# 資料分析與學習基石
# (*Fundamental of Data Analytics and Learning*)

## -- Advanced Classifier

*Hung-Yu Kao* (高宏宇)
***I**ntelligent **K**nowledge **M**anagement Lab*

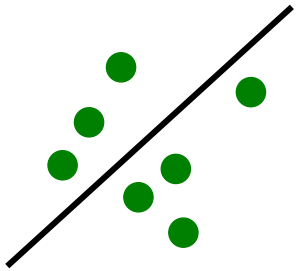
*Master Program of Artificial Intelligence*
*Institute of Medical Informatics,*
*Dept. of Computer Science and Information Engineering*
*National Cheng Kung University, Tainan, Taiwan*

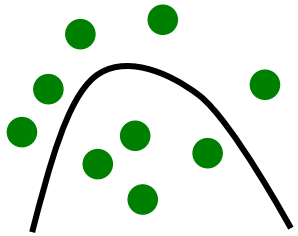
# Classification Techniques

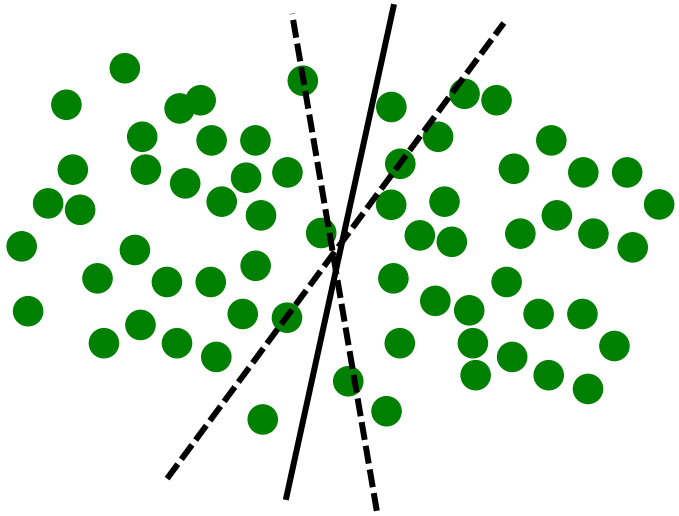
- Decision Tree based Methods
- Memory based reasoning (K-NN)
- Naïve Bayes
- Support Vector Machines

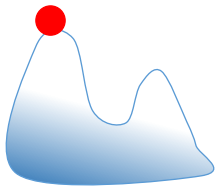
- Regression
- Neural Networks
  - DNN
- Ensemble
  - Boosting
  - Random Forest, XGBoost (Regression Tree, Regression Tree Ensemble)
- Semi-supervised Learning

