Chapter 12

Big Data Mining - Classification

Acknowledgements

- Classification, Wikipedia. https://en.wikipedia.org/wiki/Classification
- Precision and recall, Wikipedia. https://en.wikipedia.org/wiki/Precision_and_recall
- k-nearest neighbors algorithm, Wikipedia. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- Decision tree, Wikipedia. https://en.wikipedia.org/wiki/Decision_tree
- Random forest, Wikipedia. https://en.wikipedia.org/wiki/Random_forest
- Naive Bayes classifier, Wikipedia. https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- Naive Bayes. http://scikit-learn.org/stable/modules/naive_bayes.html
- Logistic regression, Wikipedia. https://en.wikipedia.org/wiki/Logistic_regression
- What is Logistic Regression? http://www.statisticssolutions.com/what-is-logistic-regression/
- Support vector machine, Wikipedia. https://en.wikipedia.org/wiki/Support_vector_machine
- Introduction to Information Retrieval, Lecture 14: Support vector machines and machine learning on documents. Christopher Manning and Pandu Nayak, CS276.
- Kernel method, Wikipedia. https://en.wikipedia.org/wiki/Kernel_method
- Chapter 10, Neural Networks, The nature of code. http://natureofcode.com/book/chapter-10-neural-networks/
- An Introduction to Deep Learning. Professor David Wolfe Corne. Heriot-Watt University.
- Deep Learning History. http://blog.csdn.net/u012177034/article/details/52252851
- Convolutional neural network, Wikipedia. https://en.wikipedia.org/wiki/Convolutional_neural_network
- Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
- Recurrent neural network, Wikipedia. https://en.wikipedia.org/wiki/Recurrent_neural_network
- Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Autoencoder, Wikipedia. https://en.wikipedia.org/wiki/Autoencoder
- Introductory guide to Generative Adversarial Networks (GANs) and their promise! https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/
- Ensemble of Weak Learners. http://manish-m.com/?p=794

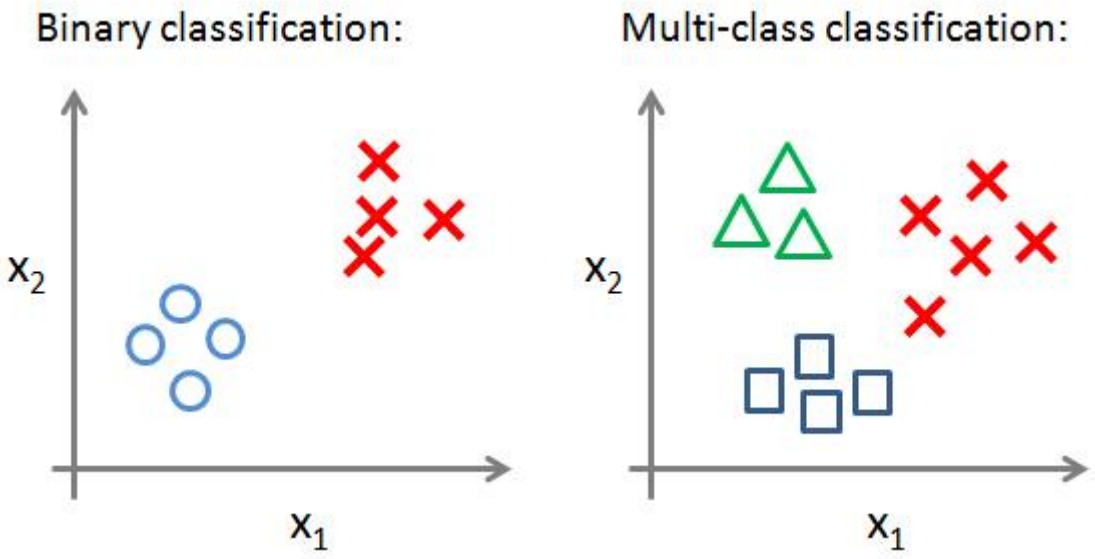
Chapter Outline

- Problem definition
- Warmups
  - *k*-NN
  - Decision tree
  - Naïve Bayes
  - Logistic Regression
  - Support Vector Machine
- Getting real
  - Neural Network
  - Extreme Learning Machine
  - Deep Learning
- Large-scale data
  - Ensemble
  - Distributed learning

What is classification?

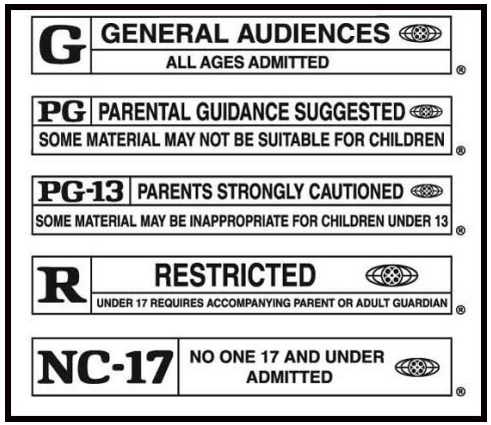
What is Classification?

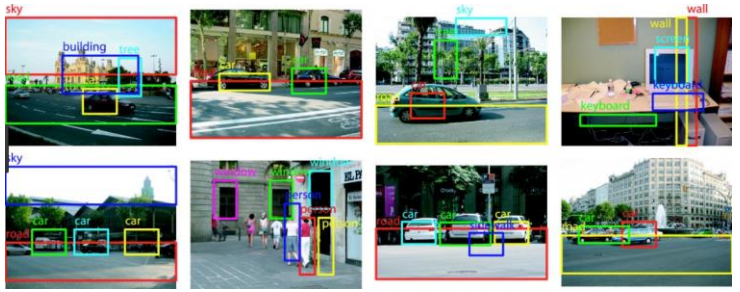
- Classification is a general process related to categorization, the process in which ideas and objects are recognized, differentiated, and understood.



Classification Applications

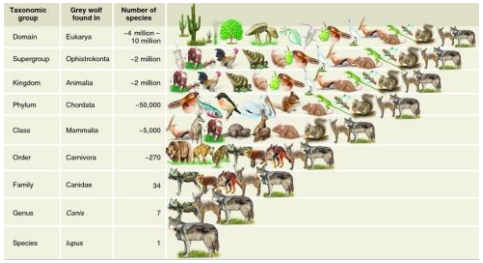
- Mathematics
  - **Mathematical classification**, a collection of sets which can be unambiguously defined by a property that all its members share
  - **Classification theorems** in mathematics
- Media
  - **Document classification**, a problem in library science, information science and computer science
  - **Library classification**, system of coding, assorting and organizing library materials according to their subject
  - **Classified information**, sensitive information to which access is restricted by law or regulation to particular classes of people
  - **Motion picture rating system**, for film classification
  - **Object recognition**

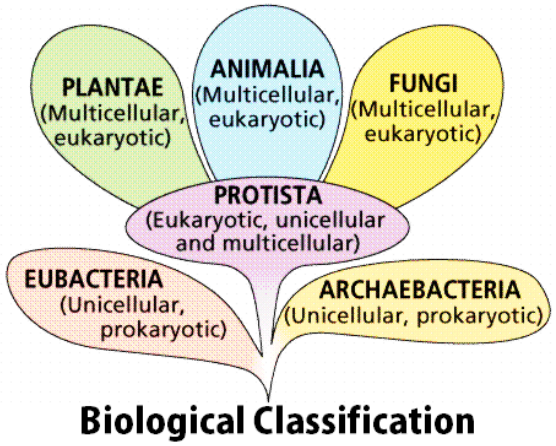




Classification Applications

- Science
  - **Chemical classification**
  - **Taxonomic classification**, also known as classification of species
  - **Biological classification** of organisms
  - **Medical classification**, the process of transforming descriptions of medical diagnoses and procedures into universal medical code numbers





MEDICAL DEVICE CLASSIFICATION IN INDIA

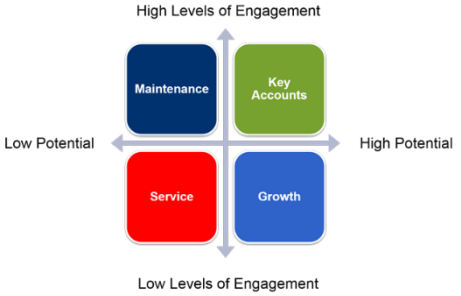
ACCORDING TO THE PROPOSED SCHEDULE M-III DRAFT

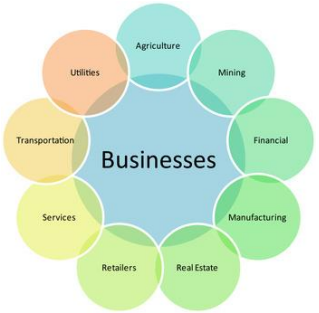
The draft Schedule M-III, released by the Central Drug Standards Control Organization (CDSCO) of India, includes a proposed risk classification for medical devices, based on their intended use.

Class	Risk Level	Device Examples
A	Low Risk	Thermometers / tongue depressors
B	Low-Moderate Risk	Hypodermic needles / suction equipment
C	Moderate-High Risk	Lung ventilator / bone fixation plate
D	High Risk	Heart valves / implantable defibrillator

Classification Applications

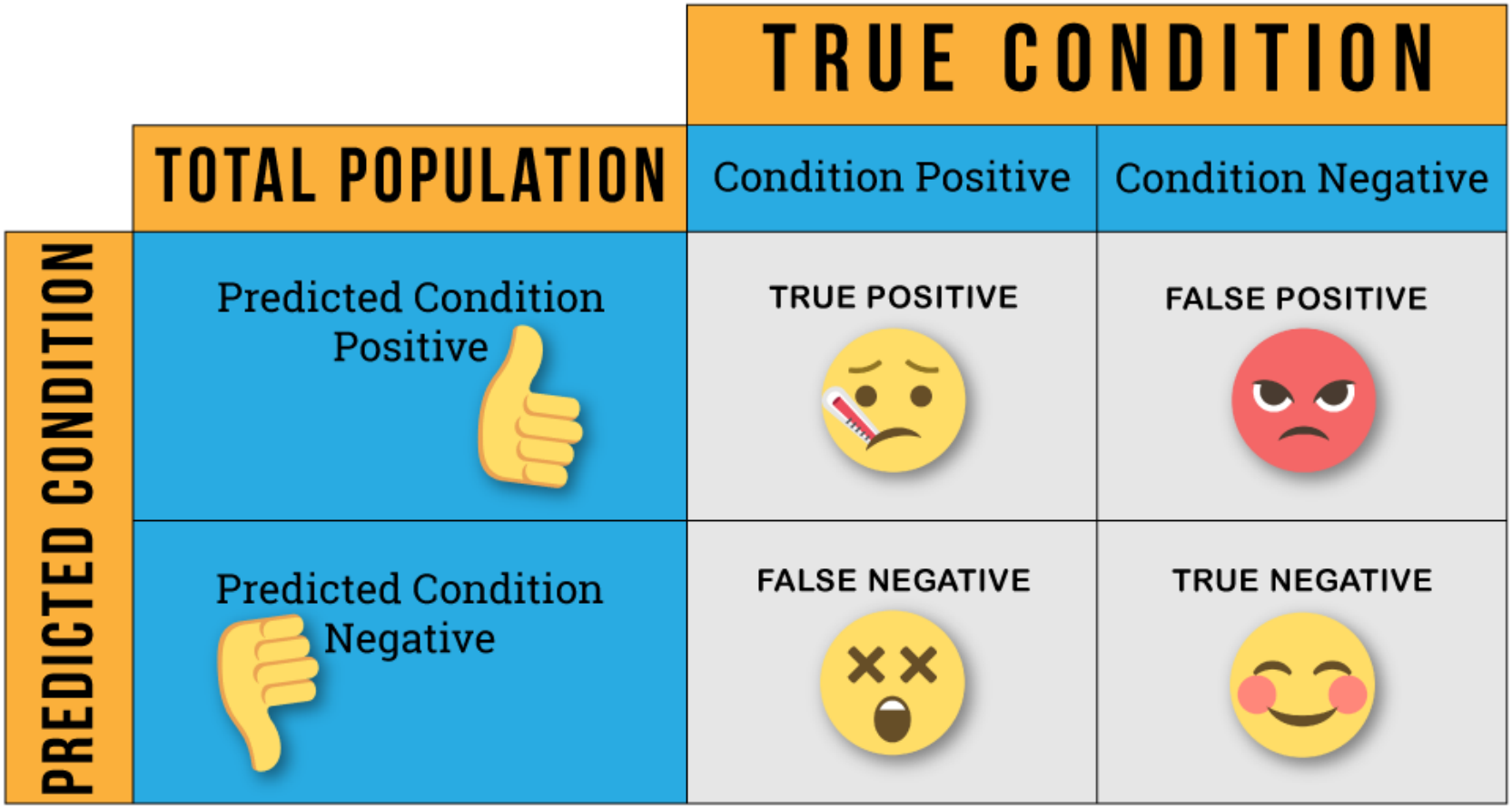
- Business, organizations and economics
  - **Classification of customers**, for marketing(as in Master data management) or for profitability (e.g. by Activity-Based Costing)
  - **Standard Industrial Classification**, economic activities
  - **Job classification**, as in job analysis
  - **Financial fraud detection**
- Many other uses …
  - Civil service classification
  - Product classification
  - …





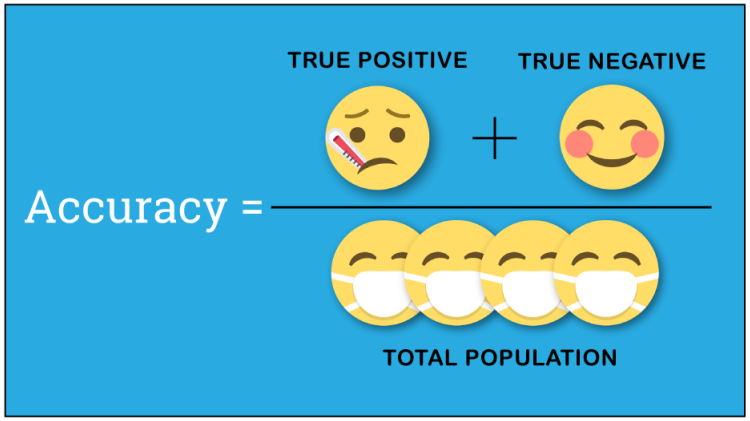
How to Evaluate?

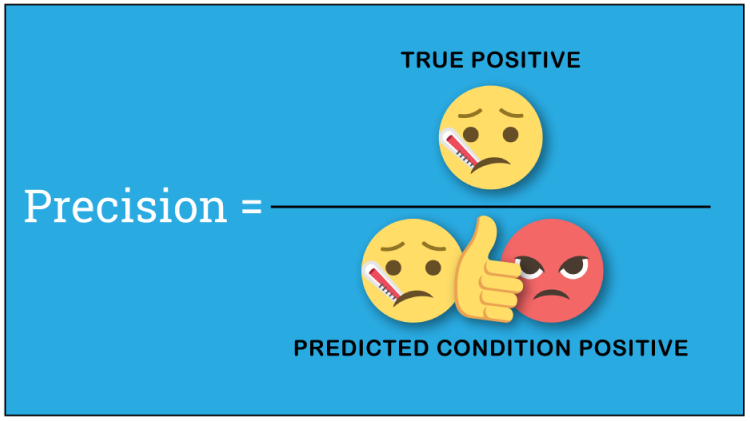
- Confusion matrix

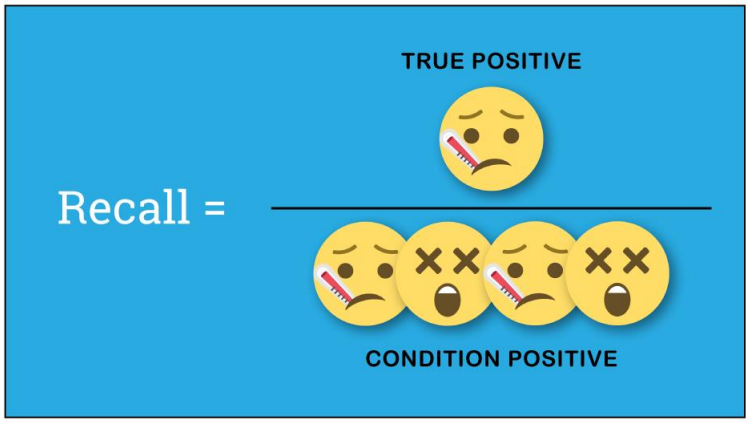


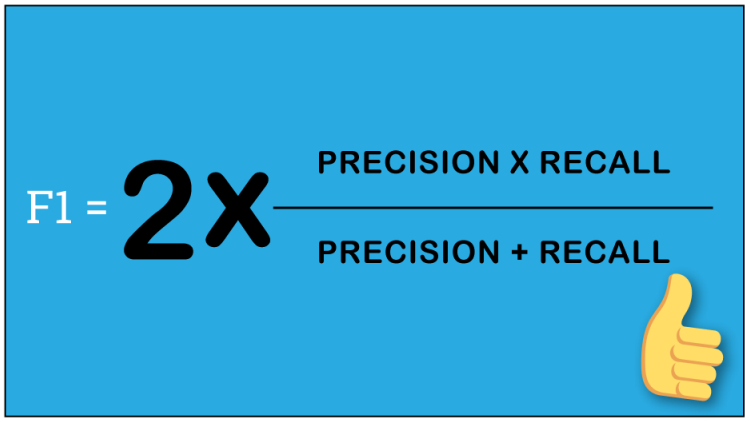
http://www.arkatechture.com/blog/ask-a-data-scientist-predicting-cancer-with-classifiers

How to Evaluate?

