s
0.00 s
0.18 s
0.13 s
0.03 s
0.00 s
0.00 s
0.07 s
0.18 s
0.13 s
0.00 s
0.00 s
1.58 s
0.26 s
0.16 s

DBSCAN

- **Advantages**
  - Does not require to specify the number of clusters (k)
  - Can find arbitrarily shaped clusters: even find a cluster completely surrounded by a different cluster.
  - Has a notion of noise, and is robust to outliers.

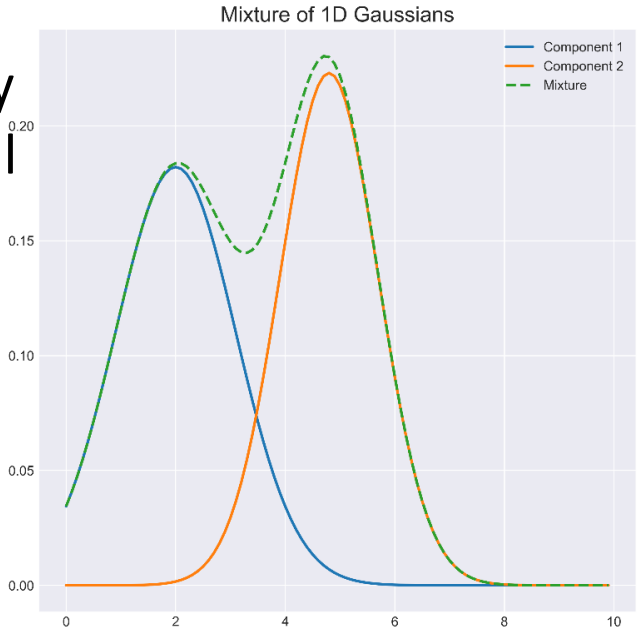
- **Disadvantages**
  - DBSCAN is not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster depending on the order the data is processed.
  - Need to determine meaningful parameters ε and *minPts*.

Mixture Model

- **Types of clustering methods**
  - Hard clustering: clusters do not overlap
    - element either belongs to the cluster or it does not.
    - e.g., K-means, DBSCAN
  - Soft clustering: clusters may overlap
    - Each data point can belong to more than one cluster.
    - use mixture models
    - parameters (e.g., mean/covariance) are unknown

Mixture Model

- **Mixture models in 1-dimension**
  - Mixture models allow rich probability distributions to be represented as a combination of simpler “component” distributions.
  - A mixture is capable of approximating any distribution with an accuracy proportional to the number of components.



- **Gaussian distribution**

$$\mathcal{N}(\mathbf{x}\,|\,\boldsymbol{\mu},\boldsymbol{\sigma}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

mean variance

$$\mathcal{N}(\mathbf{x}\,|\,\boldsymbol{\mu},\Sigma) = \frac{1}{(2\pi|\Sigma|)^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

mean covariance

Gaussian Mixture Model

GMM (Gaussian Mixture Model)

- Gaussian mixture model with *K* components has the form:

$$p(x) = \sum_{k=1}^{K} p(x|z = k)p(z = k)$$

  - where *z* is a categorical latent variable indicating the component identity; x is the sample with d-dimensional features.
  - represent $p(x)$ as superposition of *K* Gaussian distributions.

- The likelihood term for the *k*th component (cluster)

$$p(x|z = k) \sim N(x|\mu_k, \Sigma_k)$$

  - μ_k: mean vector with d-dimension, Σ_k: covariance matrix

- Mixture of Gaussians

$$p(x) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)$$

$$\pi_k \coloneqq p(z = k)$$

Gaussian Mixture Model

- For a given value of 'x_i', we can evaluate the corresponding posterior probabilities, called responsibilities.

$$z_k^i = p(z = k|x_i) = \frac{p(z = k)p(x_i|z = k)}{p(x_i)} = \frac{\pi_k N(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_i|\mu_j, \Sigma_j)}$$

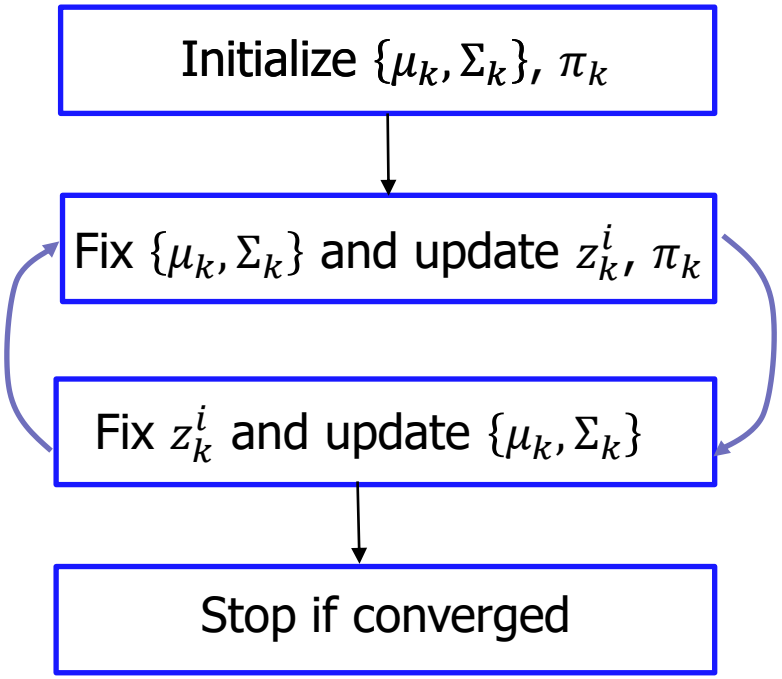
$\pi_k = \frac{N_k}{N}$, N =total number of samples,

N_k=number of cluster k samples

- For a given 'x_i', we can compute the latent variablez_k^i, which is the probability x_i belongs to cluster *k*.
  - Therefore, unlike hard clustering we can say probability to be an element of each possible cluster.
- Problem is 'we don't know $\{\mu_k, \Sigma_k, \pi_k\}$.