# Regression









Gradient Descent

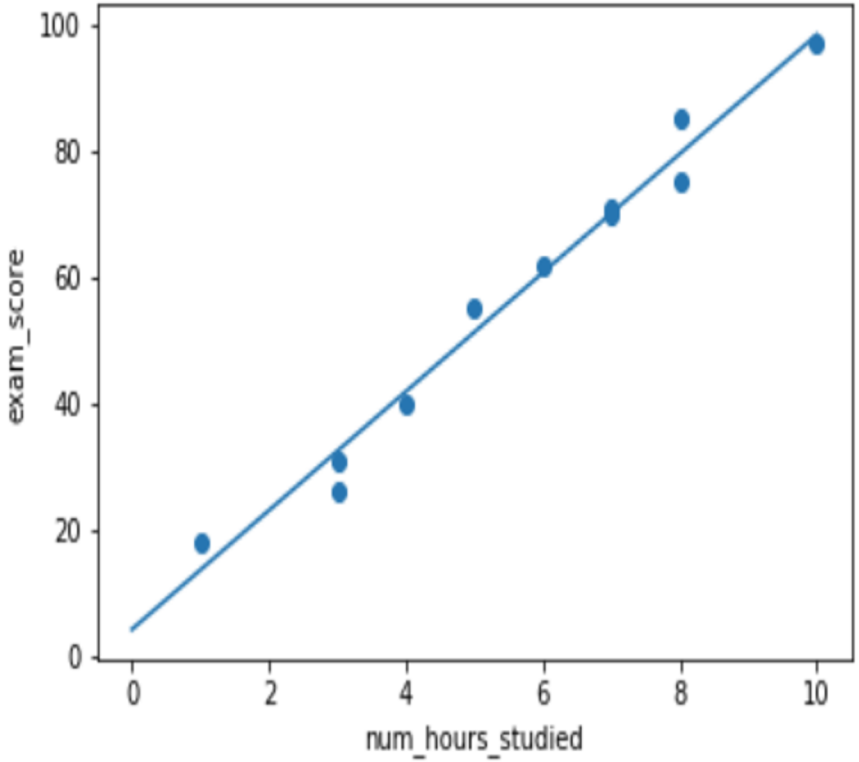
# Linear Regression

Finds a **line of best fit** through the data

We "regress" features ***x*** onto a "response", ***y***

E.g. regress number of hours studies onto exam score

Works well when the true underlying function is linear



# General Machine Learning Problem

We have some real-world data-generating function $f$ we want to approximate

We select a hypothesis $h_\theta$ that we think will suit the real function $f$ and use data to train the parameters $\theta$ to fit this function

Space of all possible functions (huge - intractable?)

A subspace we hypothesize the function $f$ lies in

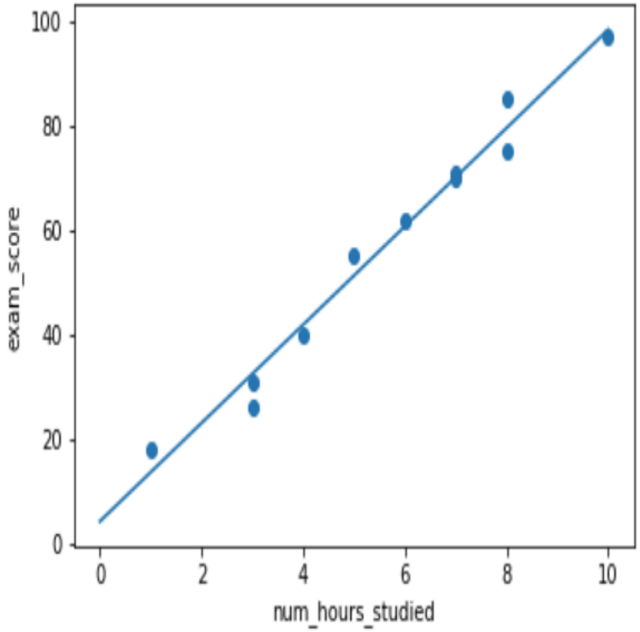
# Linear Regression

Our "hypothesis" about the true underlying function: it is linear

We express this mathematically in the well known function for a line

$$h_\theta(\mathbf{x}) = \mathbf{a}^\top \mathbf{x}$$

Where ***a*** are the parameters ($\theta$) that we have to learn and ***x*** is our feature vector



# Design Matrix

In practice we usually have *many* features. We organize them into a **num_data_points** x **num_features** "design matrix" ***X***, and add the bias (intercept) as a column of ones on the left.

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 8 \\ 1 & 5 & 6 \\ 1 & 7 & 6 \\ 1 & 8 & 4 \end{bmatrix}$$

For example, if we added **num_hours_slept_last_night** to **num_hours_studied** we could have the design matrix with 4 data points on the right

$$h_\theta\left(\mathbf{X}^{(i)}\right) = \mathbf{a}^\top \mathbf{X}^{(i)}$$

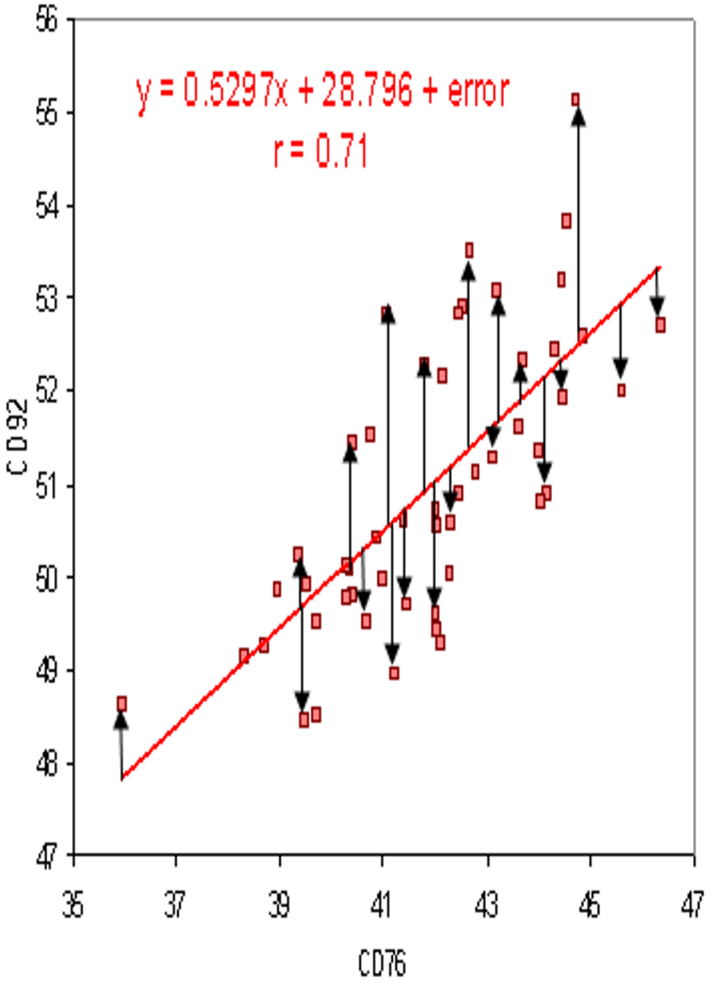
# What is a "Good Fit" ?