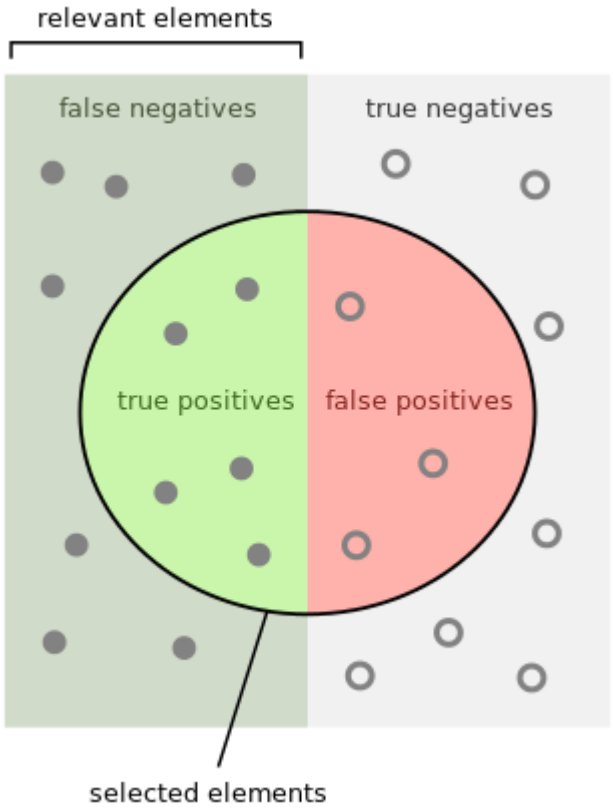








How to Evaluate?



How many selected items are relevant?

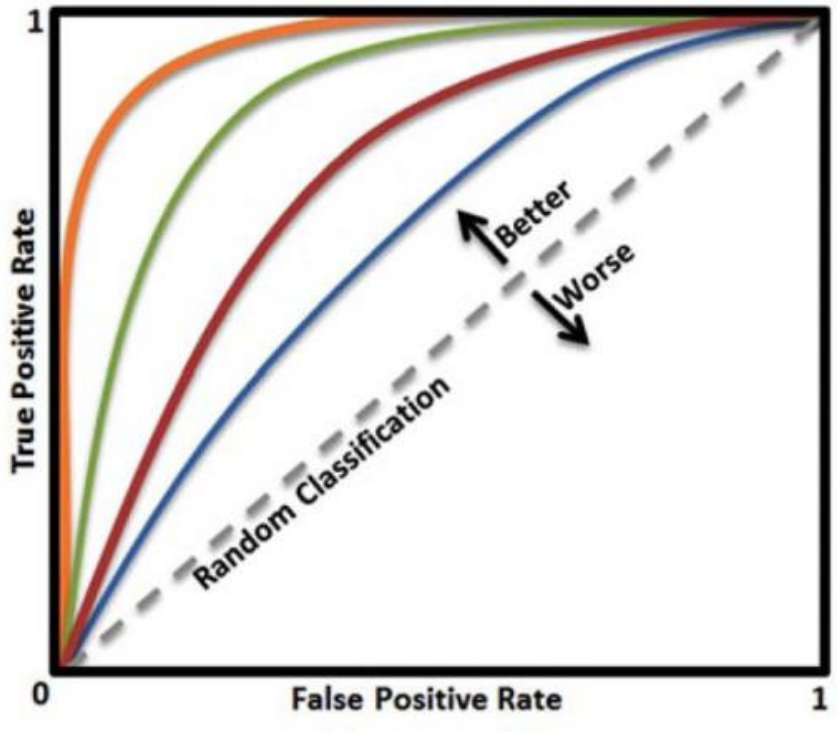
$$\text{Precision} = \frac{\text{true positives}}{\text{selected elements}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

How to Evaluate?

- ROC & AUC
  - Receiver Operating Characteristic
  - Area Under Curve



Warm-up Methods

Warm-ups

k-NN
Decision Tree
Naïve Bayes
Logistic Regression
Support Vector Machine

k-NN

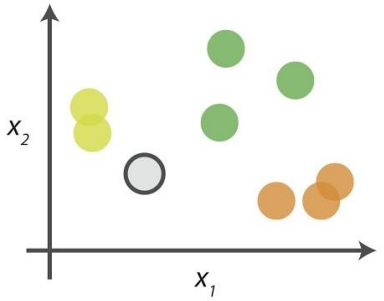
- ***k*-Nearest Neighbors** algorithm (*k*-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression:
  - In *k*-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
  - In *k*-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

- ***k*-NN** is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification.

k-NN

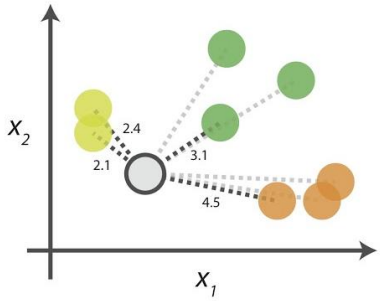
kNN Algorithm

0. Look at the data



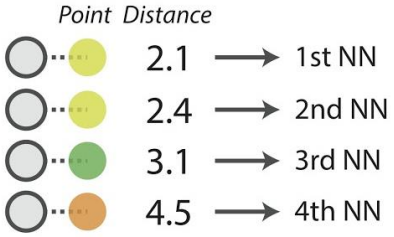
Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.

1. Calculate distances



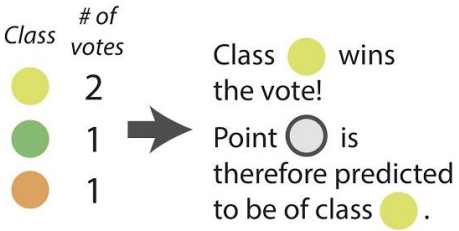
Start by calculating the distances between the grey point and all other points.

2. Find neighbours



Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.

3. Vote on labels



Vote on the predicted class labels based on the classes of the k nearest neighbours. Here, the labels were predicted based on the k=3 nearest neighbours.

Traits

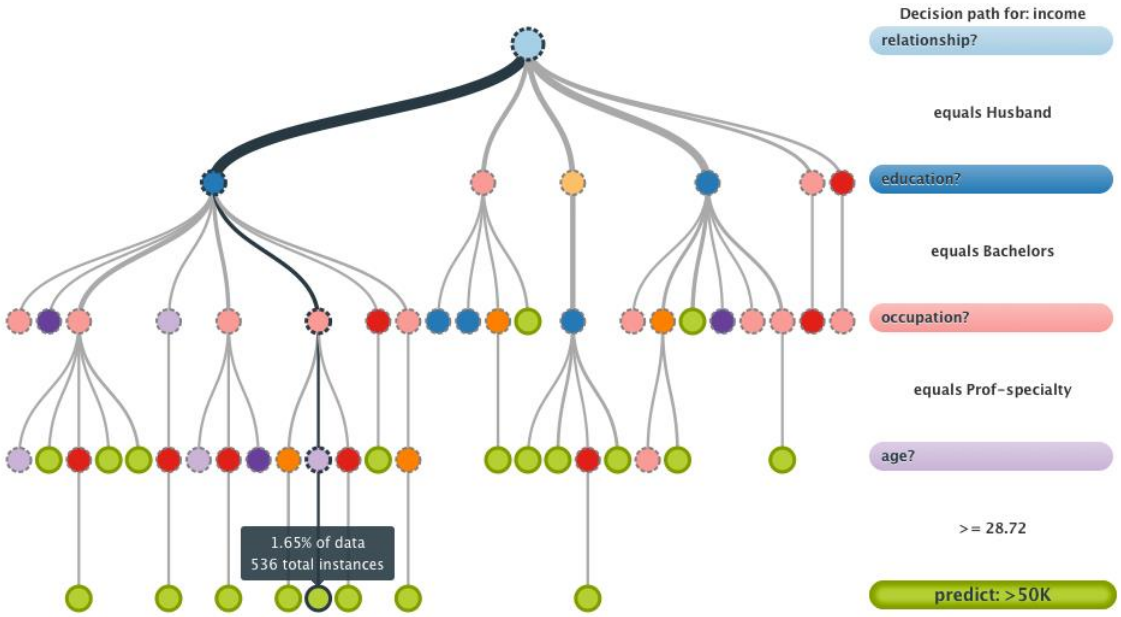
- Pros
  - Simple & easy for implementation
  - Training time is linear to the scale of training dataset
  - Suitable for multi-class classification
- Cons
  - Slow testing for lazy learning
  - High Calculation cost
  - Poor performance on imbalanced dataset
  - Parameter (*k*) sensitive

Warm-ups

k-NN
Decision Tree
Naïve Bayes
Logistic Regression
Support Vector Machine

Decision Tree

- A **decision tree** is a flowchart-like tree structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.



Dataset

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

There are 9 positive and 5 negative examples.

- *Humidity = High* has 3 positive and 4 negative.
- *Humidity = Normal* has 6 positive and 1 negative.
- *Wind = Weak* has 6 positive and 2 negative.
- *Wind = Strong* has 3 positive and 3 negative.

Which one is better as a root node, Humidity or Wind?

Decision Tree Construction

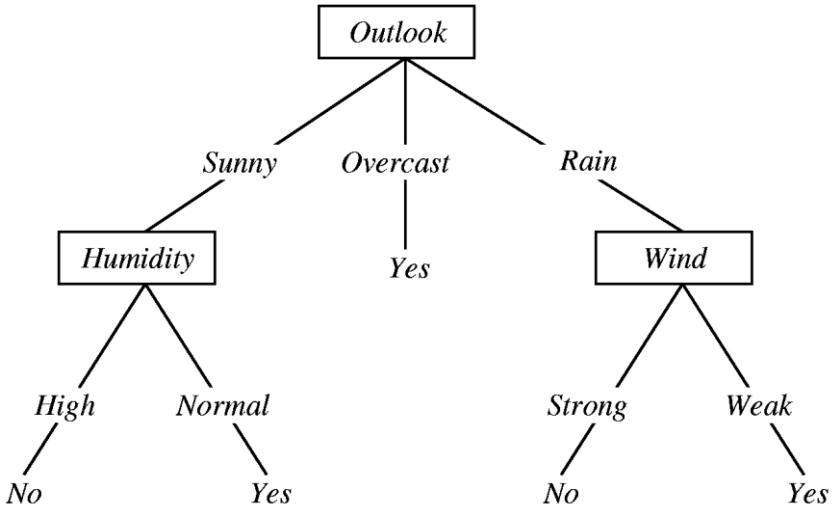
- Entropy

$$H(T) = I_E(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

- Information gain

$$\overbrace{IG(T,a)}^{\text{Information Gain}} = \overbrace{H(T)}^{\text{Entropy(parent)}} - \overbrace{H(T|a)}^{\text{Weighted Sum of Entropy(Children)}}$$

- Algorithms
  - ID3
  - C4.5
  - C5.0
- Output

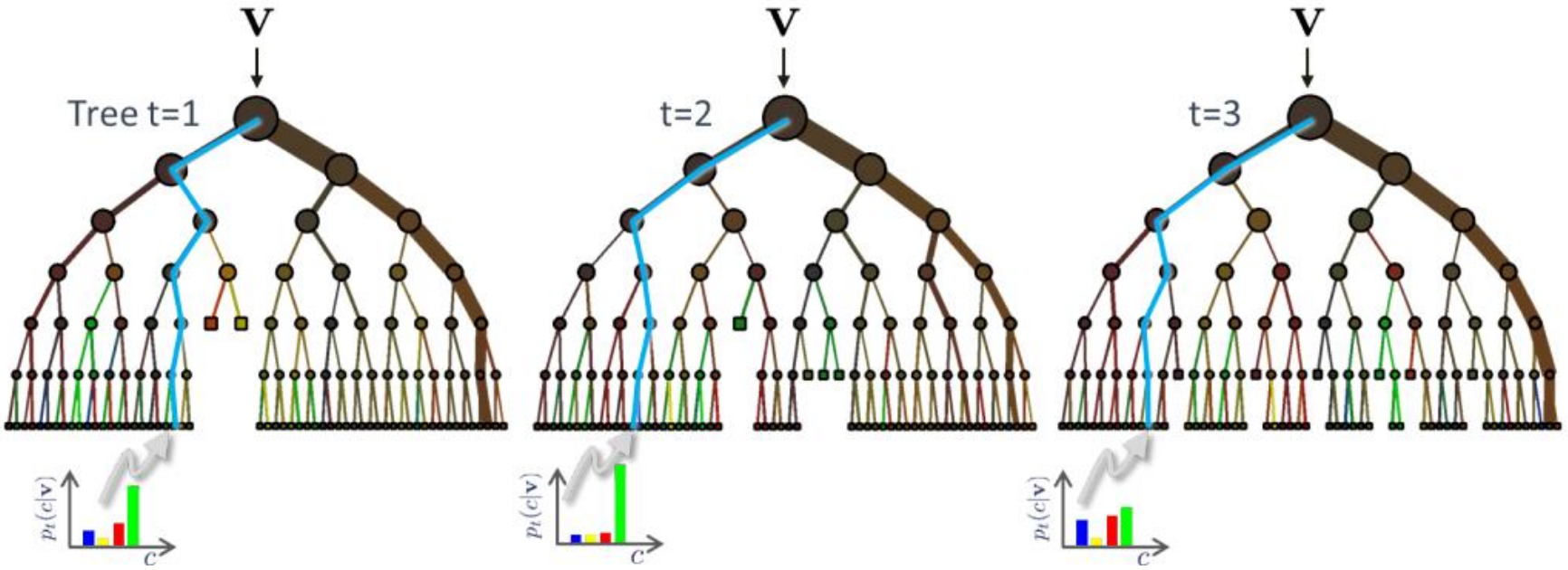


Traits

- Pros
  - Intuitive to understand
  - Easy to derive logical expressions
  - Simple preprocessing
  - Construction based on multiple attributes
- Cons
  - IG favors the attributes with multi-numeric values
  - Sensitive of noise data
  - Easy to overfit
  - Ignores the relationship between attributes
  - Cannot deal with missing attribute values

One more thing…

- Random Forest

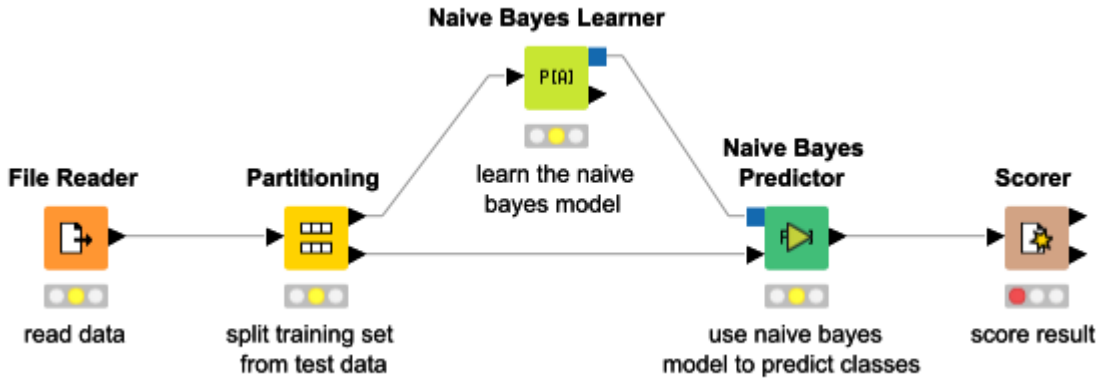


Warm-ups

k-NN
Decision Tree
Naïve Bayes
Logistic Regression
Support Vector Machine

Naïve Bayes

- Naive Bayes method is a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features.