GMM Parameter Estimation

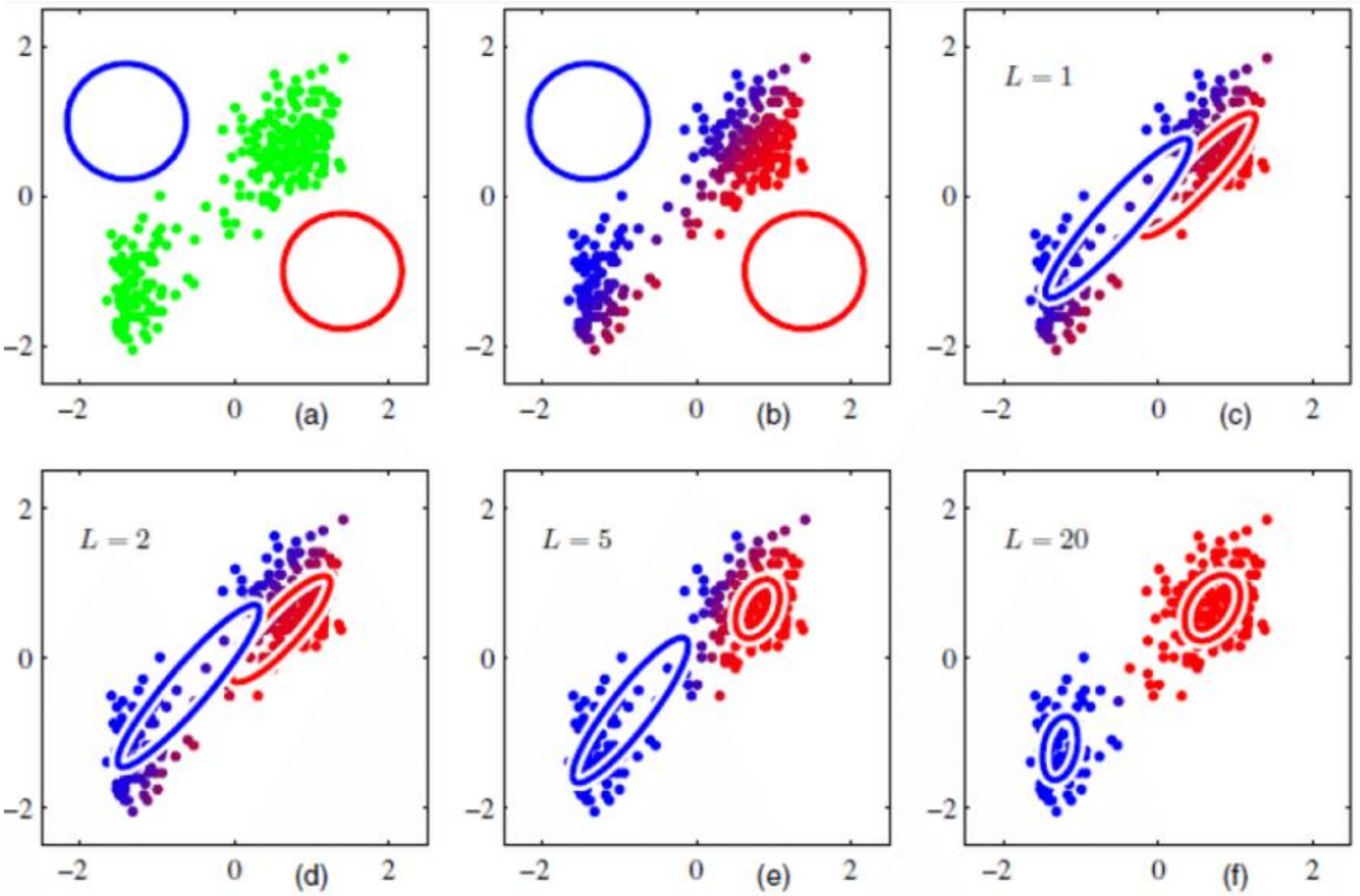
- **EM (Expectation Maximization) for GMM parameter estimation**



$$z_k^i = \frac{\pi_k N(x_i|\mu_k,\Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_i|\mu_j,\Sigma_j)}, \quad \pi_k = \frac{1}{N}\sum_{i=1}^{N} z_k^i$$

$$\mu_k = \frac{\sum_{i=1}^{N} z_k^i \cdot x_i}{\sum_{i=1}^{N} z_k^i}, \quad \Sigma_k = \frac{\sum_{i=1}^{N} z_k^i \cdot (x_i-\mu_k)(x_i-\mu_k)^T}{\sum_{i=1}^{N} z_k^i}$$

GMM Example

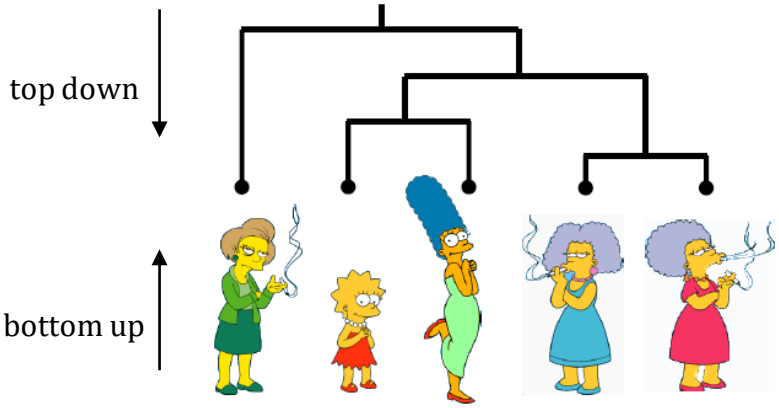


Hierarchical Clustering

- Two types of clustering
  - **Partitional algorithms**: Construct various partitions and then evaluate them by some criterion
  - **Hierarchical algorithms**: Create a hierarchical decomposition of the set of objects using some criterion (focus of this class)

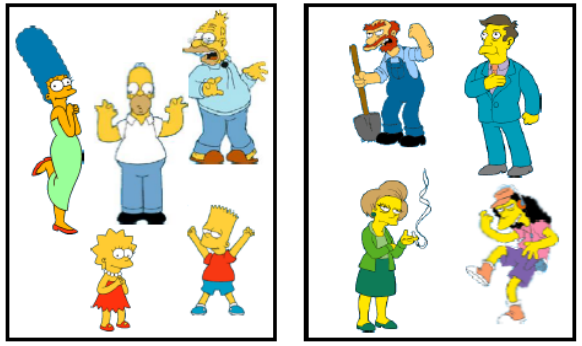
Bottom up or top down

Hierarchical



Top down

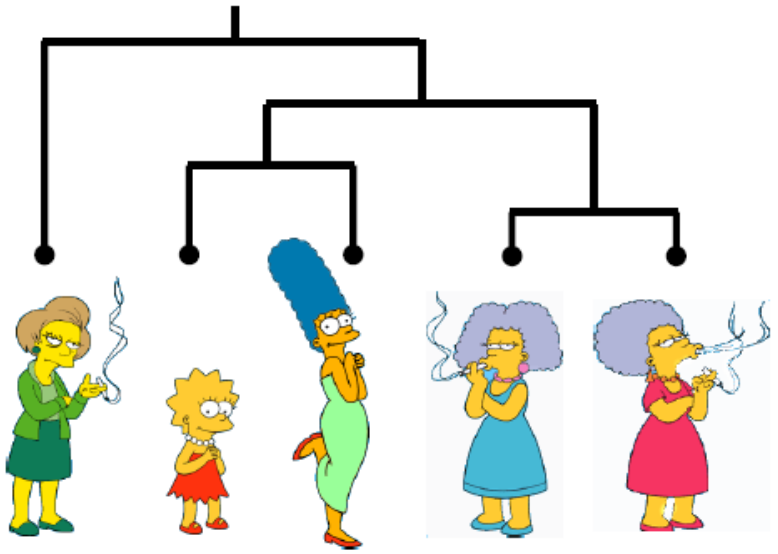
Partitional



Hierarchical Clustering

- **Agglomerative clustering**
  - **Bottom-Up (agglomerative):** Starting with each item in its own cluster, find the best pair to merge into a new cluster. Repeat until all clusters are fused together.
  - More popular hierarchical clustering technique
  - The number of possible dendrograms with n leafs = $\frac{(2n-3)!}{2^{(n-2)}(n-2)!}$

Number of Leafs	Number of Possible Dendrograms
2	1
3	3
4	15
5	105
...	…
10	34,459,425