We said our goal is to finc

What is a "good fit" ? Ty
"mean squared error"

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - h_\theta^{(i)}\right)^2$$

We use this as our "**loss**"
measure of how badly we
generally want to minimiz



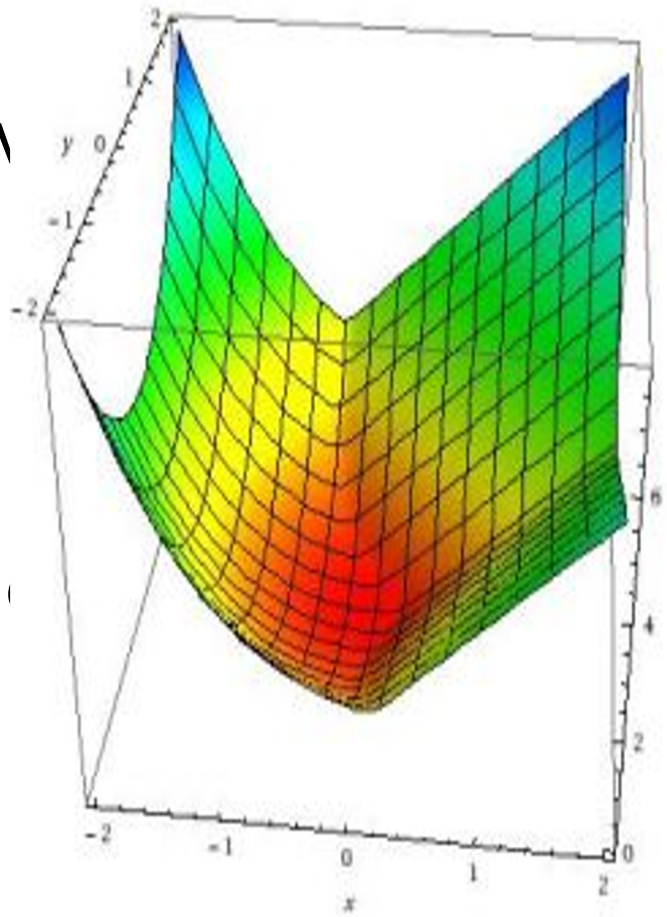
# How to Learn the Parameters?

With our loss function we hav
optimization problem

$$\min_{\mathbf{a}} \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \mathbf{a}^\top \mathbf{X}^{(i)}\right)^2$$