Calculation

- Given a class variable *y* and a dependent feature vector x_1 through x_n, Bayes' theorem states the following relationship:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

- Using the naive independence assumption that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y),$$

- for all *i*, this relationship is simplified to

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

- Since P(x_1,…, x_n) is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

Traits

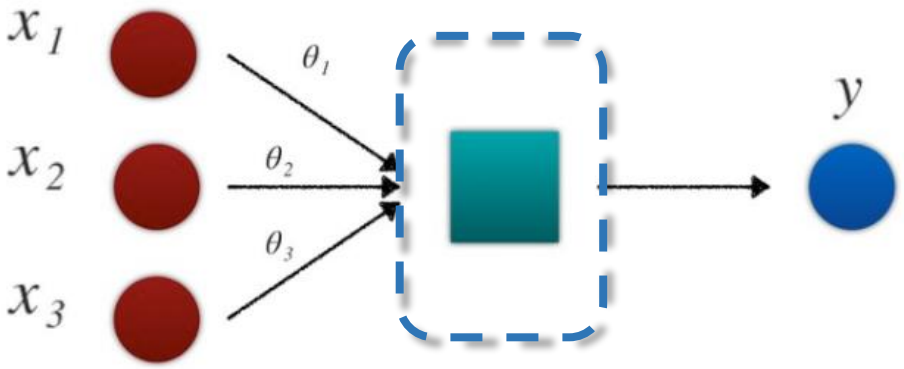
- Pros
  - Solid foundation of mathematics
  - Steady prediction
  - Easy to explain the result
  - Non-sensitive of parameters and missing data
  - No complex / iterative computation
- Cons
  - Naïve?
  - Requires prior probability

Warm-ups

k-NN
Decision Tree
Naïve Bayes
Logistic Regression
Support Vector Machine

Logistic Regression

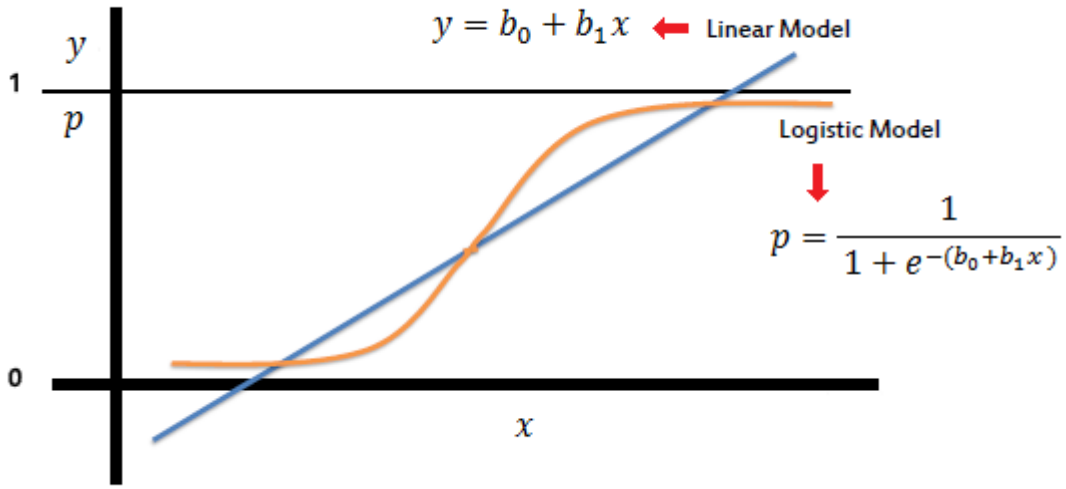
- Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.



Logistic Function

- Linear regression
  - least square method
  - gradient descent method
- Sigmoid function

$$g(z) = \frac{1}{1 + e^{-z}}$$



Traits

- Pros
  - Low computation cost
  - Easy to understand and implement
  - Suits regression and classification problem
- Cons
  - Easy to underfit
  - Low accuracy

Warm-ups

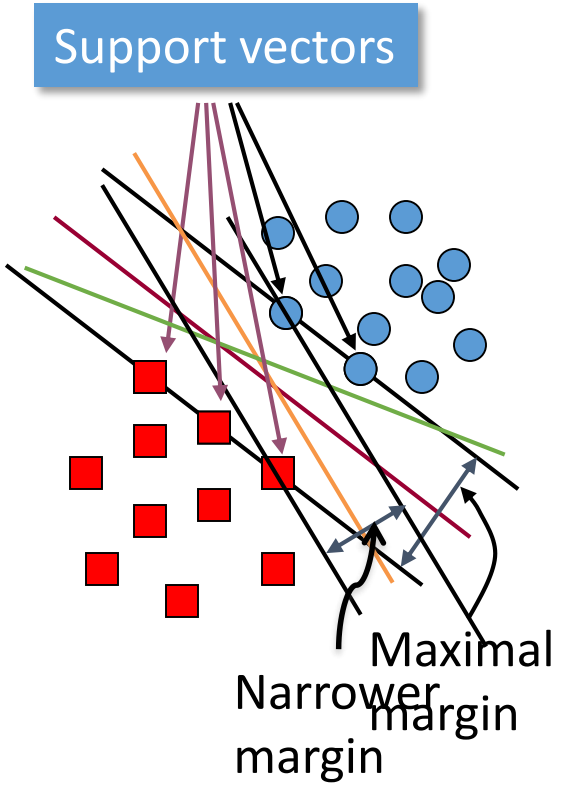
k-NN
Decision Tree
Naïve Bayes
Logistic Regression
Support Vector Machine

SVM

- Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting).
- An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Margin

- 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



Linear SVM

- Assume that all data is at least distance 1 from the hyperplane, then the following two constraints follow for a training set $\{(\mathbf{x}_i, y_i)\}$

$$\mathbf{w}^{\mathbf{T}}\mathbf{x}_{\mathbf{i}} + b \geq 1 \quad \text{if } y_i = 1$$

$$\mathbf{w}^{\mathbf{T}}\mathbf{x}_{\mathbf{i}} + b \leq -1 \quad \text{if } y_i = -1$$

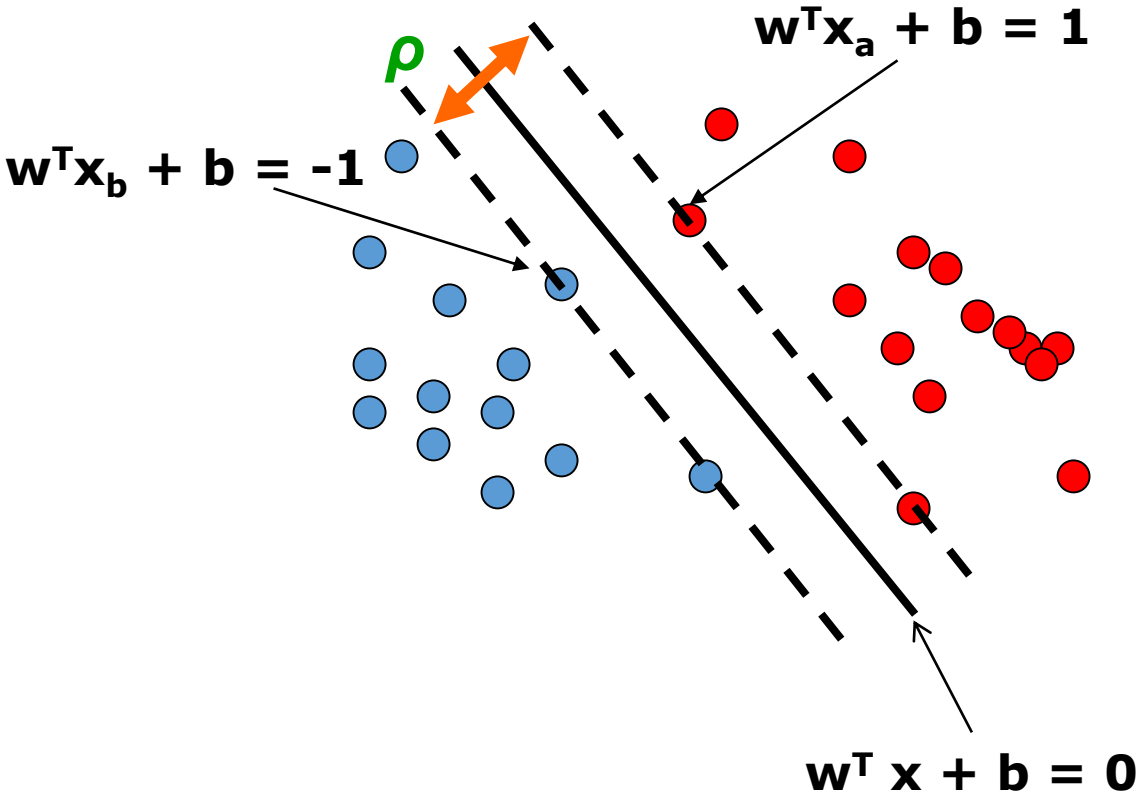
- For support vectors, the inequality becomes an equality
- Then, since each example's distance from the hyperplane is

$$r = y\frac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|}$$

- The margin is:

$$r = \frac{2}{\|\mathbf{w}\|}$$

Linear SVM



Mathematically

- Then we can formulate the *quadratic optimization problem:*

Find $\mathbf{w}$ and b such that

$r = \frac{2}{\|\mathbf{w}\|}$ is maximized; and for all $\{(\mathbf{x_i}, y_i)\}$

$\mathbf{w^T x_i} + b \geq 1$ if $y_i = 1$; $\mathbf{w^T x_i} + b \leq -1$ if $y_i = -1$

- A better formulation (min $||\mathbf{w}||$ = max $1/||\mathbf{w}||$):

Find $\mathbf{w}$ and b such that

$\Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ is minimized;

and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w^T x_i} + b) \geq 1$

Mathematically

- This is now optimizing a *quadratic* function subject to *linear* constraints
- Quadratic optimization problems are a well-known class of mathematical programming problem, and many (intricate) algorithms exist for solving them (with many special ones built for SVMs)
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^T\mathbf{x_j}$ is maximized and

(1) $\Sigma\alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

Mathematically

- The solution has the form:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^T \mathbf{x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0$$

- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x_i}^T \mathbf{x} + b$$

- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x_i}^T \mathbf{x_j}$ between all pairs of training points.

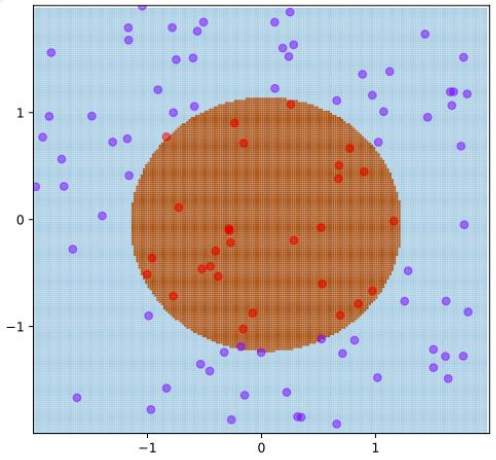
Kernel Trick

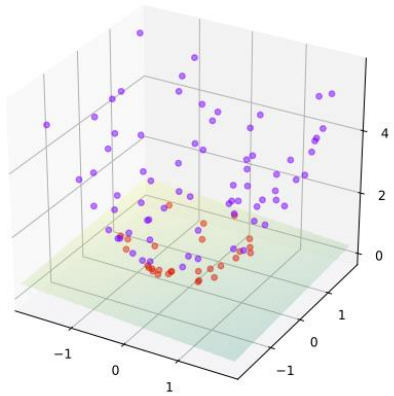
If the arguments to the kernel are in a measurable space X, and if the kernel is positive semi-definite—i.e.

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

for any finite subset $\{x_1, \ldots, x_n\}$ of **X** and subset $\{c_1, \ldots, c_n\}$ of objects (typically real numbers or even molecules)—then there exists a function φ(x) whose range is in an inner product space of possibly high dimension, such that

$$K(x, y) = \varphi(x) \cdot \varphi(y)$$





SVM with kernel given by $\phi((a, b)) = (a, b, a^2, b^2)$ and thus $K(\mathbf{x}, \mathbf{y}) = \mathbf{x} \bullet \mathbf{y} + \mathbf{x}^2\, \mathbf{y}^2$. The training points are mapped to a 3-dimensional space where a separating hyperplane can be easily found.

Traits

- Pros
  - Suits small-scale training dataset
  - Improved generalization performance
  - Solves high dimension disaster
  - Solves non-linear problem
  - Avoid neural network architecture selection and local minima
- Cons
  - Sensitive of missing data
  - No general solution to non-linear problem. Sensitive of kernel function selection

More Popular Methods

Advanced

Neural Network
Extreme Learning Machine
Deep Learning

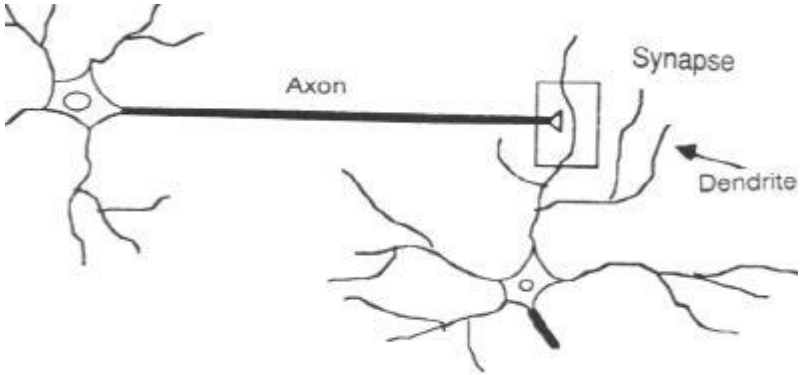
What is Neural Network?