Hierarchical Clustering

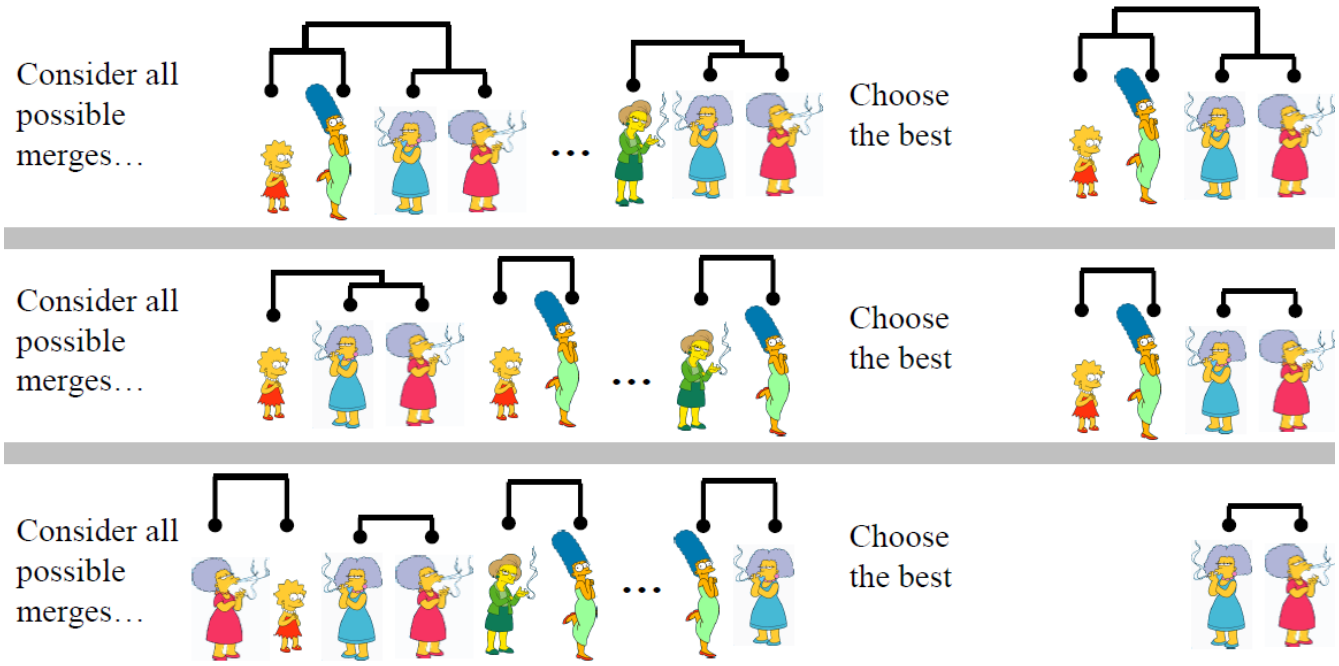
- Basic algorithm is straightforward
  - Key operation is the computation of the distance between two clusters
    - Different approaches to defining the distance between clusters distinguish the different algorithms.

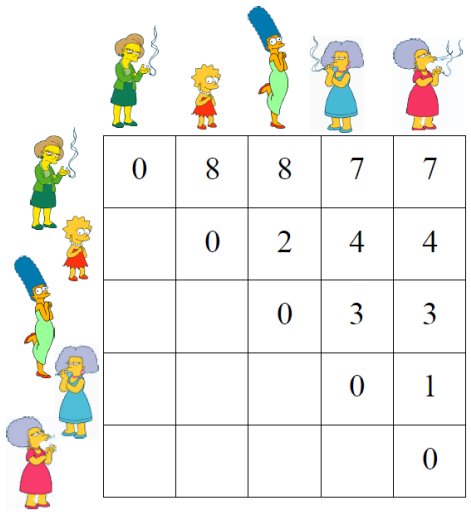
1.	Compute the distance matrix
2.	Let each data point be a cluster
3.	**Repeat**
4.	Merge the two closest clusters
5.	Update the distance matrix
6.	**Until** only a single cluster remains

- Defining distance measures
  - Definition: Let *O*1 and *O*2 be two objects from the universe of possible objects. The distance (dissimilarity) between *O*1 and *O*2 is a real number denoted by *D*(*O*1,*O*2)
  - Distance examples
    - Euclidian distance $d(x,y) = \sqrt{\sum_i (x_i - y_i)^2}$
    - Correlation coefficient (similarity) $s(x,y) = \frac{\sum_i (x_i - \mu_x)(y_i - \mu_y)}{\sigma_x \sigma_y}$

Hierarchical Clustering

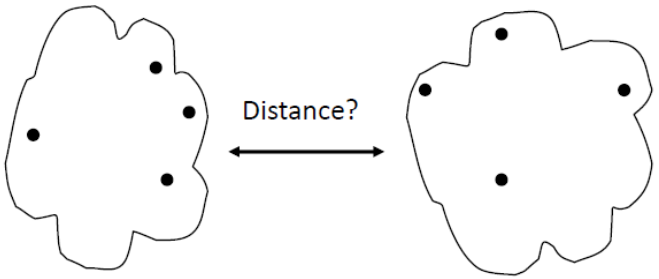
- Distance matrix





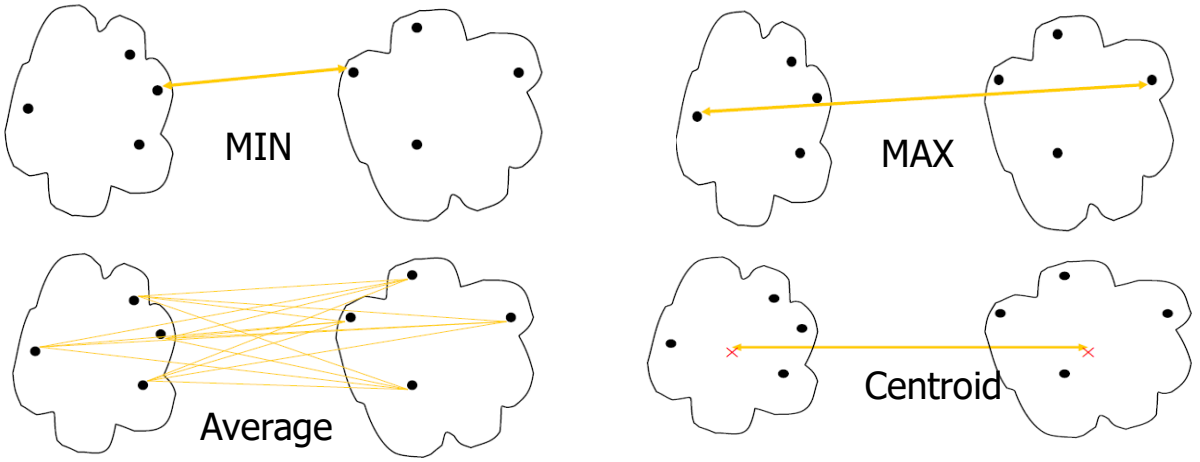
0	8	8	7	7
	0	2	4	4
		0	3	3
			0	1
				0

- But how do we compute distances between clusters rather than objects?



Hierarchical Clustering

- Distance between two clusters
  - MIN (or single link): the distance of the *closest pair of data objects* belonging to different clusters
  - MAX (or complete link): the distance of the *farthest pair of data objects* belonging to different clusters
  - Group Average (or average link): the *average distance of all pairs of data objects* belonging to different clusters
  - Centroid: the *distance between the centers of the clusters*



Hierarchical Clustering

- **Clustering methods: comparison**

	Hierarchical	K-means	GMM
Running time	naively, $O(N^3)$	fastest (each iteration is linear)	fast (each iteration is linear)
Assumptions	requires a similarity / distance measure	strong assumptions	strongest assumptions
Input parameters	none	K (number of clusters)	K (number of clusters)
Clusters	subjective (only a tree is returned)	exactly K clusters	exactly K clusters

Naïve Bayes Classifier

Bayes Theorem
Naïve Bayes Classifier

Bayes Theorem

- **Conditional probability model**
  - represented by a vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities for each of K possible outcomes or *classes* c_k.

$$p(c_k|\boldsymbol{x}) = p(c_k|x_1, x_2, \cdots, x_n)$$

  - Bayes Theorem

$$p(c_k|\boldsymbol{x}) = \frac{p(c_k)p(\boldsymbol{x}|c_k)}{p(\boldsymbol{x})}$$

$posterior = \frac{prior \times likehood}{evidence}$, $p(\boldsymbol{x}|c_k)$: complex calculation and need so many data

    - For a given n dimensional input feature data $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$, we want to calculate the probability that it is class c_k.