This can be solved with the n

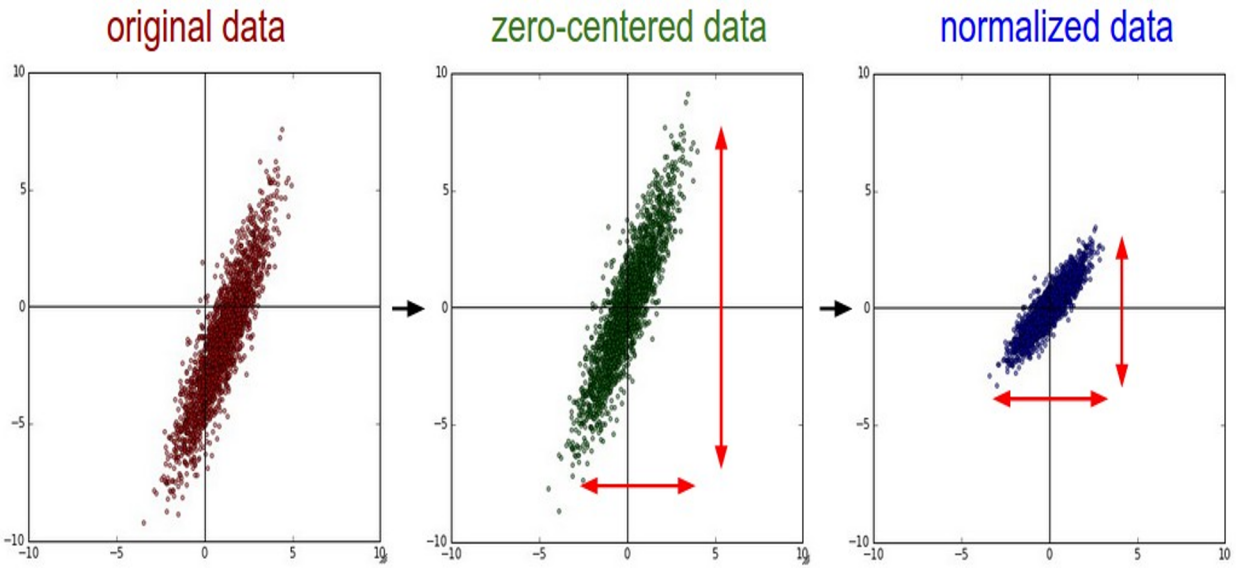
$$\mathbf{a}^* = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$



# Normalization

Different features may use different scales (think of "age" versus "salary")

Normalization can help learning algorithms be more effective and is required for certain techniques (e.g. ridge regression, which we will see later)



# Regression Practice

- https://github.com/IKMLab/Linear-Regression-Tutorial

# Classification

The response variable ***y*** takes a discrete value representing class membership

E.g. we might want to regress **num_hrs_studied** onto **pass/fail** an exam

$$y \in \{0, 1\} \text{ where } 1 = \text{pass and } 0 = \text{fail}$$

We will also aim to generate a probability of class membership

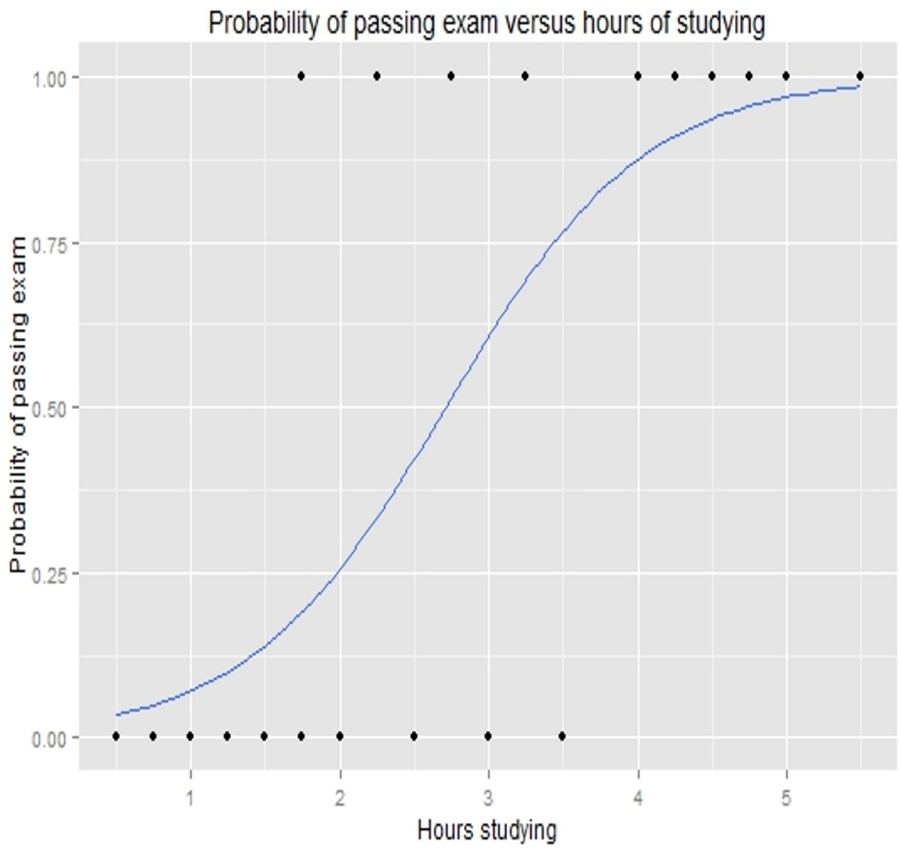



Image taken from https://en.wikipedia.org/wiki/Logistic_regression

# Decision Boundary

We need to make a "decision" about each observation as to which class it belongs to