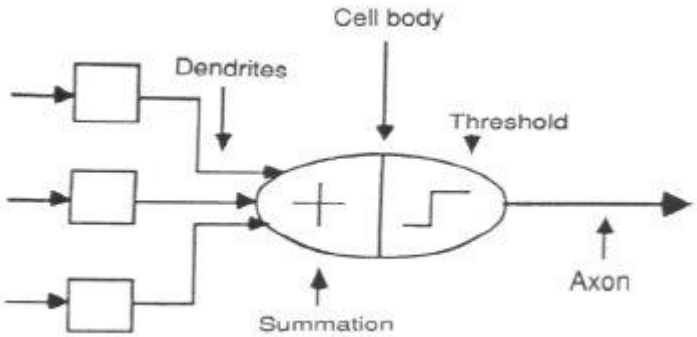
The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

"...*a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.* "

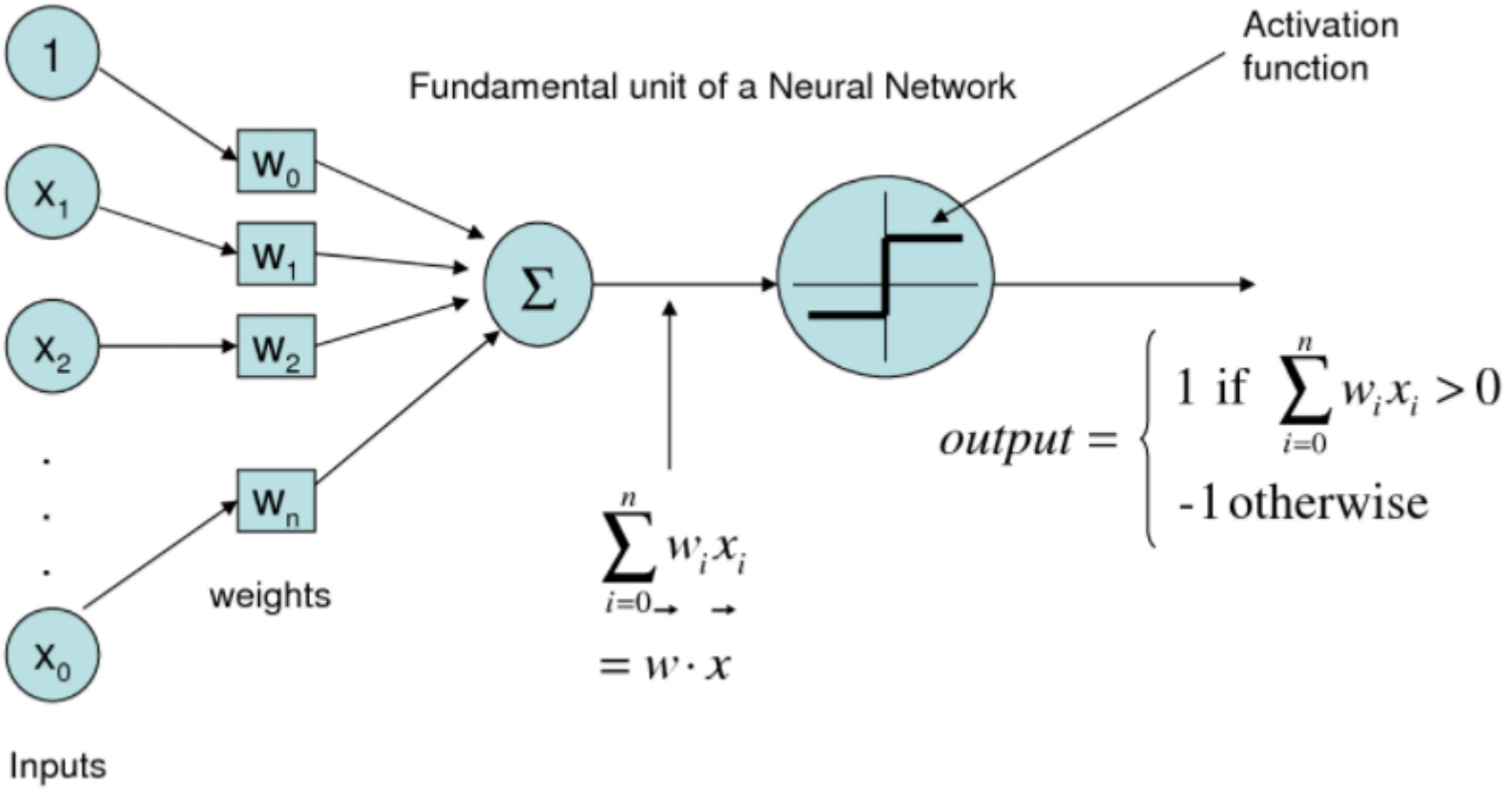
In "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989.

Human and Artificial Neurones



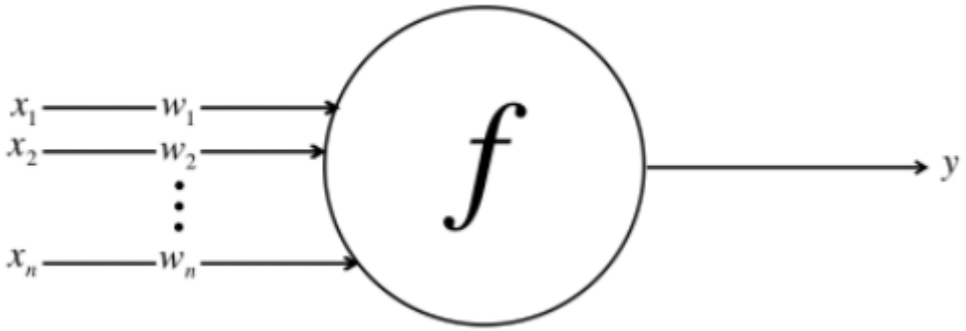


Neural Network Structure



Feedforward

- A perceptron follows the “feed-forward” model, meaning inputs are sent into the neuron, are processed, and result in an output.



- Steps
  1. Receive inputs (float[] inputs = {12 , 4};)
  2. Weight inputs (float[] weights = {0.5,-1}, bias = {2,1})
  3. Sum inputs (Sum = 8 + -3 = 5)
  4. Generate output (Output = sign(5) ⇒ +1)

Backpropagation

- The network can learn from its mistake and adjust its weights
  1. Provide the perceptron with inputs for which there is a known answer.
  2. Ask the perceptron to guess an answer.
  3. Compute the error. (Did it get the answer right or wrong?)
  4. Adjust all the weights according to the error.
  5. Return to Step 1 and repeat!

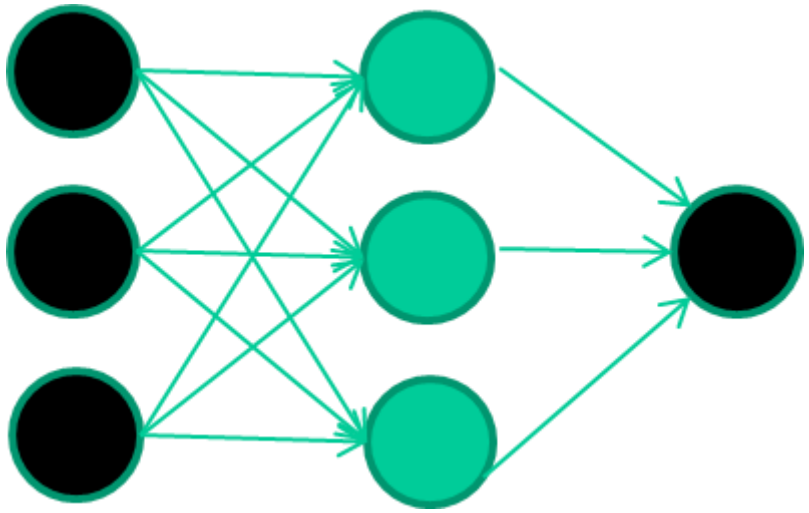
- Calculations
  1. ERROR = DESIRED OUTPUT - GUESS OUTPUT
  2. STEERING = DESIRED VELOCITY - CURRENT VELOCITY
  3. NEW WEIGHT = WEIGHT + ΔWEIGHT
  4. ΔWEIGHT = ERROR * INPUT
  5. NEW WEIGHT = WEIGHT + ERROR * INPUT
  6. NEW WEIGHT = WEIGHT + ERROR * INPUT * LEARNING CONSTANT

Neural Network Construction

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Initialise with random weights