Bayesian Classification

- **Binary Classification**
  - Suppose we have two classes c_1 and c_2.

$$p(c_1|\boldsymbol{x}) = \frac{p(c_1)p(\boldsymbol{x}|c_1)}{p(\boldsymbol{x})}, p(c_2|\boldsymbol{x}) = \frac{p(c_2)p(\boldsymbol{x}|c_2)}{p(\boldsymbol{x})}$$

    - $p(\boldsymbol{x})$ is common so that we just need to compare $p(c_k)p(\boldsymbol{x}|c_k)$ probability.

- **Joint probability** $p(c_k, x_1, x_2, \cdots, x_n)$

$$\begin{aligned} p(c_k, x_1, x_2, \cdots, x_n) &= p(x_1, x_2, \cdots, x_n, c_k) \\ &= p(x_1|x_2, \cdots, x_n, c_k)p(x_2, \cdots, x_n, c_k) \\ &= p(x_1|x_2, \cdots, x_n, c_k)p(x_2|x_3, \cdots, x_n, c_k)p(x_3, \cdots, x_n, c_k) \\ &= \cdots \\ &= p(x_1|x_2, \cdots, x_n, c_k)p(x_2|x_3, \cdots, x_n, c_k) \cdots p(x_{n-1}|x_n, c_k)p(x_n|c_k)p(c_k) \end{aligned}$$

  - The 'naïve' conditional independence assumptions
    - All features $(x_1, x_2, \cdots, x_n)$ in $\boldsymbol{x}$ are mutually independent.

Naïve Bayes Classifier

Naïve Bayes Classifier

- From 'naïve' independent assumption

$$p(x_i|x_{i+1},\cdots,x_n,c_k) = p(x_i|c_k)$$

- The joint model can be expressed as

$$\begin{aligned} p(c_k|\boldsymbol{x}) &\propto p(c_k)p(\boldsymbol{x}|c_k) \\ &= p(c_k)\frac{p(c_k,x_1,x_2,\cdots,x_n)}{p(c_k)} = p(c_k,x_1,x_2,\cdots,x_n) \\ &= p(x_1|c_k)p(x_2|c_k)\cdots p(x_{n-1}|c_k)p(x_n|c_k)p(c_k) \\ &= p(c_k)\prod_{i=1}^{n} p(x_i|c_k) \end{aligned}$$

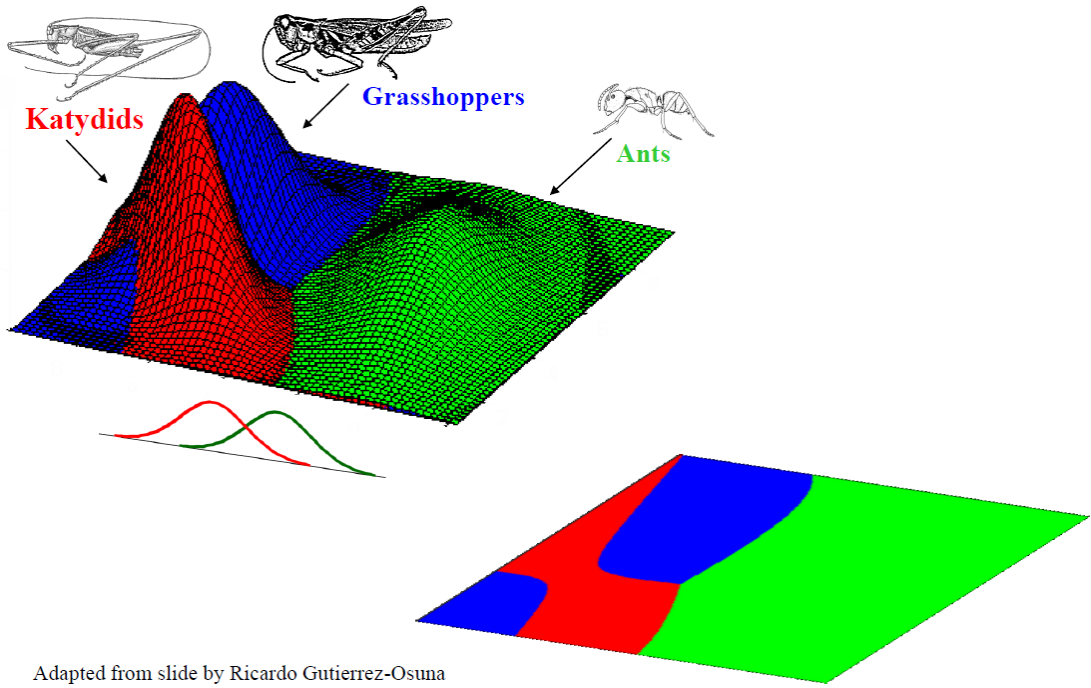
- Classification

$$\hat{y} = \underset{k\in\{1,\ldots,K\}}{\operatorname{argmax}}\ p(C_k)\prod_{i=1}^{n} p(x_i \mid C_k)$$

Naïve Bayes Classifier

- The Naïve Bayesian classifier has a piecewise quadratic decision boundary.

- Advantages
  - Fast to train and fast to classify
  - Not sensitive to irrelevant features
- Disadvantages
  - Assumes independence of features



Adapted from slide by Ricardo Gutierrez-Osuna

Support Vector Machine

SVM (linear)

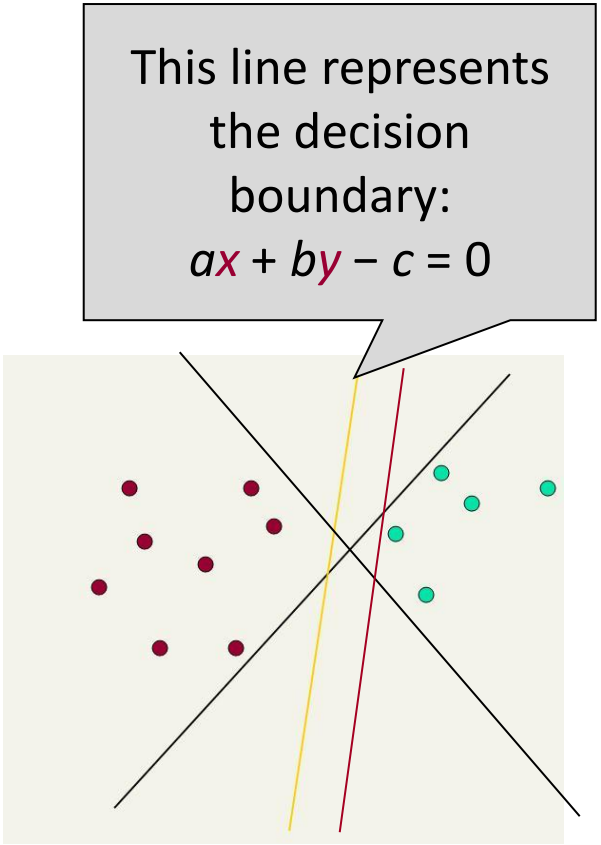
Decision Rule

Soft SVM

Kernel SVM (non-linear)