With two features we might find a classification situation like this

Here our decision boundary is a line - but it could be non-linear, too

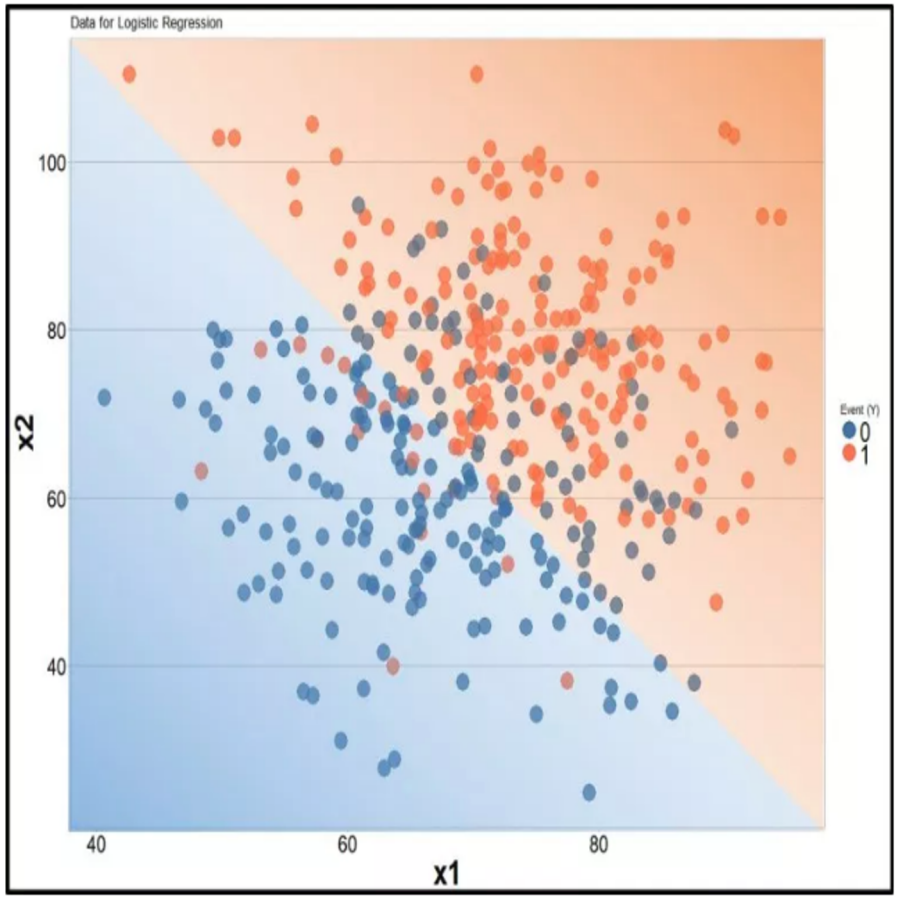


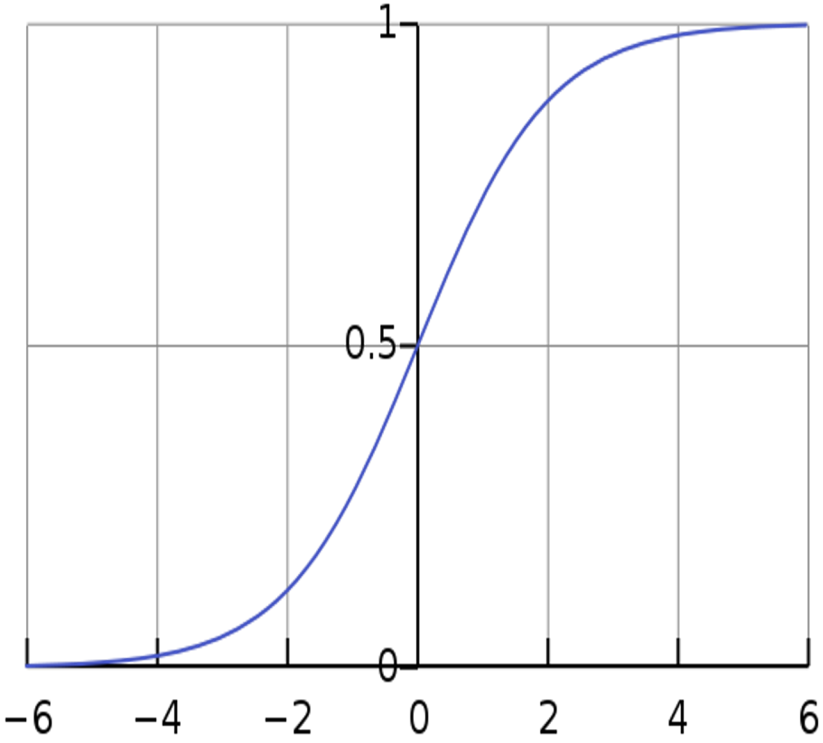

Image taken from "Elements of Statistical Learning" by Hastie and Tibshirani

# Logistic Function

Interpreted as taking log-odds as input and giving probability as output

It is easy to see it squashes the input values to be within *[0, 1]*

$$\sigma(h) = \frac{e^h}{e^h + 1} = \frac{1}{1 + e^{-h}}$$



# Hypothesis

Whereas in linear regression we had (assuming the bias is added to $\boldsymbol{x}$)