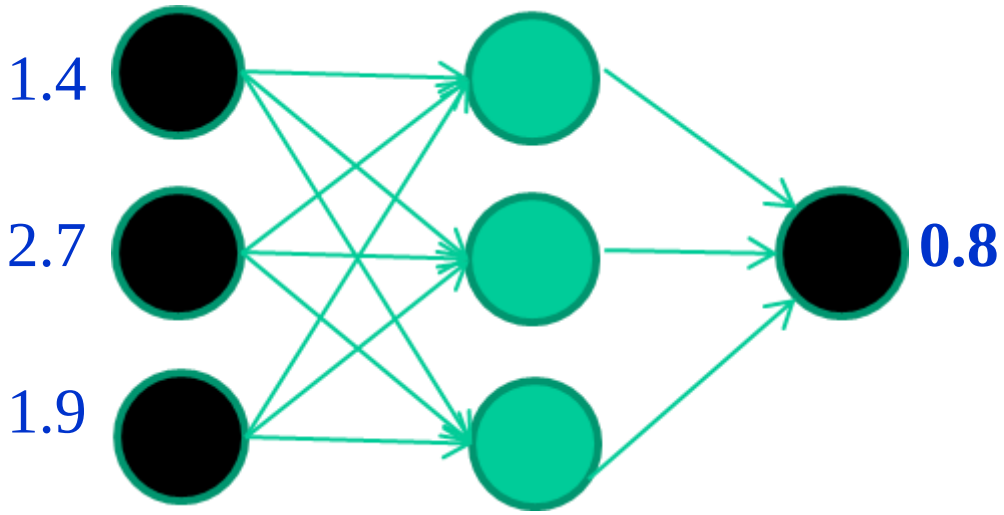


Neural Network Construction

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get output

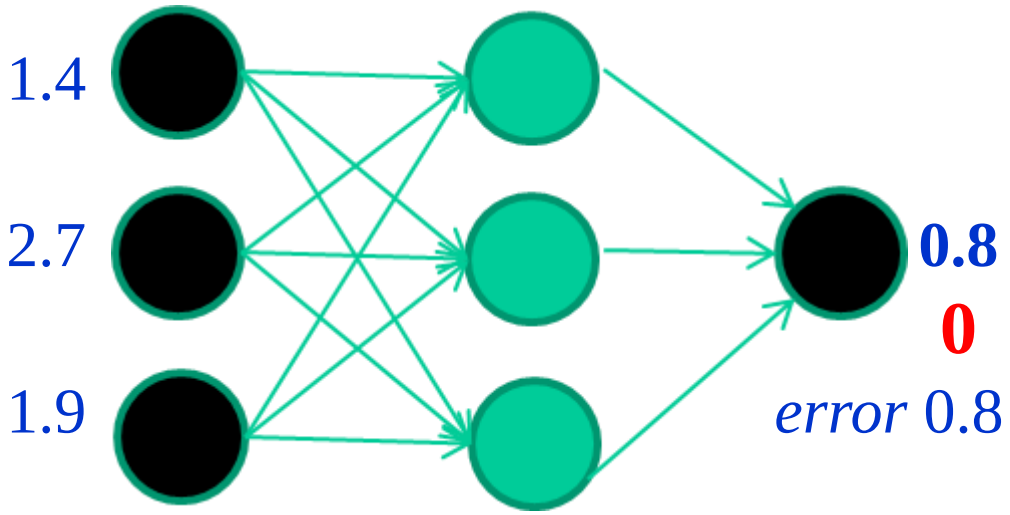


Neural Network Construction

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Compare with target output

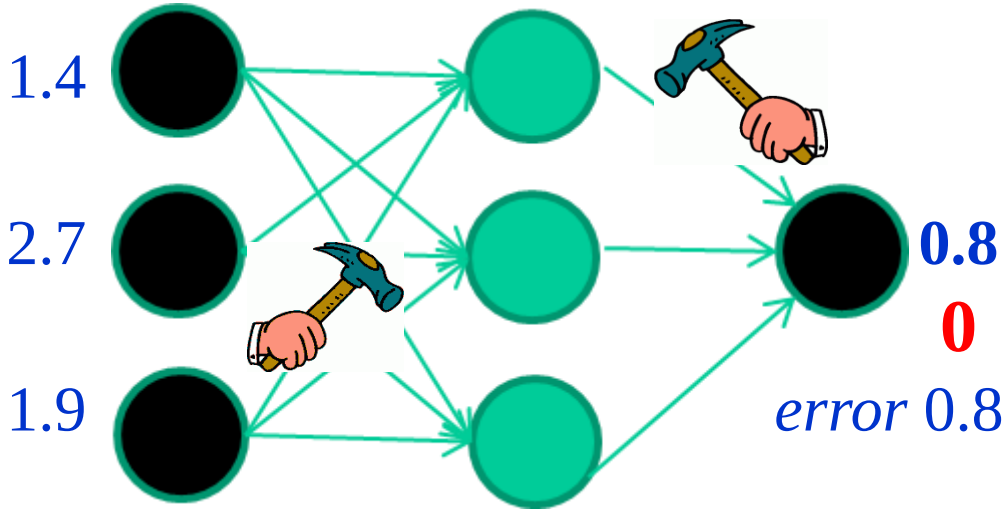


Neural Network Construction

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error

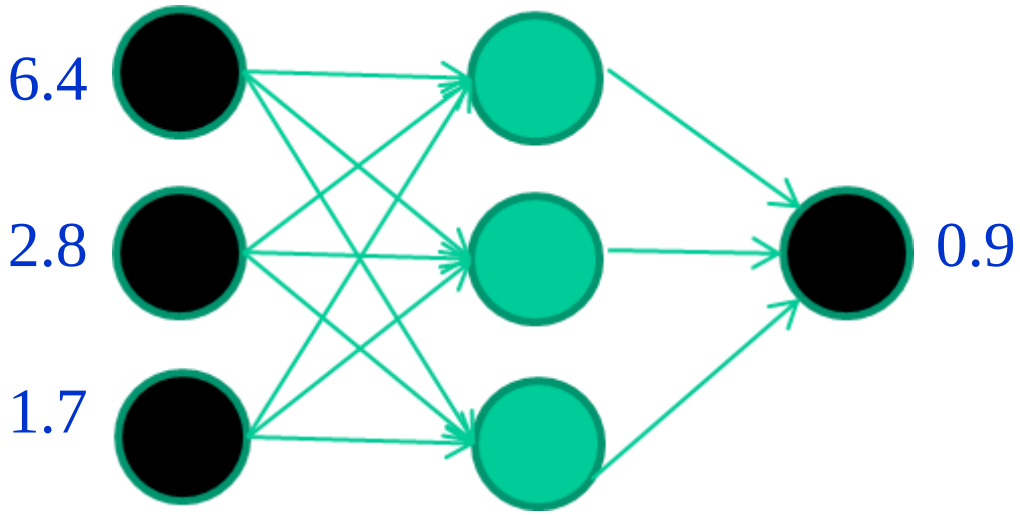


Neural Network Construction

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get output

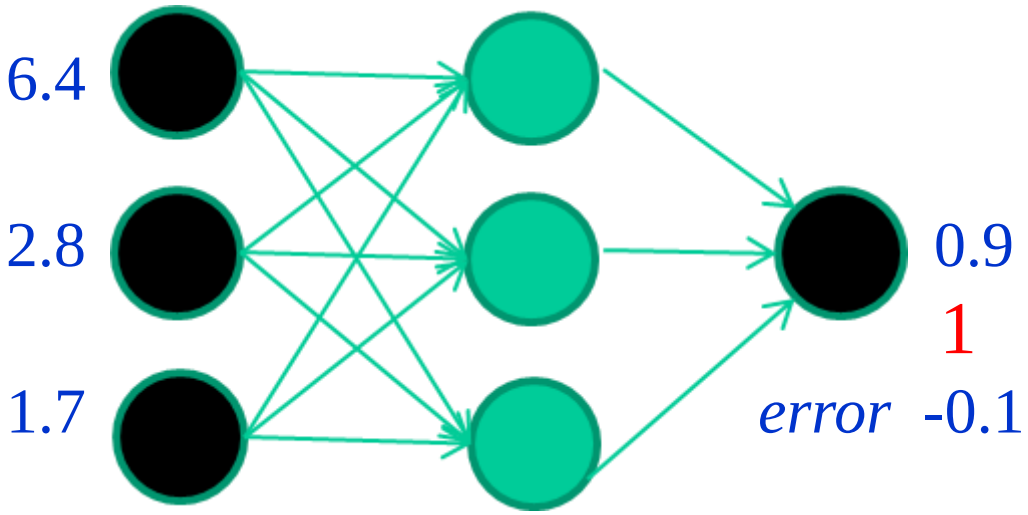


Neural Network Construction

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Compare with target output

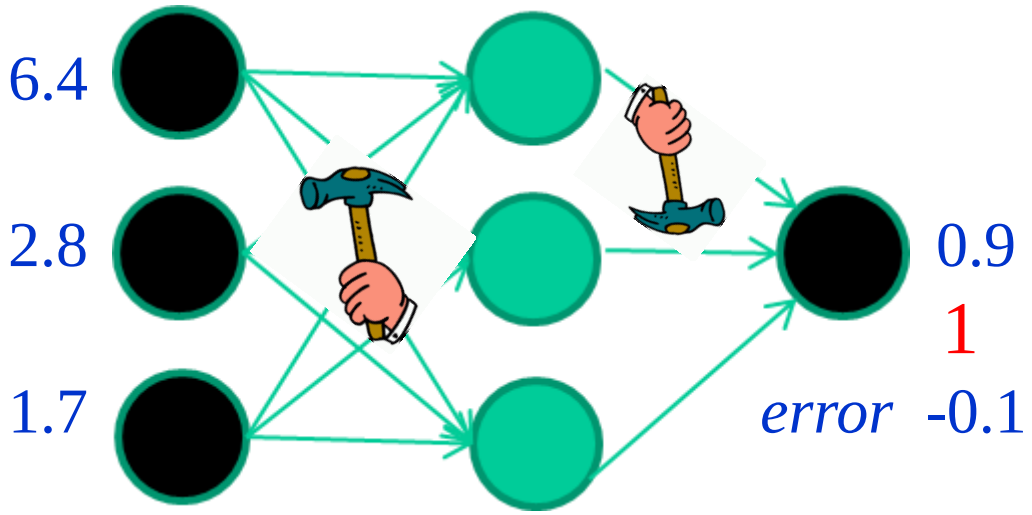


Neural Network Construction

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

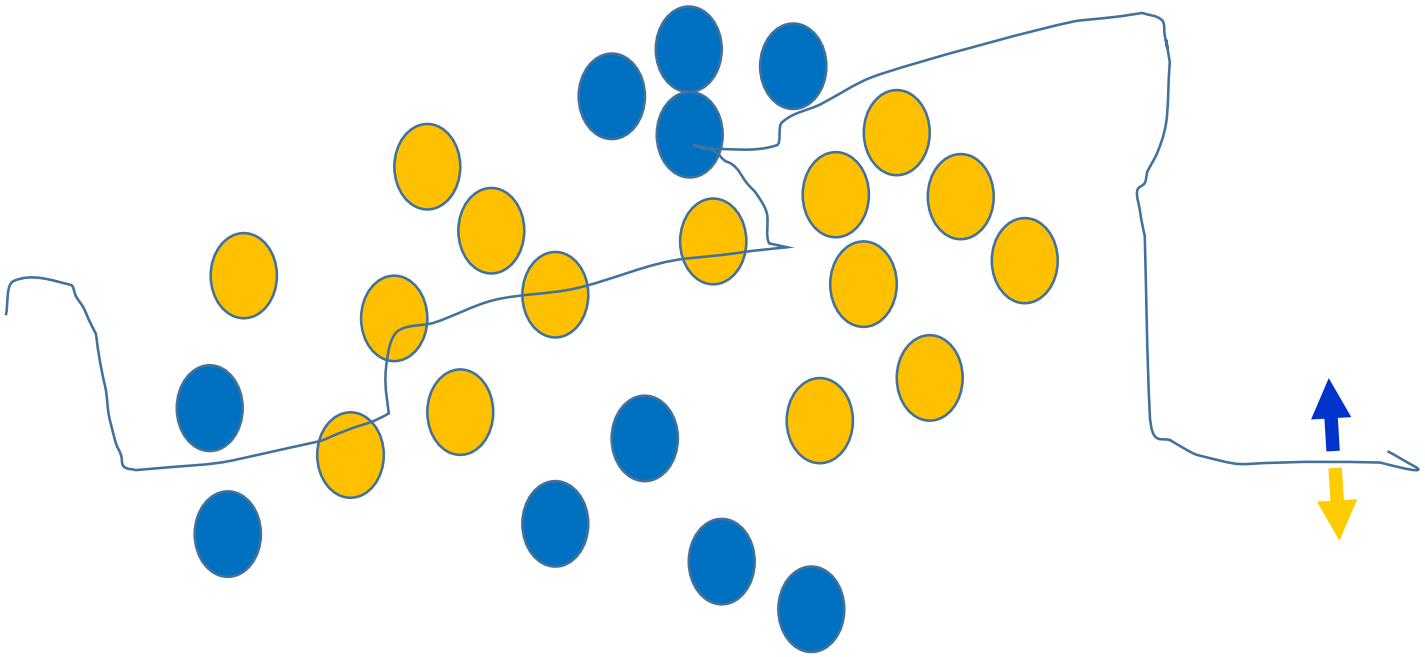
Adjust weights based on error



Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments!

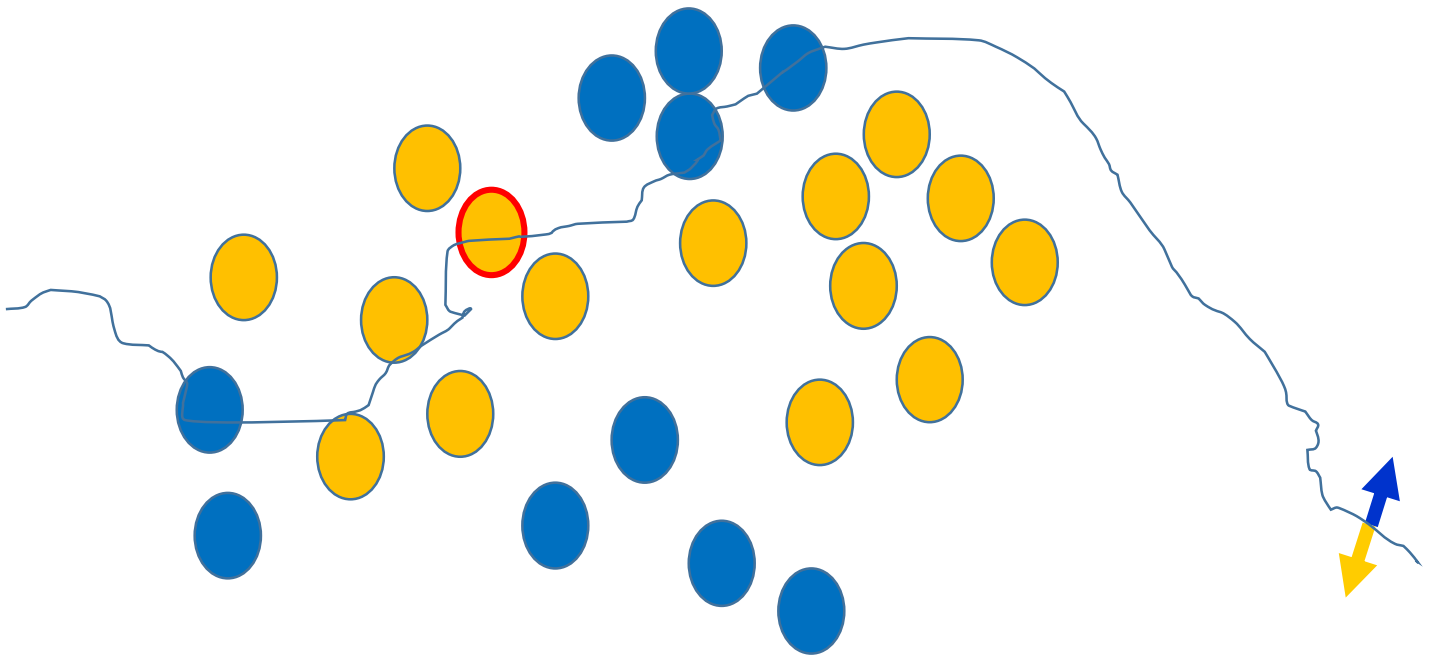
Decision Boundary

Initial random weights



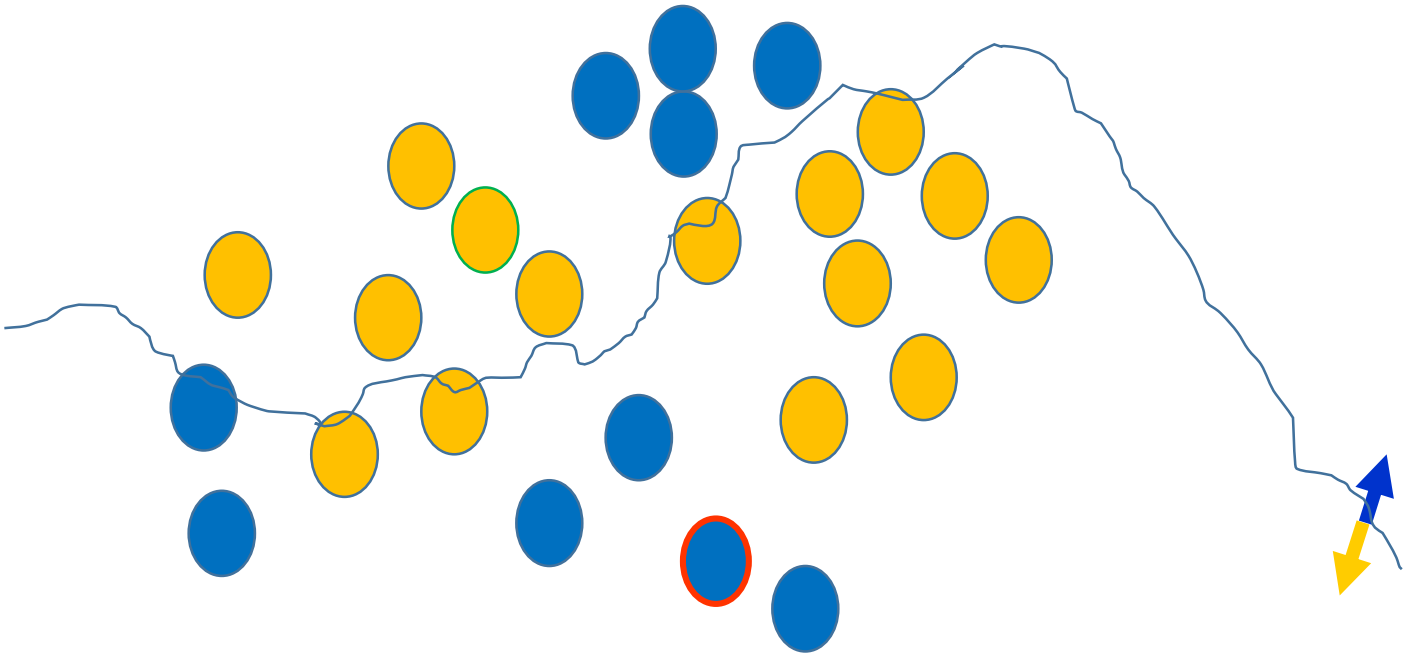
Decision Boundary

Present a training instance / adjust the weights



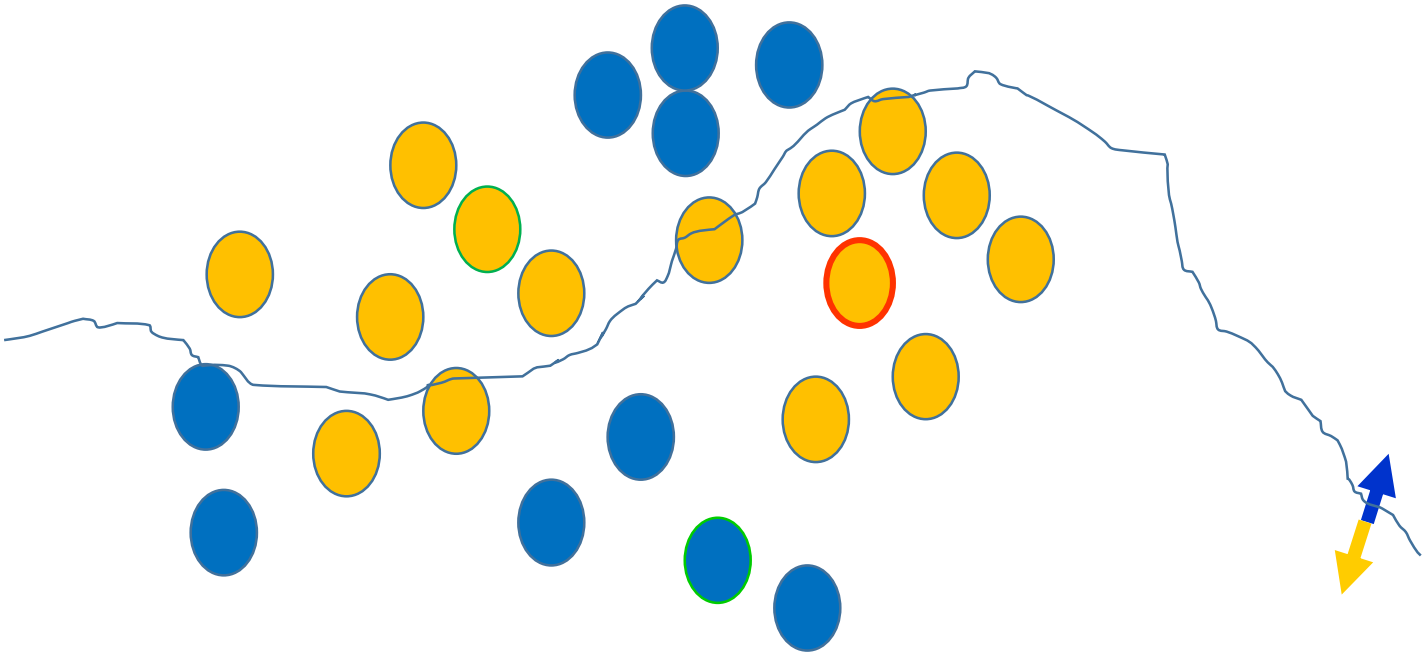
Decision Boundary

Present a training instance / adjust the weights



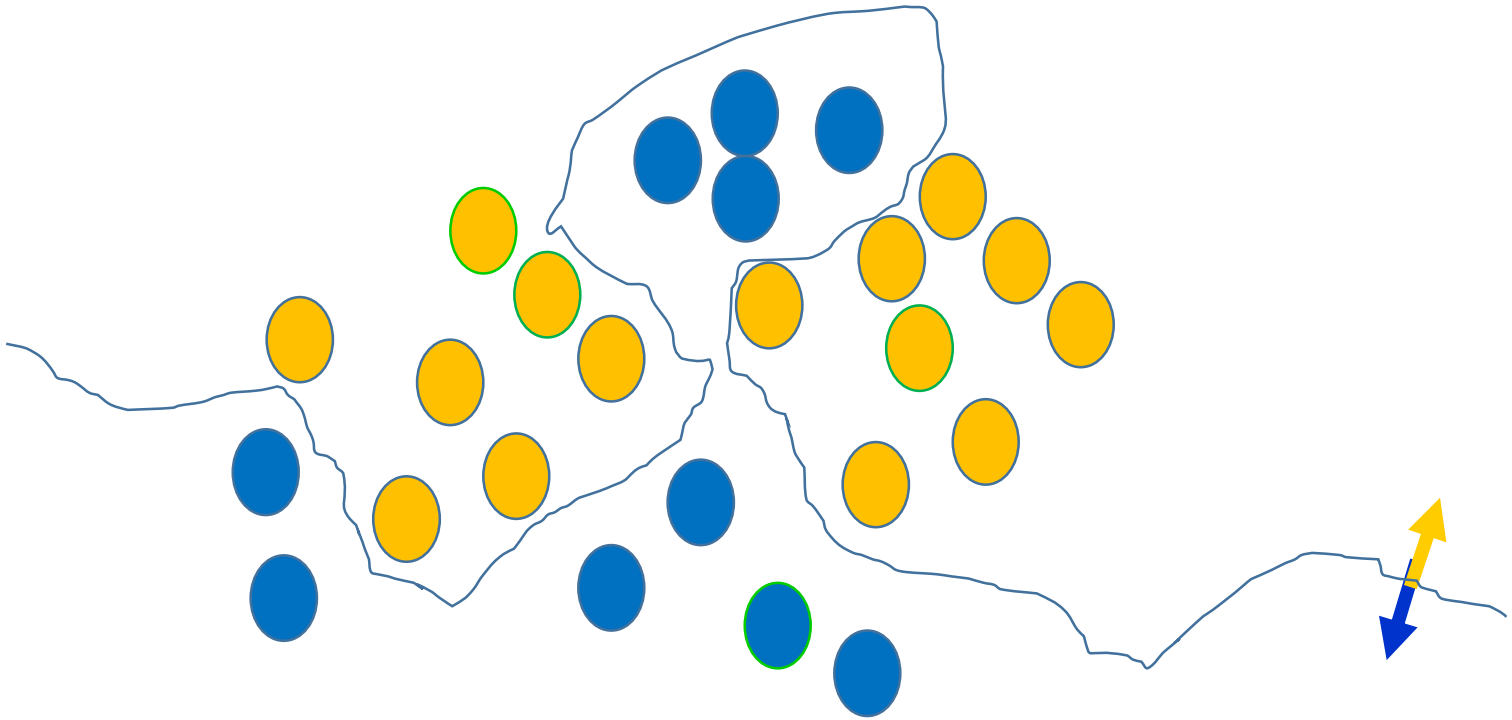
Decision Boundary

Present a training instance / adjust the weights



Decision Boundary

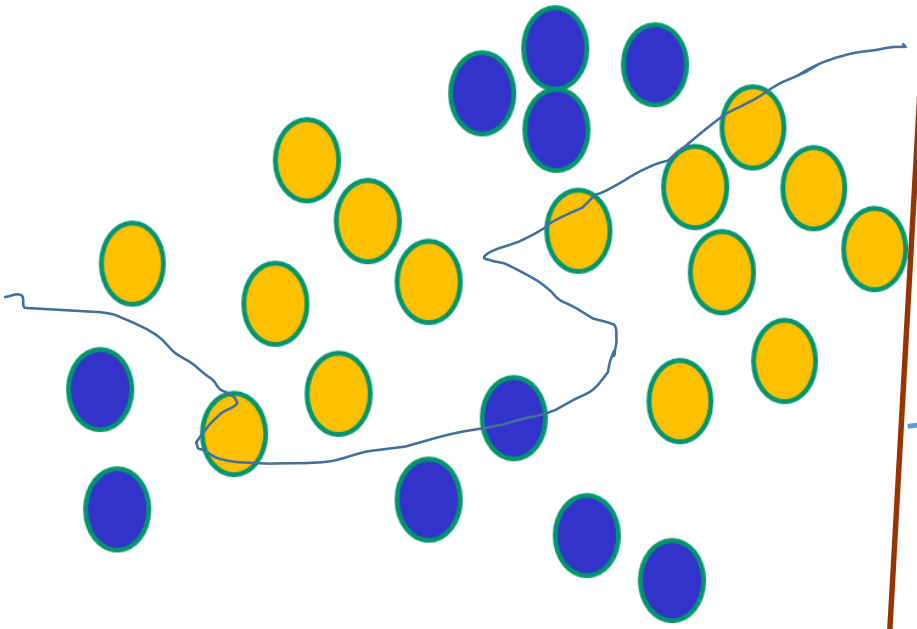
Eventually ….