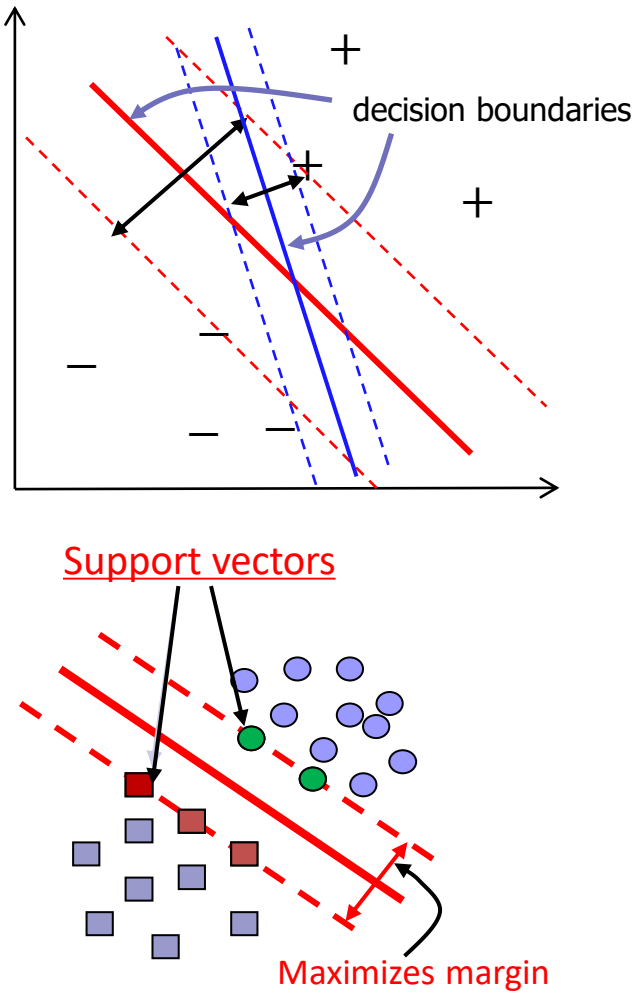
Linear Classifier

- **To divide two groups effectively: binary classification**
  - Find a decision boundary
  - lots of possible solutions for *a, b, c.*
- **Some methods find a separating hyperplane, but not the optimal one**
  - according to some criterion of expected goodness
  - E.g., Nearest Neighbor
  - Neural Network Perceptron



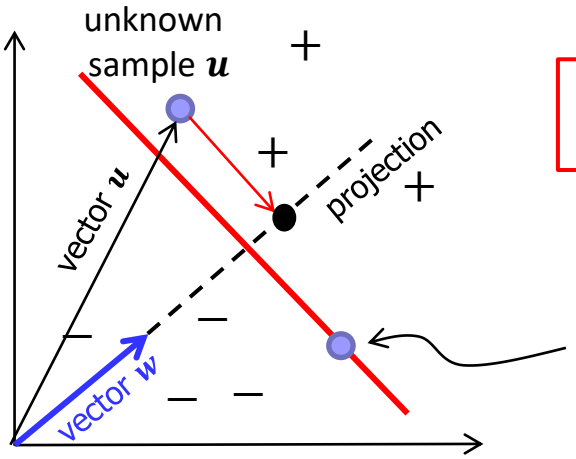
Linear Classifiers: Which Hyperplane

- **Motivation**
  - Find a straight line with widest street between two group
- **Support Vector Machine (SVM) finds an optimal solution.**
  - Maximizes the distance between the hyperplane and the "points" close to decision boundary
  - One intuition: if there are no points near the decision surface, then there are no very uncertain classification decisions
- **SVMs maximize the *margin* around the separating hyperplane.**
  - The decision function is fully specified by a subset of training samples, *the support vectors*.
  - Solving SVMs is a *quadratic programming* problem



Support Vector Machine (SVM)

- **Approach**



$\boldsymbol{w} \cdot \boldsymbol{u} \geq c \quad (c = -b)$

$\boldsymbol{w} \cdot \boldsymbol{u} + b \geq 0 \;\; then +, else \; -$

$\boldsymbol{decision\ rule}$

any point on the decision line

$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$

→ decison line is orthogonal with $\boldsymbol{w}$(normal)

many W possible (short or long)

vector inner(dot) product

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

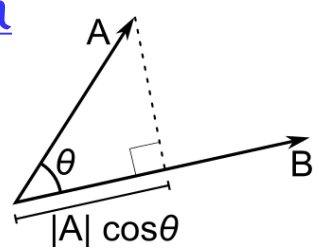
$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos(\theta) \qquad \|\mathbf{a}\| = \sqrt{\mathbf{a} \cdot \mathbf{a}}$$

where θ is the angle between **a** and **b**.

scalar projection of a vector **a** in the direction of a vector **b**

$$a_b = \mathbf{a} \cdot \hat{\mathbf{b}}$$

$$a_b = \|\mathbf{a}\| \cos\theta$$



where $\hat{\mathbf{b}} = \mathbf{b} / \|\mathbf{b}\|$ is the unit vector in the direction of **b**

Decision Boundary

- **What w and b should be selected?**
  - For fully separable (learning) samples, satisfy the following conditions

$$\boldsymbol{w} \cdot \boldsymbol{x}_+ + b \geq 1$$
$$\boldsymbol{w} \cdot \boldsymbol{x}_- + b \leq -1$$

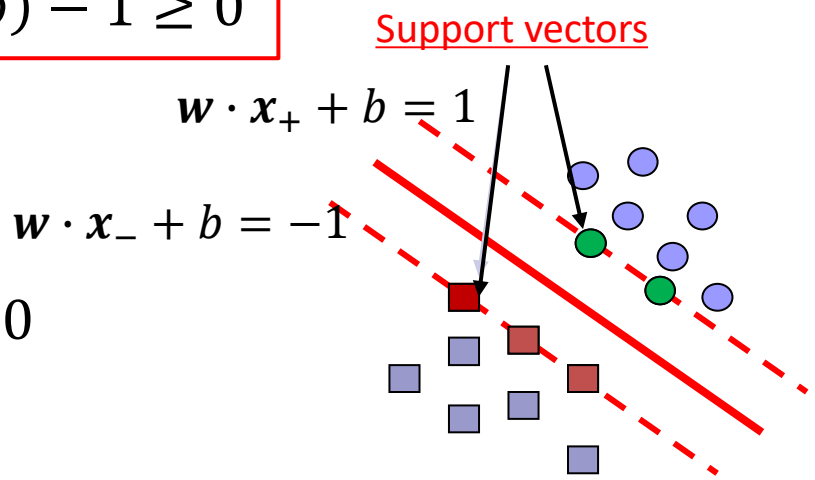
  - Define y_i such that $y_i = +1$ for $+$ samples, -1 for $-$ samples.

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1$$

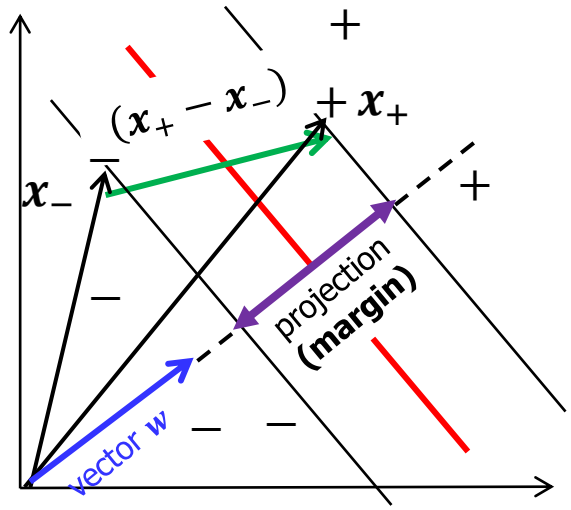
$$\boxed{y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1 \geq 0}$$

  - For $\boldsymbol{x_i}$ on the gutter

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1 = 0$$



Maximum Margin



- For gutter samples x_+ and x_-

$$margin = (x_+ - x_-) \cdot \frac{w}{\|w\|}$$

  - w is normal vector, unit vector is $\frac{w}{\|w\|}$.
  - Gutter samples

$$(w \cdot x_+ + b) - 1 = 0$$
$$-(w \cdot x_- + b) - 1 = 0$$
$$margin = \frac{2}{\|w\|}$$