$$h_\theta(\mathbf{x}) = \theta^\top \mathbf{x}$$

With logistic regression we now have

$$h_\theta(\mathbf{x}) = \sigma(\theta^\top \mathbf{x}) = P(y = 1|\mathbf{x})$$

Which can be interpreted as the probability of being classified as class 1. The probability of class 2 is then given by

$$P(y = 2|\mathbf{x}) = 1 - P(y = 1|\mathbf{x}) = 1 - \sigma(\theta^\top \mathbf{x})$$

# Optimization

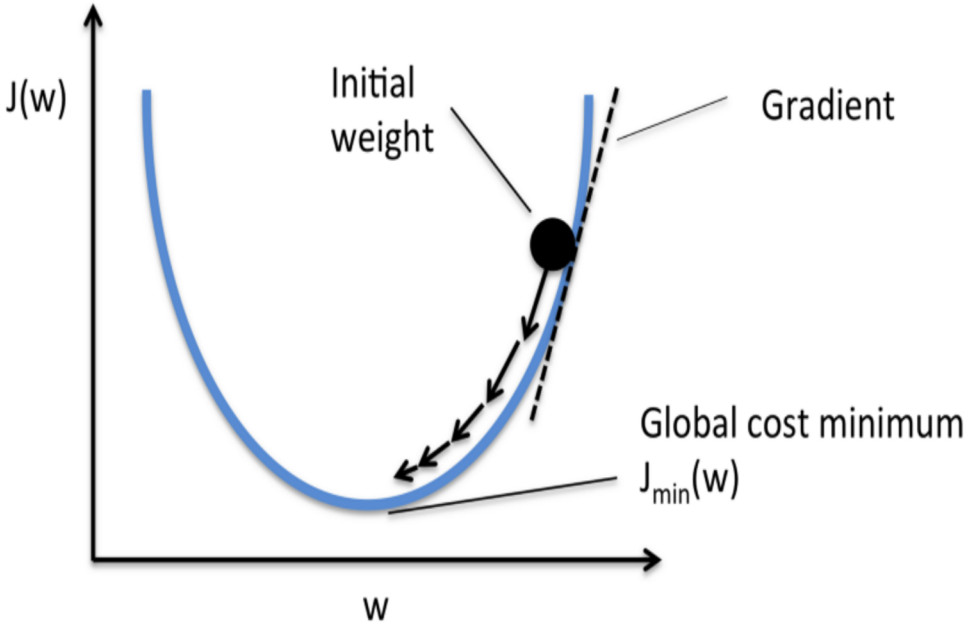
There is no closed-form solution for logistic regression like there is for linear regression

Instead we need to use iterative methods:

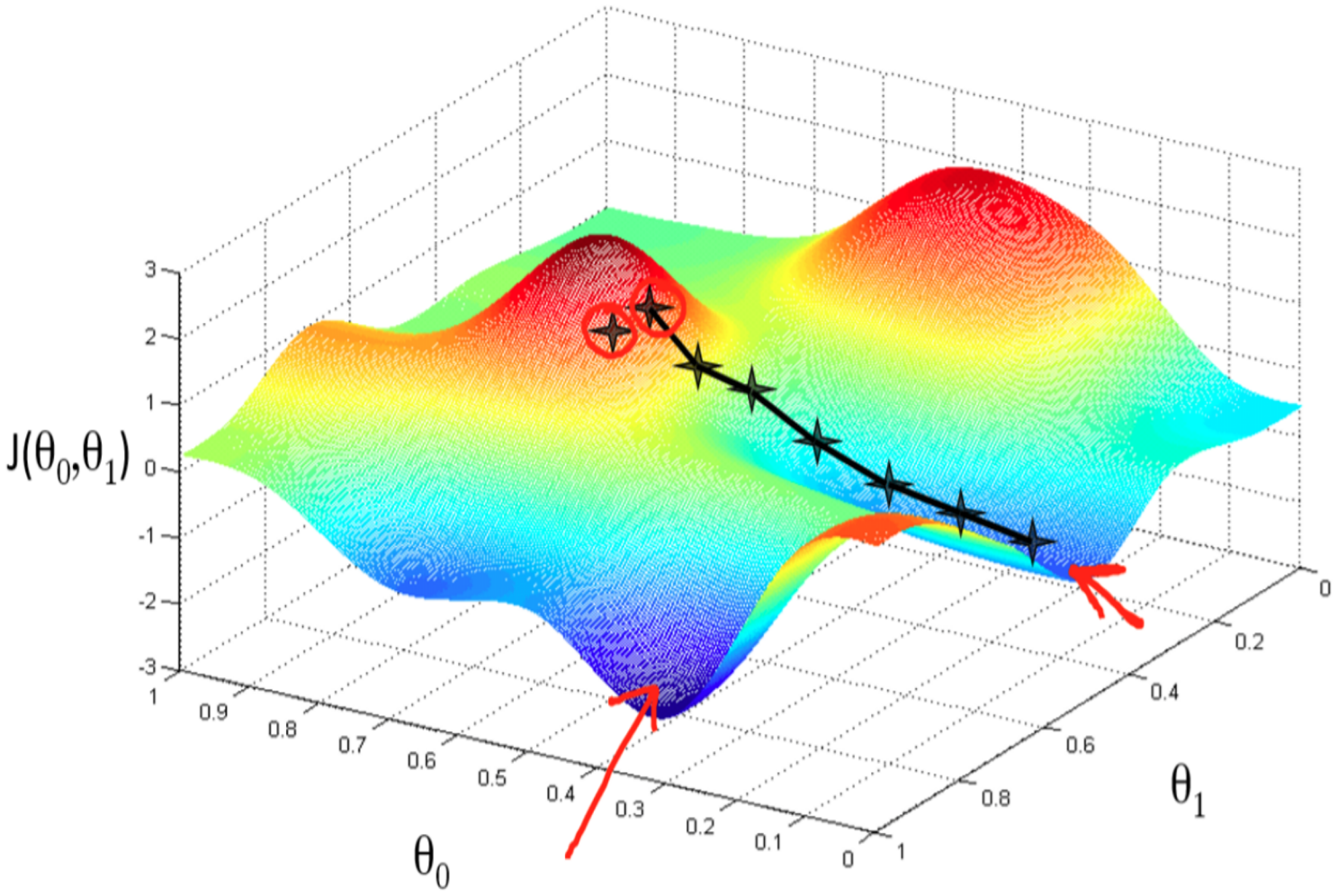
- start off with a tentative solution
- try to improve it iteratively
- Stop when no more improvements are possible

# Stochastic Gradient Descent

If we're at the black dot the gradient is going to tell us how to change $w$ in order to reduce the cost - move towards the left, make $w$ smaller



# Stochastic Gradient Descent



# Hyperparameters

Gradient descent introduces some important hyperparameters you will need to tune

- Learning rate: how big the steps are
  - Too small and fail to get over bumps in the loss surface
  - Too large and bounce around local minima, failing to settle
- Number of iterations: how many steps should we take before stopping?
  - Too few and we fail to get a good solution
  - Too many and we might overfit to the training data

# Grid Search

How to select the best set of hyperparameters?

One option is to define a reasonable range of possible values for each and try out each combination (with cross-validation on the training set)

For example

- step size: [0.01, 0.1, 1.]
- max iterations: [1,000, 10,000]