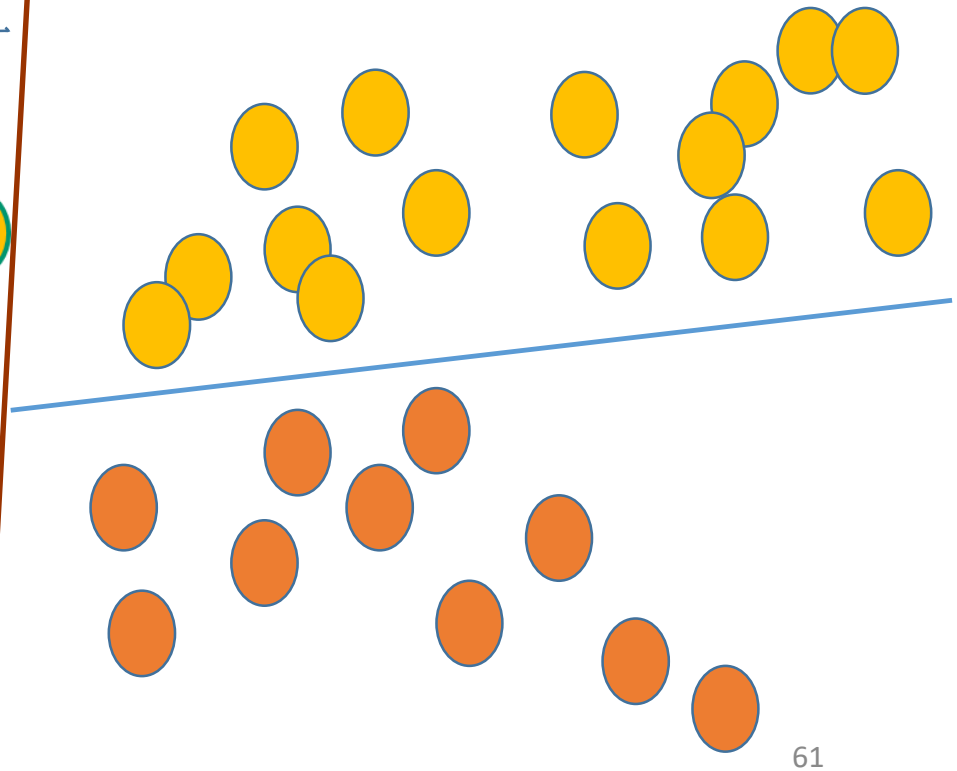


NN vs SVM

NNs use nonlinear *f*(x) so they can draw complex boundaries, but keep the data unchanged

SVMs only draw straight lines, but they transform the data first in a way that makes that OK





So…

- weight-learning algorithms for NNs are “dumb”

- they work by making thousands and thousands of tiny adjustments, each making the network do better at the most recent pattern, but perhaps a little worse on many others

- but, by dumb luck, eventually this tends to be good enough to learn effective classifiers for many real applications

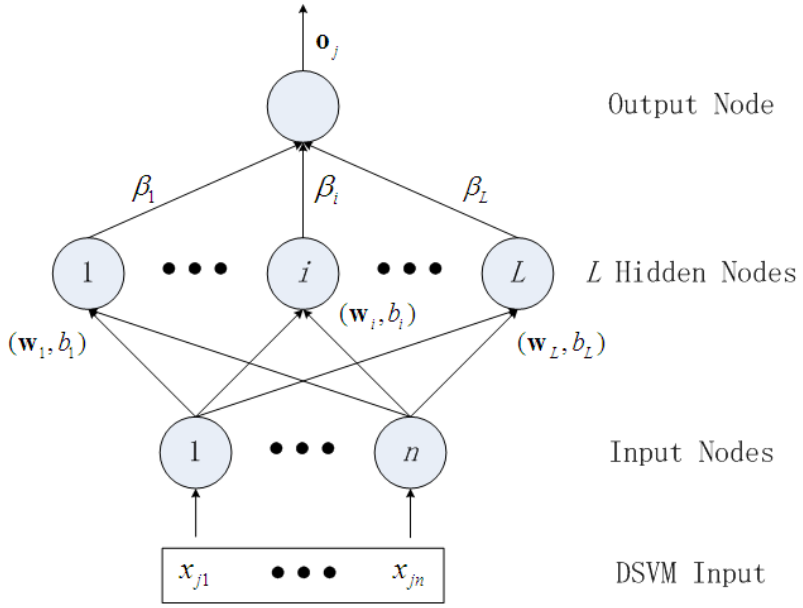
Advanced

Neural Network

Extreme Learning Machine

Deep Learning

Extreme Learning Machine



Steps

- Randomly generate (w_i, b_i)
- Calculate H
- Calculate $\beta = H^{\dagger}T$

Model

Given samples$(x_i, t_i) \in R^{n \times m}$, L is the number of hidden layer nodes

$$\sum_{i=1}^{L} \beta_i \, g(w_i x_j + b_i) = t_j, \; j = 1, \ldots, n$$

where:

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_n) \end{bmatrix} = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \cdots & g(w_L \cdot x_1 + b_L) \\ \vdots & \cdots & \vdots \\ g(w_1 \cdot x_n + b_1) & \cdots & g(w_L \cdot x_n + b_L) \end{bmatrix}_{n \times L}$$

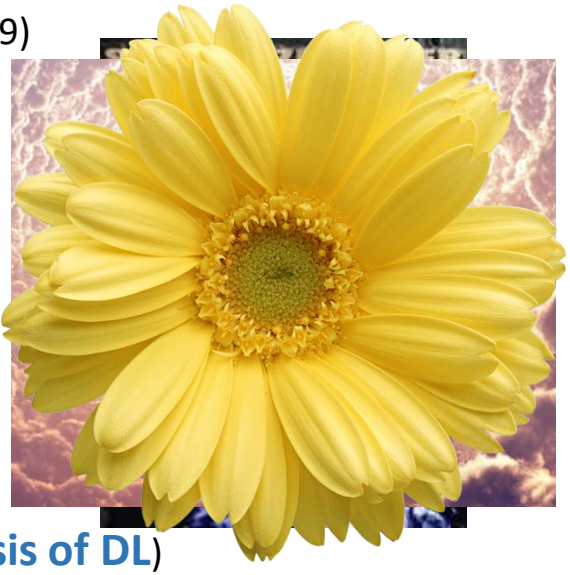
$$T = \left[t_1^T, \ldots, t_L^T \right]_{m \times L}^T \qquad \beta = \left[\beta_1^T, \ldots, \beta_L^T \right]_{m \times L}^T$$

Advanced

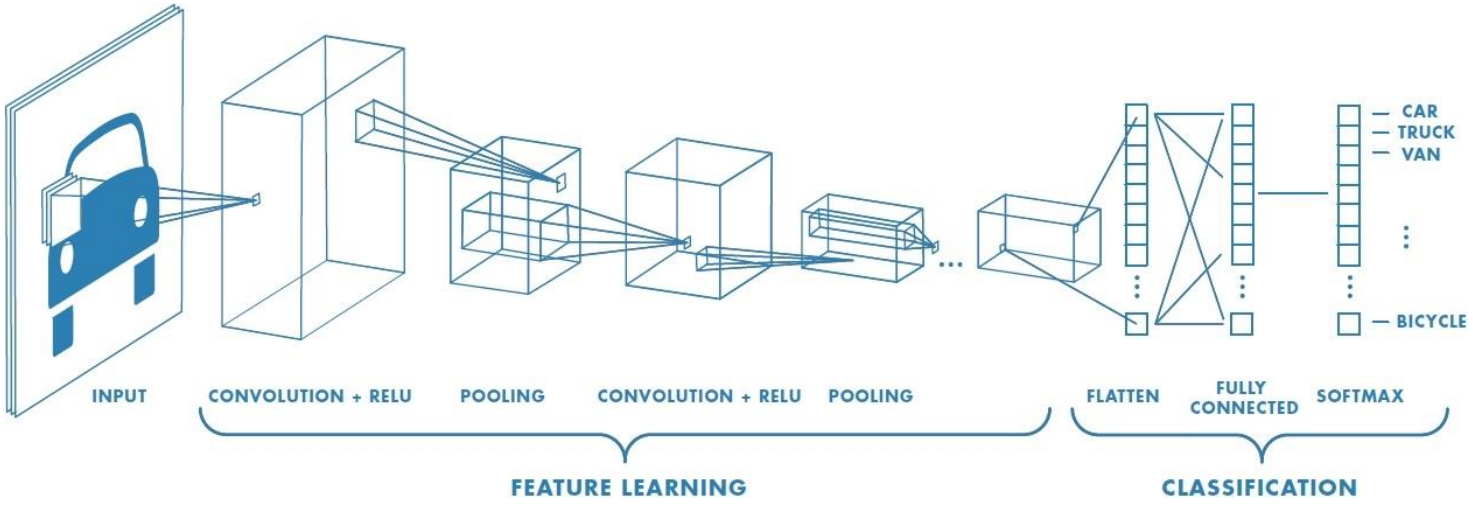
Neural Network
Extreme Learning Machine
Deep Learning

(D)NN Annals

- **1st generation (1958~1969)**
  - MCP artificial neuron model (1943)
  - Perceptron on binary classification problem by Rosenblatt (1958)
  - Proved by Minsky that it can only solve linear classification problem. Even XOR cannot be classified correctly. **The AI Winter**. (1969)
- **2nd generation (1986~1998)**
  - Multi Layer Perceptron (MLP: BP + sigmoid) by Hinton (1986)
  - Universal approximation theorem by Robert Hecht-Nielsen (1989)
  - Convolutional NN (LeNet) by LeCun (1989)
  - Long Short-Term Memory (LSTM) model (1997)
- Meanwhile in statistical learning
  - Decision tree (ID3, ID4, CART, etc.) (1986)
  - Linear SVM by Vapnik (1995)
  - AdaBoost (PAC + ensemble) (1997)
  - Kernel SVM. **Terminator of NN**. (2000)
  - Random forest (2001)
- **3rd generation (2006~now)**
  - Solution to vanishing gradient problem by Hinton (2006, **Genesis of DL**)
  - Activation function ReLU (2011)
  - Champion of ImageNet AlexNet by Hinton outperforms runner-up SVM (2012) (**Blooming of DL**)
  - NIN (2013), GoogLeNet(2014), ResNet(2015), ENet (2016)…



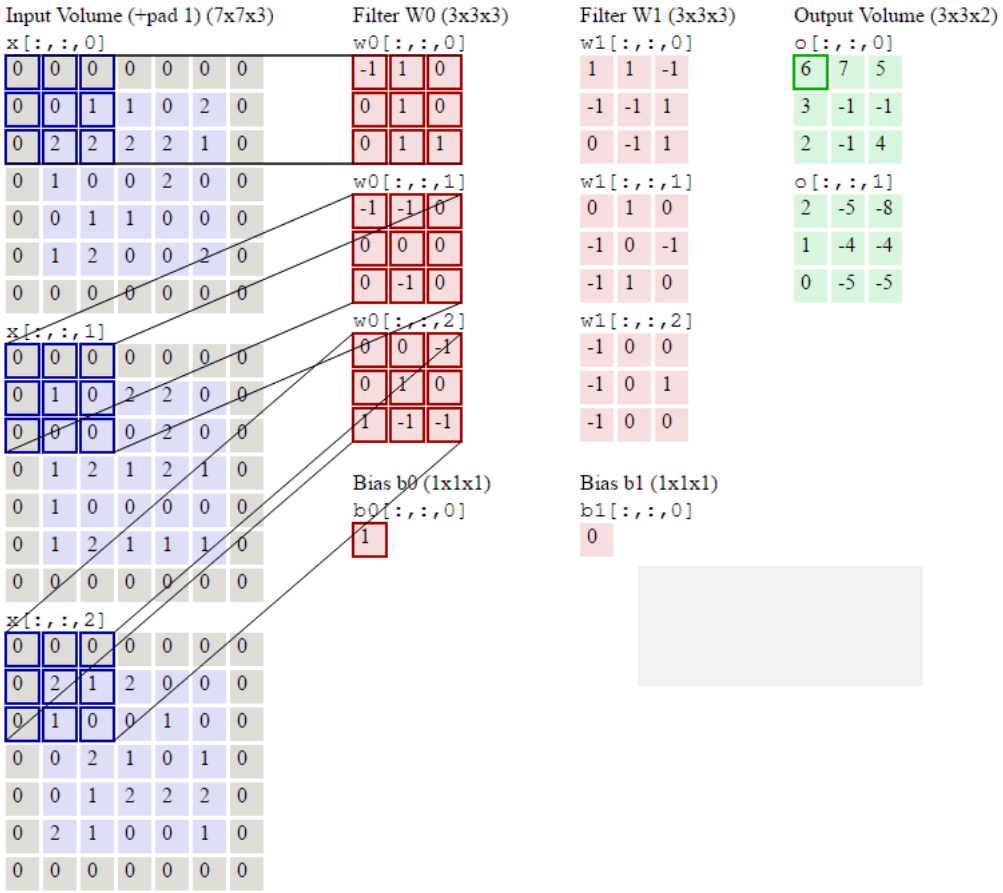
Convolutional Neural Network



- Key concepts:
  - Convolution
  - ReLU
  - Pooling
  - Dropout
  - SoftMax

Convolutional Neural Network

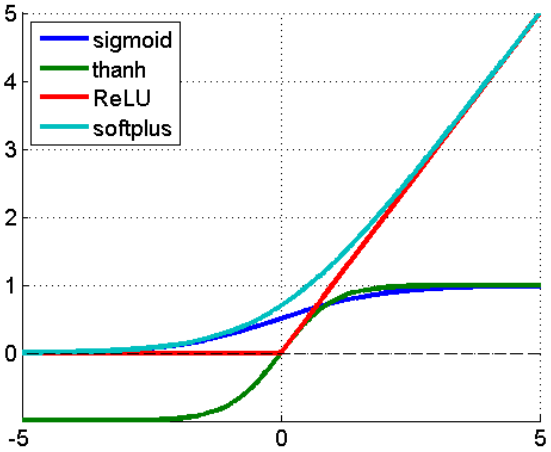
- Convolutional layer



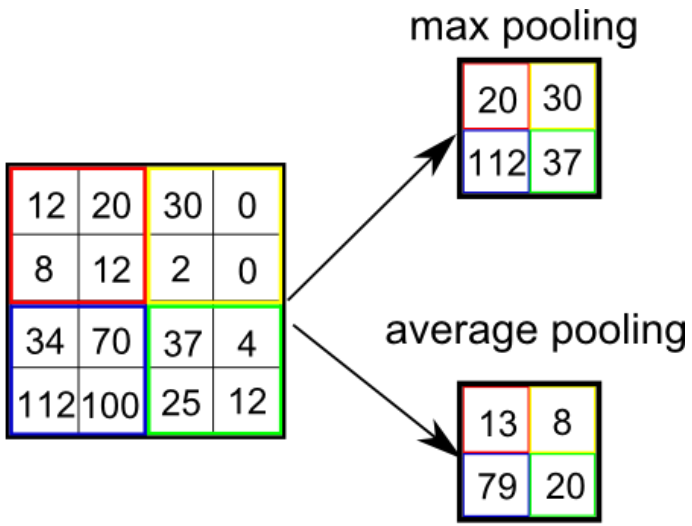
via web, source unknown.