- Determine w that maximize the margin

$$max \frac{2}{\|w\|} \rightarrow min \frac{\|w\|}{2} \quad with\ respect\ to\ y_i(w \cdot x_i + b) - 1 \geq 0\ \ for\ i = 1,2,\cdots,N$$
$$\rightarrow min \frac{\|w\|^2}{2} \quad (for\ mathematically\ convenient)$$

Constrained Optimization

- **Optimum w and b : Lagrangian Multipliers**

$$L = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i [y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1]$$

minimize w.r.t $\boldsymbol{w}, b$,and maximize w.r.t each $\alpha_i \geq 0$

$$\frac{\partial L}{\partial w} = \boldsymbol{w} - \sum_i \alpha_i y_i \boldsymbol{x_i} = 0 \quad \rightarrow \quad \boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i}$$

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i = 0 \quad \rightarrow \quad \sum_i \alpha_i y_i = 0$$

$$L = \frac{1}{2}\left(\sum_i \alpha_i y_i \boldsymbol{x_i}\right)\left(\sum_j \alpha_j y_j \boldsymbol{x_j}\right) - \sum_i \alpha_i y_i \boldsymbol{x_i} \cdot \left(\sum_j \alpha_j y_j \boldsymbol{x_j}\right) - \sum_i \alpha_i y_i b + \sum_i \alpha_i$$

$$L = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j \, y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$

find $\{\alpha_i\}$ that maximize L → can calculate $\boldsymbol{w}$ and b

Quadratic Programing

- Find $\alpha_1 \ldots \alpha_N$ such that $\sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$ is maximized and $\sum_i \alpha_i y_i = 0$, $\alpha_i \geq 0$ for all α_i
- Solution: Quadratic Programing

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T \underbrace{\begin{bmatrix} y_1 y_1 \boldsymbol{x}_1^T \boldsymbol{x}_1 & y_1 y_2 \boldsymbol{x}_1^T \boldsymbol{x}_2 & \cdots & y_1 y_N \boldsymbol{x}_1^T \boldsymbol{x}_N \\ y_2 y_1 \boldsymbol{x}_2^T \boldsymbol{x}_1 & y_2 y_2 \boldsymbol{x}_2^T \boldsymbol{x}_2 & \cdots & y_2 y_N \boldsymbol{x}_2^T \boldsymbol{x}_N \\ \vdots & \vdots & \vdots & \vdots \\ y_N y_1 \boldsymbol{x}_N^T \boldsymbol{x}_1 & y_N y_2 \boldsymbol{x}_N^T \boldsymbol{x}_1 & \cdots & y_N y_N \boldsymbol{x}_N^T \boldsymbol{x}_N \end{bmatrix}}_{\text{Quadratic Coefficients}} \boldsymbol{\alpha} + (-\mathbf{1}^T)\boldsymbol{\alpha}$$

$$\min_{\boldsymbol{\alpha}} \ \boldsymbol{\alpha}^T \boldsymbol{Q} \boldsymbol{\alpha} + (-\mathbf{1}^T)\boldsymbol{\alpha}$$

- Use MATLAB tool to solve Quadratic Programing

Decision Rule

- $\alpha_i > 0 \rightarrow$ $\boldsymbol{x}_i$ is a support vector.
  Solve for b using any support vector $\boldsymbol{x}_i$

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) = 1$$

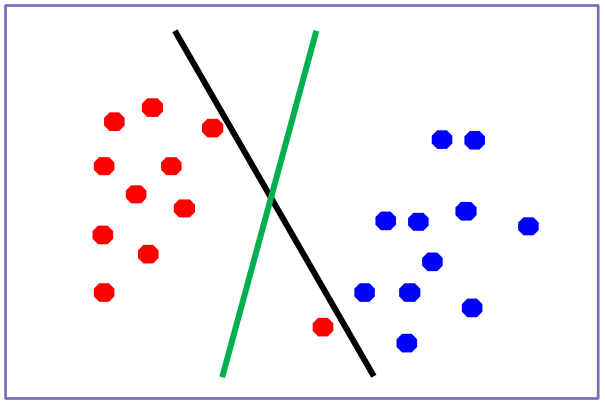
$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i} = \sum_{\boldsymbol{x_i}\ is\ SV} \alpha_i y_i \boldsymbol{x_i}$$

- **Decision Rule**
  - For unknown input $\boldsymbol{u}$

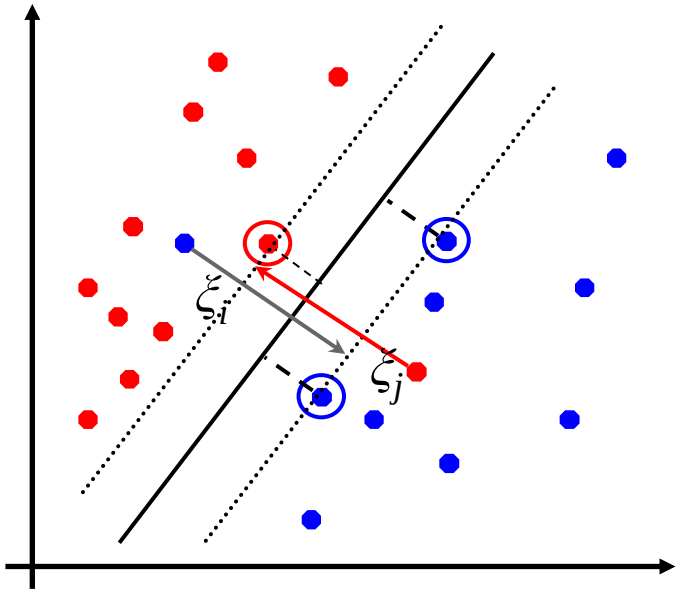
$$\boldsymbol{w} \cdot \boldsymbol{u} + b \geq 0 \ \text{ then } +$$
$$\rightarrow \quad \textstyle\sum_{\boldsymbol{i}} \alpha_i y_i \boldsymbol{x_i} \cdot \boldsymbol{u} + b \geq 0 \ \text{ then } +$$

Soft Margin Classification

- Which one is better? →



- If the training data is not linearly separable, ***slack variables*** ξ_i can be added to allow misclassification of difficult or noisy examples.
- **Allow some errors**
  - Let some points be moved to where they belong, at a cost
- Still, try to minimize training set errors, and to place hyperplane "far" from each class (large margin)



Soft Margin Classification

- **Consider slacks**

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0$$

  - Total violation $= \sum_{i=1}^{N} \xi_i$

- **The new optimization**

regularization

$$\text{Minimize} \quad \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{i=1}^{N} \xi_i$$

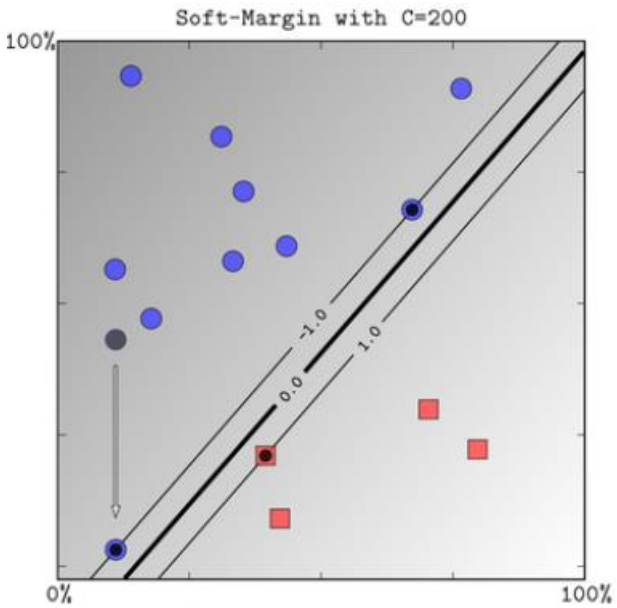
$$subject\ to \quad y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 - \xi_i \ \ and \ \ \xi_i \geq 0 \quad for\ i = 1, \cdots, N$$