Convolutional Neural Network

- ReLU (Rectified Linear Units)

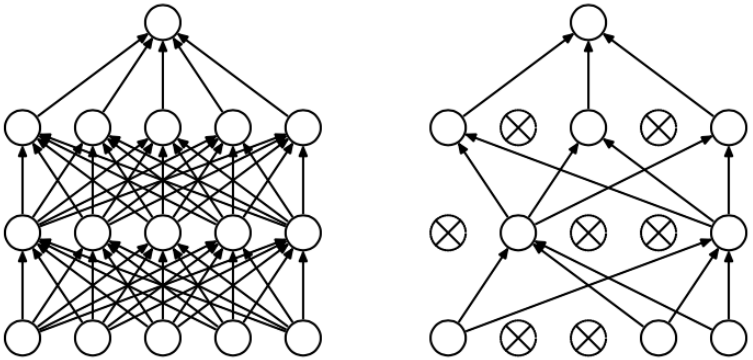


- Pooling

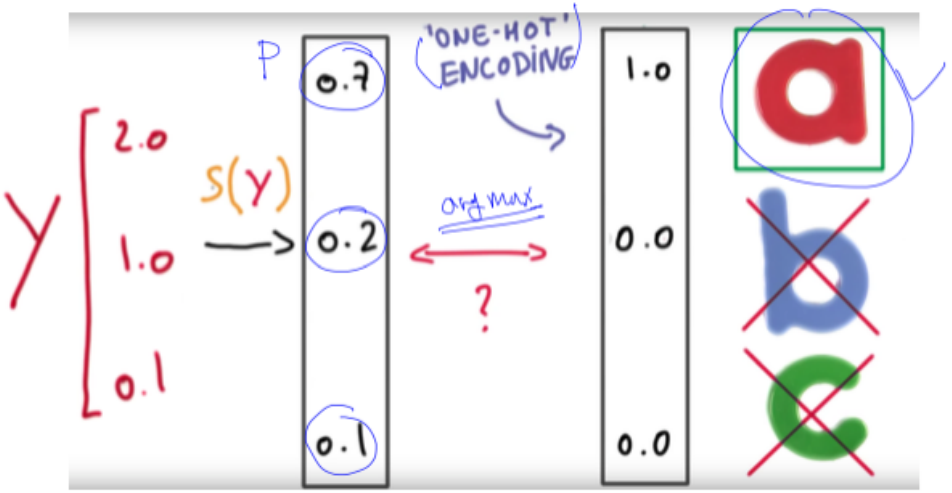


Convolutional Neural Network

- Dropout
  - To avoid overfitting

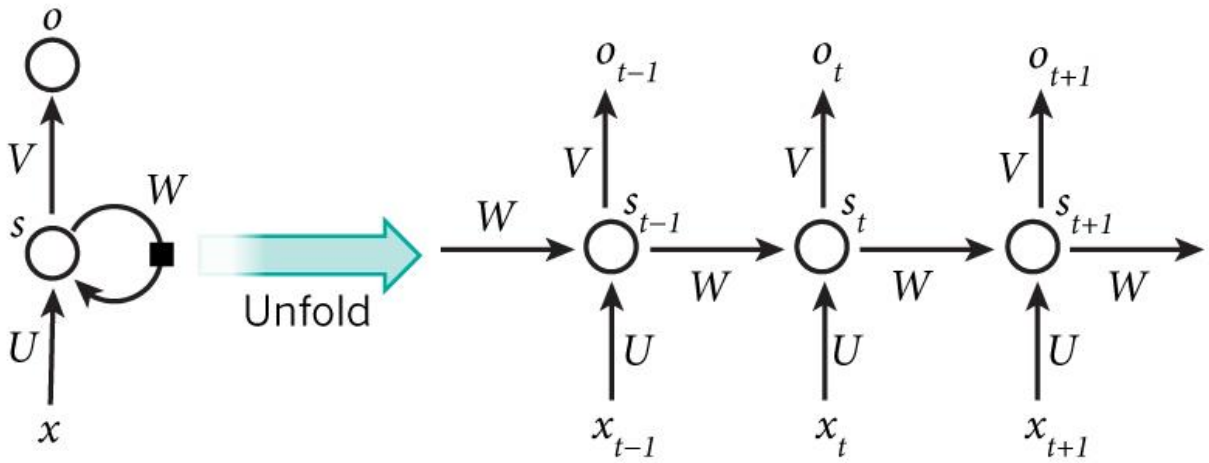


- Softmax



https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

Recurrent Neural Network

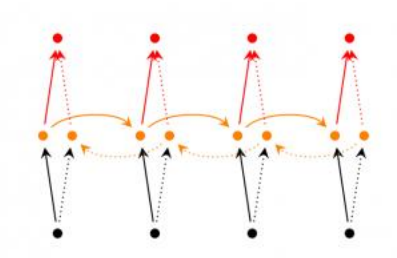


$$s_t = f(Ux_t + Ws_{t-1})$$

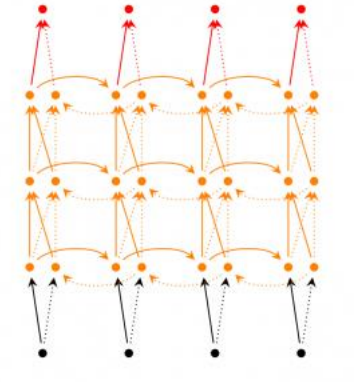
- x_t is the input at time step t.
- s_t is the hidden state at time step t. It's the **memory of the network**. s_t is calculated based on previous hidden state and the current input. It captures information about what happened in **all the previous** time steps.
- Function f usually is nonlinearity such as *tanh* or *ReLU*.
- o_t is the output **solely** based on the memory at step t.
- RNN shares the **same parameters** (U, V, W above) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs.
- The above diagram has outputs at each time step, but depending on the task this may not be necessary. We may only care about the **final output**.
- The main feature of an RNN is its hidden state, which captures some information about a **sequence**.

RNN Extensions

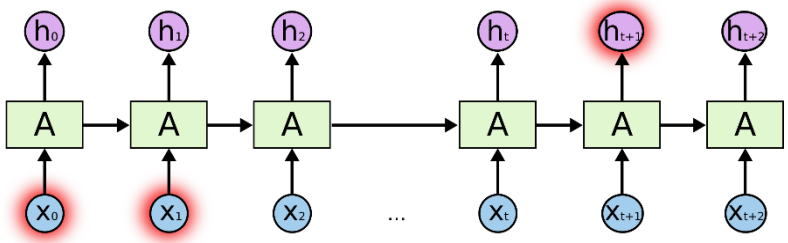
- Bidirectional RNN



- Deep (Bidirectional) RNN

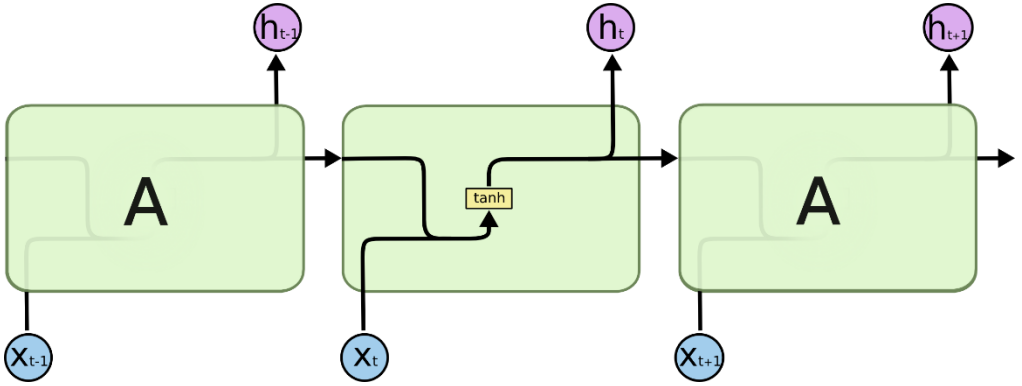


- LSTM

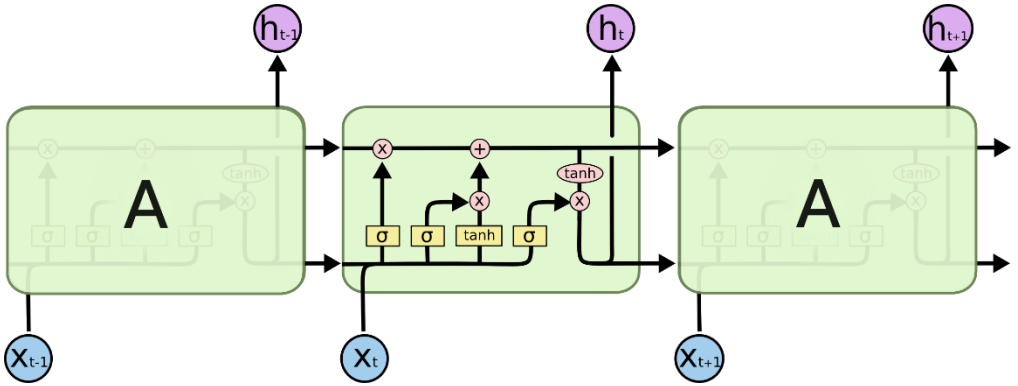


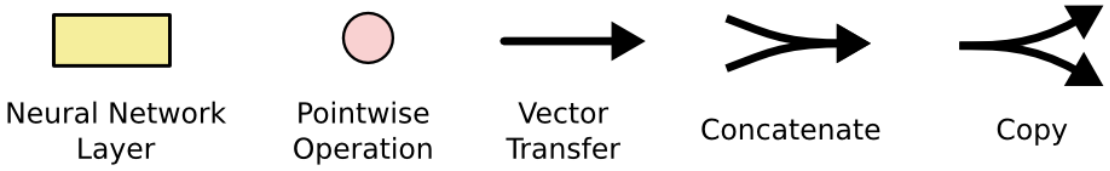
Long Short Term Memory

- Standard RNN

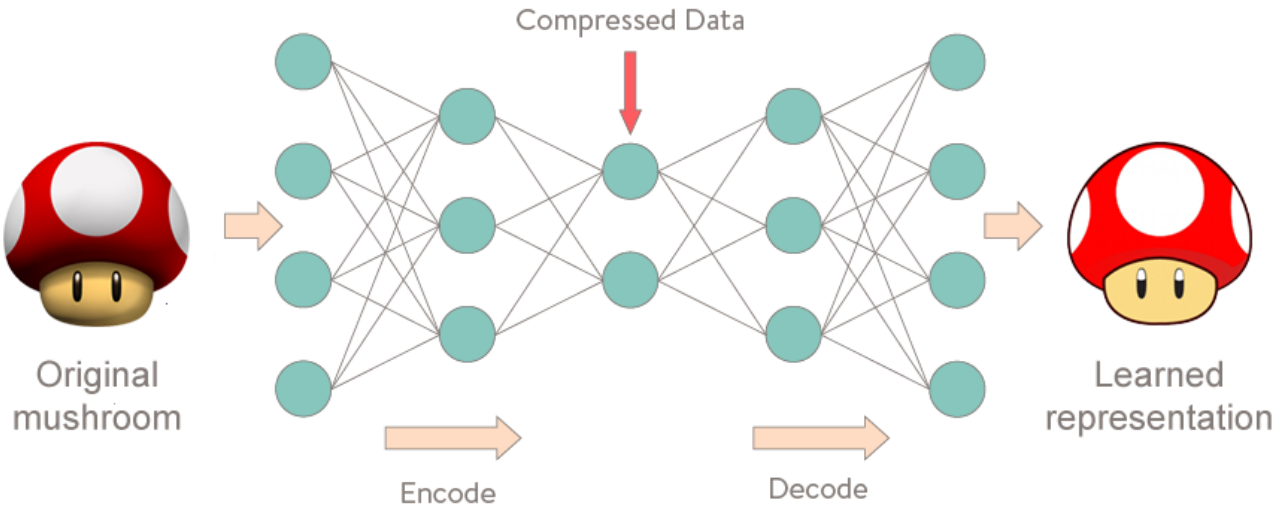


- LSTM



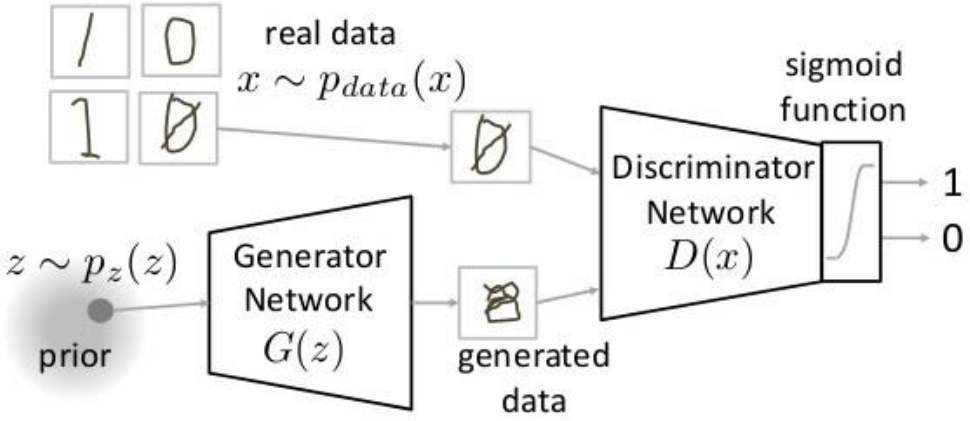


Autoencoder



- Autoencoder is a neural network (NN), as well as an **un-supervised learning** (feature learning) algorithm.
- **Encoders** and **Decoders** are **feed-forward NN**.
- It applies **backpropagation**, by setting the target value same as input.
- It tries to predict ***x*** from ***x***, **without** need for **labels.**
- It finds the **low dimensional representation** of input data.
- It represents original input from **compressed, noisy, or corrupted data.**
- Variations: Convolutional AutoEncoders (CAE), Variational AutoEncoder(VAE), Sparse AutoEncoder, Stacked AutoEnoder (SAE), Deep AutoEncoders (DAE)…

Generative Adversarial Network



- The **Generator Network** takes an random input and tries to generate a sample of data.
- The **Discriminator Network** takes input either from the real data or from the generator and try to predict whether the input is real or generated.

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))]$$

- The discriminator is trying to **maximize** our function V.
- The generator is trying to **minimize** the function V.

Generative Adversarial Network

- Steps to train a GAN
  1. **Define the problem**. Here you should completely define the problem and collect data for it.
  2. **Define architecture of GAN**. Define how your GAN should look like. Should both your generator and discriminator be multi layer perceptrons, or convolutional neural networks?
  3. **Train Discriminator on real data for *n* epochs**. Get the data you want to fake on and train the discriminator to correctly predict them as real.
  4. **Generate fake inputs for generator and train discriminator on fake data**. Get generated data and let the discriminator correctly predict them as fake.
  5. **Train generator with the output of discriminator**. Get the predictions of discriminator and use it as an objective for training the generator. Train the generator to fool the discriminator.
  6. **Repeat step 3 to step 5 for a few epochs**.
  7. **Check if the fake data manually if it seems legit**. This is a bit of a manual task, as hand evaluating the data is the best way to check the fakeness.

Some Frameworks

- **Theano**
  - Genesis (2007)
  - Based on Python +NumPy
  - From and for academia
  - Developers including Yoshua Bengio and Ian Goodfellow
- **Caffe**
  - Convolution Architecture For Feature Extraction
  - Developed by Yangqing Jia during his PhD at UC Verkeley (2013)
- **TensorFlow**
  - Google
  - Fast compiling
  - Based on Python / C++
- **Keras**
  - Easy and clean
  - Based on Python
  - Supports interface to TensorFlow and Theano
- **SciKit-learn**
  - Modularized implementation
  - Easy to get started
  - Based on Python
- **MXNet**
  - Amazon
- **Paddle**
  - Baidu
- **Torch**
  - Smooth in CNN development
  - Supports various RNNs
  - Based on Lua

Big Data Classification

Ensemble
Distributed Learning

What is Ensemble?

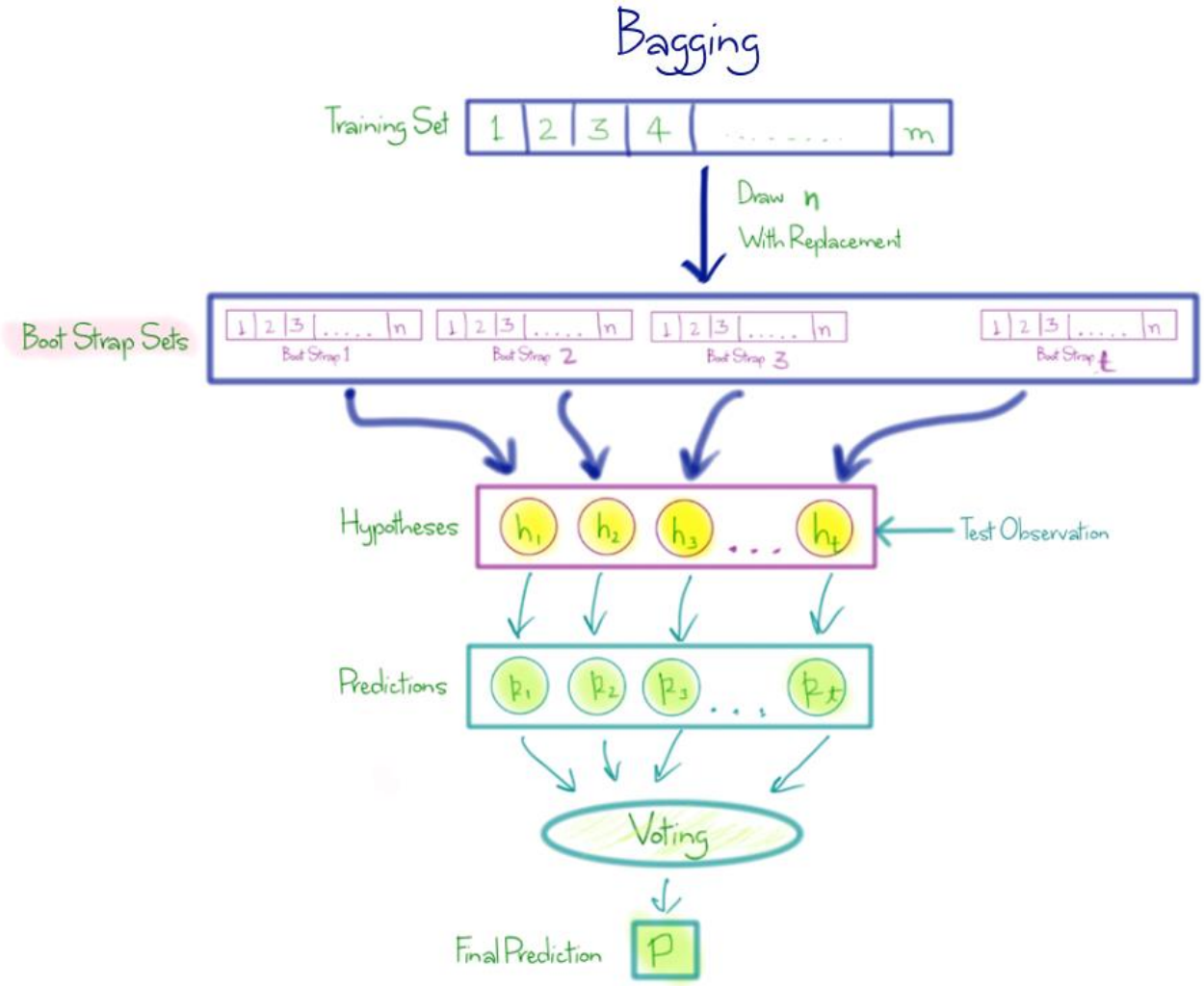
- In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. A machine learning ensemble consists of only a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives.

- Methods
  - Bootstrap aggregating
  - Boosting
  - Stacking

Bootstrap Aggregating

- Often abbreviated as **bagging**
- In order to promote model variance, bagging trains each model using a randomly drawn subset of the training set
- Sampling with replacement
- Each model in the ensemble vote with equal weight
- Tends to cancel out the random variances of base learners
- create an "averaged out" learner
- Random forest

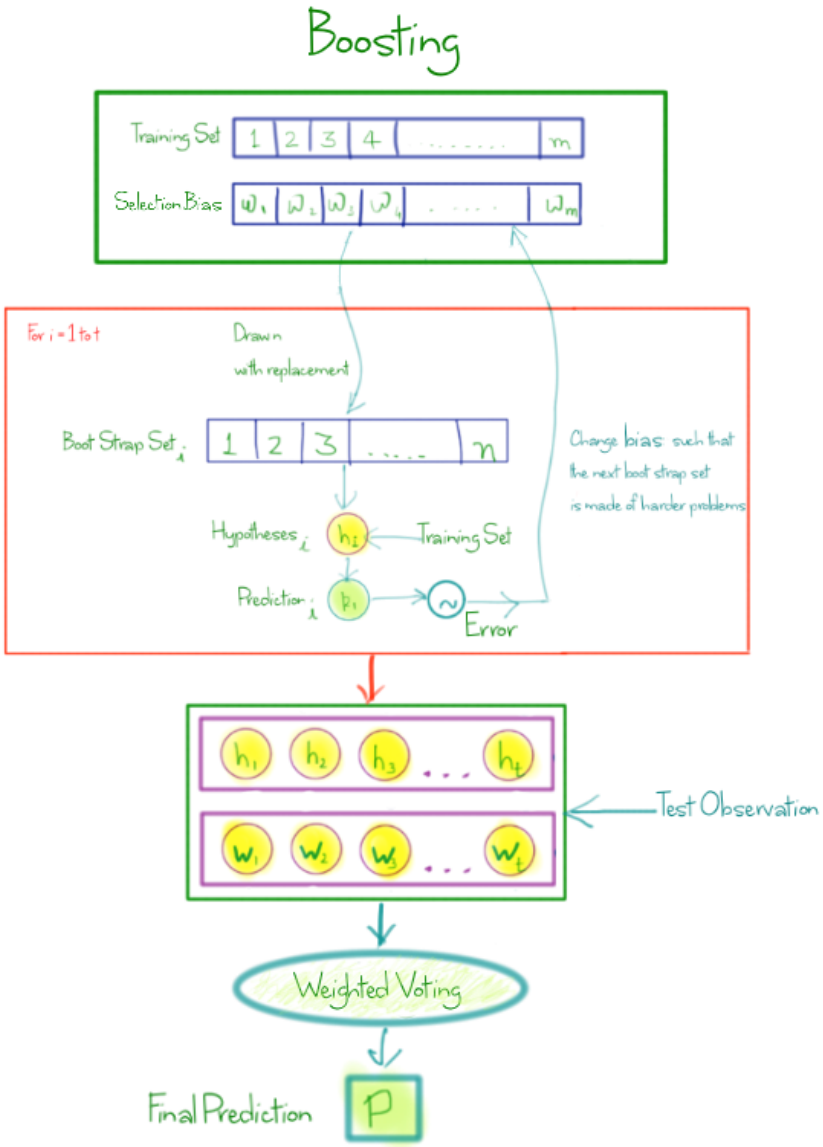
Bootstrap Aggregating



Boosting

- The idea is that a "**weak**" learning algorithm, can be "boosted" into a "**strong**" learning algorithm in a set of iterative trainings
- Increase the selection bias for each wrongly classified observation
- The final classifier takes a weighted vote on the predictions of base classifiers
- Many different versions, the main difference is their method of weighting training data points and hypotheses
- AdaBoost

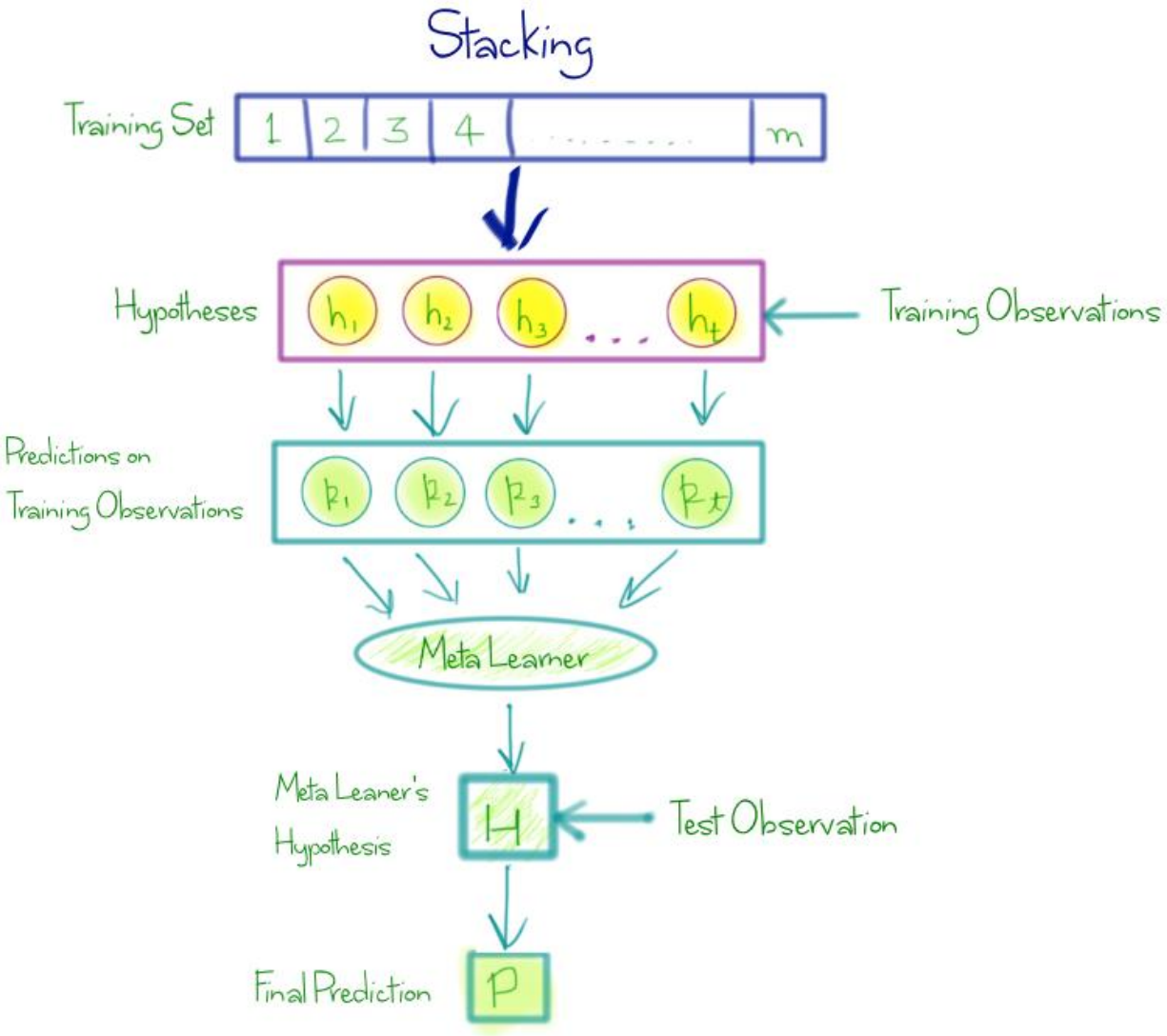
Boosting


Boosting
Training Set
1
2
3
4
m
Selection Bias
For i = 1 to t
Drawn
with replacement
Boot Strap Set i
1
2
3
n
Change bias such that
the next boot strap set
is made of harder problems
Hypotheses i
Training Set
Prediction i
Error
Test Observation
Weighted Voting
Final Prediction
P

Stacking

- Sees patterns of agreements, and disagreements between base learners
- Uses a meta-learner to discover these patterns
- The meta-learner looks for **patterns** like
  - "when learner A, and learner B are in unison, the predictions are always correct"
  - or "when A, B, C disagree, then C is almost always correct", etc.

Stacking


Stacking
Training Set
1
2
3
4
m
Hypotheses
h_1
h_2
h_3
h_t
Training Observations
Predictions on
Training Observations
p_1
p_2
p_3
p_t
Meta Learner
Meta Leaner's
Hypothesis
H
Test Observation
Final Prediction
P

Big Data Classification

Ensemble
Distributed Learning

Qing He, Tianfeng Shang, Fuzhen Zhuang, Zhongzhi Shi. Parallel extreme learning machine for regression based on MapReduce. Neurocomputing 102(2013):52-58.

Distributed ELM

- Original ELM

$$\beta = \left(\mathbf{I}/\lambda + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T}$$

- Distributed ELM

$$\mathbf{X}[i][j] = \sum_{k=1}^{N} H[k][i] \times H[k][j]$$

$$\mathbf{Y}[i][j] = \sum_{k=1}^{N} H[k][i] \times T[k][j]$$

Distributed ELM

Mapper

Input: ⟨*key*,*value*⟩ pair, where *key* is the offset in bytes, and *value* is a string containing the hidden node output vector of each sample and the original sample.

Output: ⟨*key*′,*value*′⟩ pair, where *key*′ is a random string, and *value*′ is a string of the intermediate results.

```
(1)  Parse the string value to an array, named onesample;
(2)  Normalize the dependent variables of each input sample
     data;
(3)  Initiate string outvalue as a null string;
(4)  Initiate string depVarMatrix as a null string;
(5)  for i=1 to onesample.length
(6)   for j=1 to onesample.length
(7)     outvalue.append(onesample[i] × onesample[j]);
(8)     outvalue.append(",");
(9)   endfor
(10)  for j=1 to depVariables.length
(11)    depVarMatrix.append(onesample[i] × depVariables[j]);
(12)    depVarMatrix.append(",");
(13)  endfor
(14) endfor
(15) outvalue.append(depVarMatrix);
(16) output("HTH",outvalue);
```

Reducer

Input: ⟨*key*,*value*⟩ pair, where *key* is a random string, and *value* is a string of the intermediate results.

Output: ⟨*key*′,*value*′⟩ pair, where *key*′ is a string containing all the elements in matrix $H^T \times H$, and *value*′ is any string you like.

```
(1)  Initiate matrix sumMatrix as a L × L dimension matrix
     with all elements are zero; (L is hidden node number)
(2)  Initiate matrix depVarMatrix as a L × M dimension matrix
     with all elements are zero; (M is dependent variable
     number)
(3)  while(value.hasNext())
(4)   oneValue=value.next();
(5)   Parse the string oneValue to an array, named onesample;
(6)   for i=1 to L
(7)     for j=1 to L
(8)       sumMatrix[i][j]+=onesample[i × L+j];
(9)     endfor
(10)    for j=1 to M
(11)      depVarMatrix[i][j]+=onesample[L × L+i × M+j];
(12)    endfor
(13)  endfor
(14) endwhile
(15) Initiate string outkey as a null string;
(16) for i=1 to L
(17)  for j=1 to L
(18)    outkey.append(sumMatrix[i][j]);
(19)    outkey.append(",");
(20)  endfor
(21) endfor
(22) for i=1 to L
(23)  for j=1 to M
(24)    outkey.append(depVarMatrix[i][j]);
(25)    outkey.append(",");
(26)  endfor
(27) endfor
(28) output(outkey,"HTH");
```

End of Chapter 12