  - Large C: more penalty on slacks → smaller margin
    - Fewer fails for training data (may result in overfitting)
  - Small C: allow larger slacks → larger margin
    - But, more fails for training data

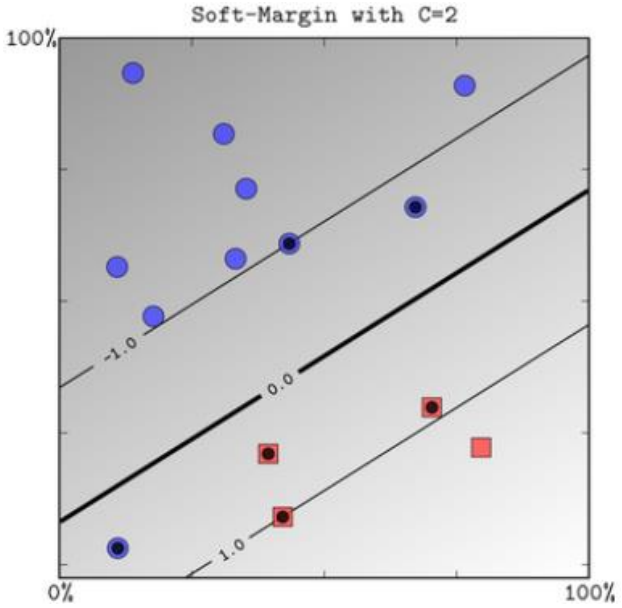
Soft Margin Classification

- **The effect of the soft-margin constant 'C'**

$$\text{Minimize} \quad \frac{\|\boldsymbol{w}\|^2}{2} + C\sum_{i=1}^{N}\xi_i$$



With a very high value of C, it mimics the behavior of the hard-margin SVM since it implies that the slack variables ξ_i have very high cost.



A smaller value of C allows us to ignore points close to the boundary, and increases the margin.

Soft Margin

Lagrange formulation

$$L(\boldsymbol{w}, b, \xi, \alpha, \beta) = \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i}_{\text{minimization goal}} - \underbrace{\sum_{i=1}^{N}\alpha_i[y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)-1+\xi_i] - \sum_{i=1}^{N}\beta_i\xi_i}_{\text{constraints}}$$

- Minimize w.r.t $\boldsymbol{w}, b, \xi$ and maximize w.r.t each $\alpha_i \geq 0$ and $\beta_i \geq 0$

$$\frac{\partial L}{\partial w} = \boldsymbol{w} - \sum_i \alpha_i y_i \boldsymbol{x_i} = 0 \quad \rightarrow \boldsymbol{w} = \sum_{\boldsymbol{i}} \alpha_i y_i \boldsymbol{x_i}$$

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i = 0 \quad \rightarrow \sum_i \alpha_i y_i = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \quad \rightarrow \quad C = \alpha_i + \beta_i$$

Solution for Soft Margin

Maximize

$$L = \frac{1}{2}\|\boldsymbol{w}\|^2 + (\alpha_i + \beta_i)\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(\boldsymbol{w}\cdot\boldsymbol{x_i} + b) - 1 + \xi_i] - \sum_{i=1}^{N}\beta_i\xi_i$$

$$= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N}\alpha_i[y_i(\boldsymbol{w}\cdot\boldsymbol{x_i} + b) - 1]$$

- It is the same as the previous case.
  - Neither slack variables ξ_i nor their Lagrange multipliers appear in the dual problem!

- $\alpha_i > 0 \rightarrow \boldsymbol{x}_i$ is a support vector.

Solve for b using any support vector $\boldsymbol{x}_i$

$$y_i(\boldsymbol{w}\cdot\boldsymbol{x_i} + b) = 1 - \xi_i = 1$$

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i} = \sum_{\boldsymbol{x_i}\ is\ SV} \alpha_i y_i \boldsymbol{x_i}$$

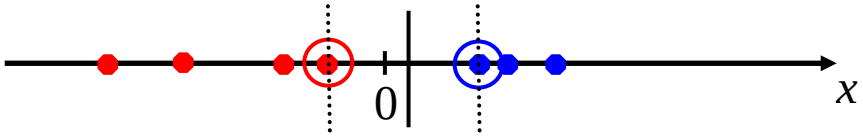
$for\ SV,\ \xi_i = 0$

$0 < \alpha_i < C \quad for\ SV,$

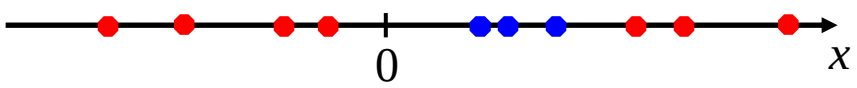
$(\alpha_i + \beta_i = C, \beta_i > 0)$

Non-linear SVMs

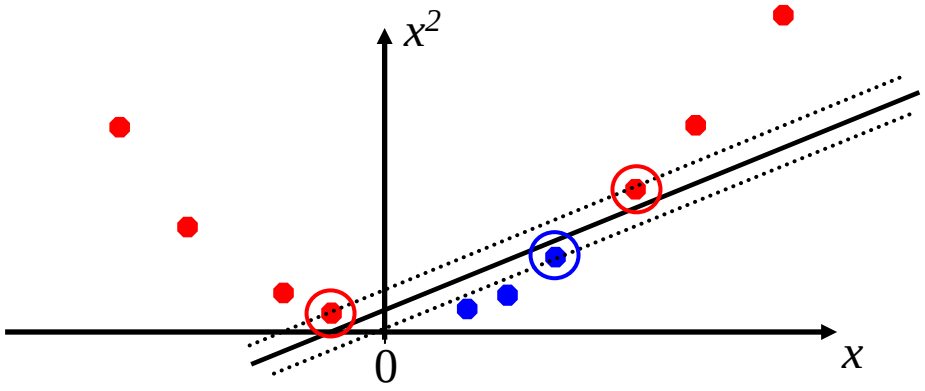
- Datasets that are linearly separable (with some noise) work out great:

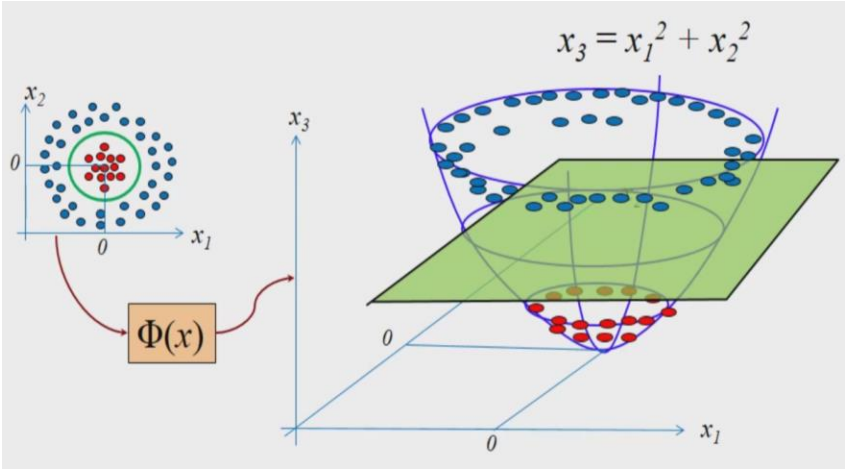


- But what are we going to do if the dataset is just too hard?



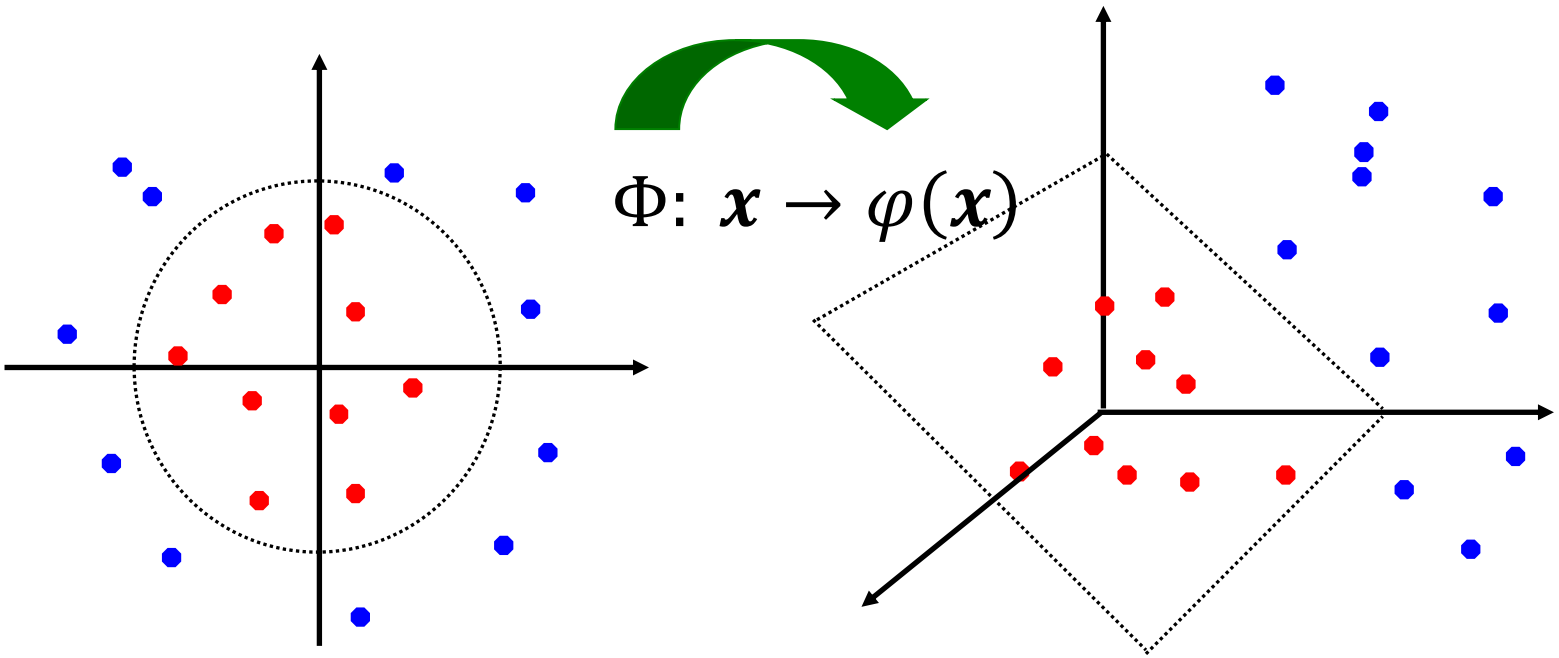
- How about … mapping data to a higher-dimensional space:





Non-linear SVMs: Feature Spaces

- **General idea:** the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:
- Kernel $\Phi: \boldsymbol{x} \rightarrow \varphi(\boldsymbol{x})$ e.g., $(x_1, x_2) \rightarrow (x_1, x_2, x_1^2 + x_2^2)$



The "Kernel Trick"

- **Non-linear SVM using Kernel**
  - Data transform to higher dimension, find the boundary, and make classification.
  - However, when there are more and more dimensions, computations within that space become more and more expensive.
- **Kernel trick**
  - It allows us to operate in the original feature space without computing the coordinates of the data in a higher dimensional space.
- **In SVM,**
  - To determine $\{\alpha_i\}$, we need to compute $\boldsymbol{x_i} \cdot \boldsymbol{x_j} \rightarrow \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j})$
  - To classify the test sample $\boldsymbol{u}$, we need to compute $\sum_{\boldsymbol{i}} \alpha_i y_i \boldsymbol{x_i} \cdot \boldsymbol{u} + b \rightarrow \sum_{\boldsymbol{i}} \alpha_i y_i \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{u}) + b$
  - Use Kernel function $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j})$
    - The Kernel function only uses $\boldsymbol{x_i}$ *and* $\boldsymbol{x_j}$

The "Kernel Trick"

- **The linear classifier:**
  - relies on an inner product between vectors $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i} \cdot \boldsymbol{x_j}$ in decision rule $(\sum_i \alpha_i y_i \boldsymbol{x_i} \cdot \boldsymbol{u} + b \geq 0 \text{ then } +)$
- **If every data point is mapped into high-dimensional space via some transformation Φ: $\boldsymbol{x} \rightarrow \varphi(\boldsymbol{x})$ the inner product becomes:**

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j})$$

  - A ***kernel function*** is some function that corresponds to an inner product in some expanded feature space.
  - Example: 2-dimensional vectors $\boldsymbol{x}_i = [x_{i1}, x_{i2}]$

let $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (1 + \boldsymbol{x_i} \cdot \boldsymbol{x_j})^2$. Need to show that $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j})$

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (1 + \boldsymbol{x_i} \cdot \boldsymbol{x_j})^2 = 1 + {x_{i1}}^2 {x_{j1}}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + {x_{i2}}^2 {x_{j2}}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$
$$= [1 \quad {x_{i1}}^2 \quad \sqrt{2}x_{i1}x_{i2} \quad {x_{i2}}^2 \quad \sqrt{2}x_{i1} \quad \sqrt{2}x_{i2}] \cdot [1 \quad {x_{j1}}^2 \quad \sqrt{2}x_{j1}x_{j2} \quad {x_{j2}}^2 \quad \sqrt{2}x_{j1} \quad \sqrt{2}x_{j2}]$$
$$= \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j}) \quad where, \varphi(\boldsymbol{x_i}) = [1 \quad {x_{i1}}^2 \quad \sqrt{2}x_{i1}x_{i2} \quad {x_{i2}}^2 \quad \sqrt{2}x_{i1} \quad \sqrt{2}x_{i2}]$$

Kernels

- **Why use kernels?**
  - Make non-separable problem separable.
  - Map data into better representational space
- **Common kernels**
  - Linear: $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i} \cdot \boldsymbol{x_j}$
  - Polynomial: $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (1 + \boldsymbol{x_i} \cdot \boldsymbol{x_j})^q$
    - gives feature conjunctions
  - RBF (Radial Basis Function)
    - infinite dimensional space

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-\|x_i - x_j\|^2 / 2\sigma^2}$$

$$\sigma: free\ parameter$$

Kernel SVM Final Decision

- **Final decision**
  - For test input $\boldsymbol{u}$: $g(\boldsymbol{u}) = sign(\boldsymbol{w} \cdot \varphi(\boldsymbol{u}) + b)$

$$\boldsymbol{w} = \sum_{\boldsymbol{z}_n \text{ is } SV} \alpha_n y_n \varphi(\boldsymbol{x_n})$$

$$g(\boldsymbol{u}) = sign\left(\sum_{\alpha_n > 0} \alpha_n y_n \varphi(\boldsymbol{x_n})\varphi(\boldsymbol{u}) + b\right)$$

$$= sign\left(\sum_{\alpha_n > 0} \alpha_n y_n K(\boldsymbol{x}_n, \boldsymbol{u}) + b\right)$$

$$\text{where } b = y_m - \sum_{\alpha_n > 0} \alpha_n y_n K(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

α_n is derived such that $\sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j \, y_i y_j K(\boldsymbol{x_i} \cdot \boldsymbol{x_j})$ is maximized.