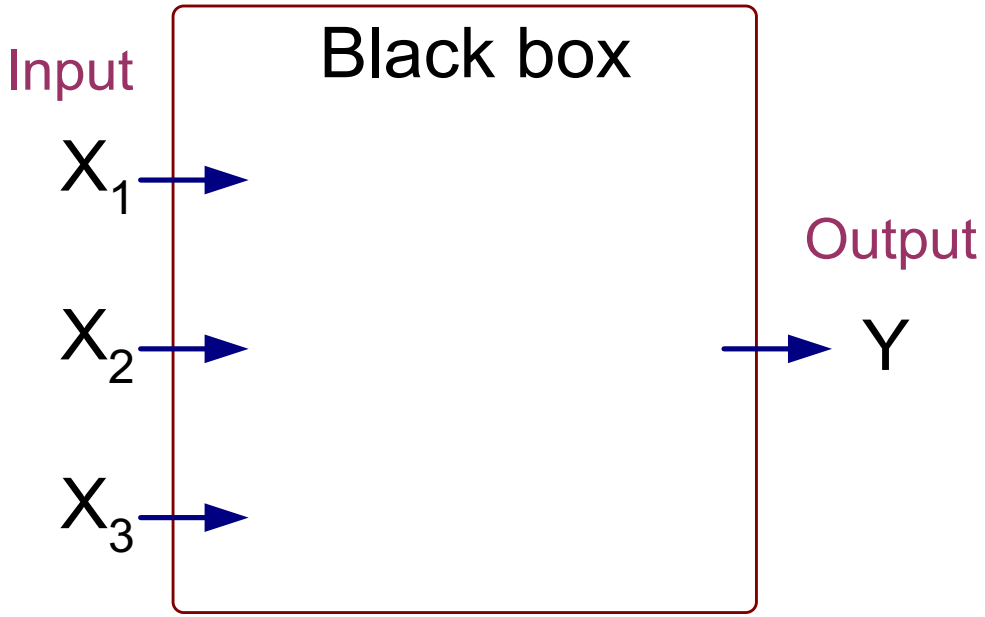
This search space yields *3 x 2 = 6* combinations. We try them all and pick the best performing combination of parameter values.

# Logistic Regression Practice

- https://github.com/IKMLab/Logistic-Regression-Tutorial

# Artificial Neural Networks (ANN)

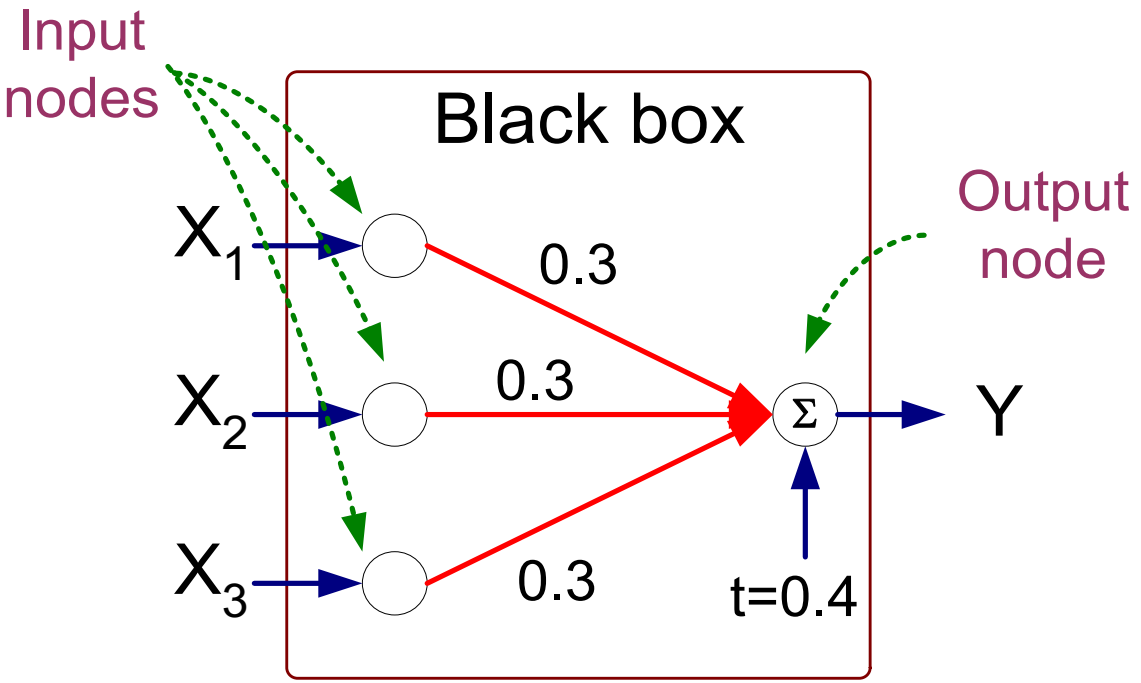
| $X_1$ | $X_2$ | $X_3$ | Y |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 |




Output Y is 1 if at least two of the three inputs are equal to 1.

# Artificial Neural Networks (ANN)

| $X_1$ | $X_2$ | $X_3$ | Y |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 |

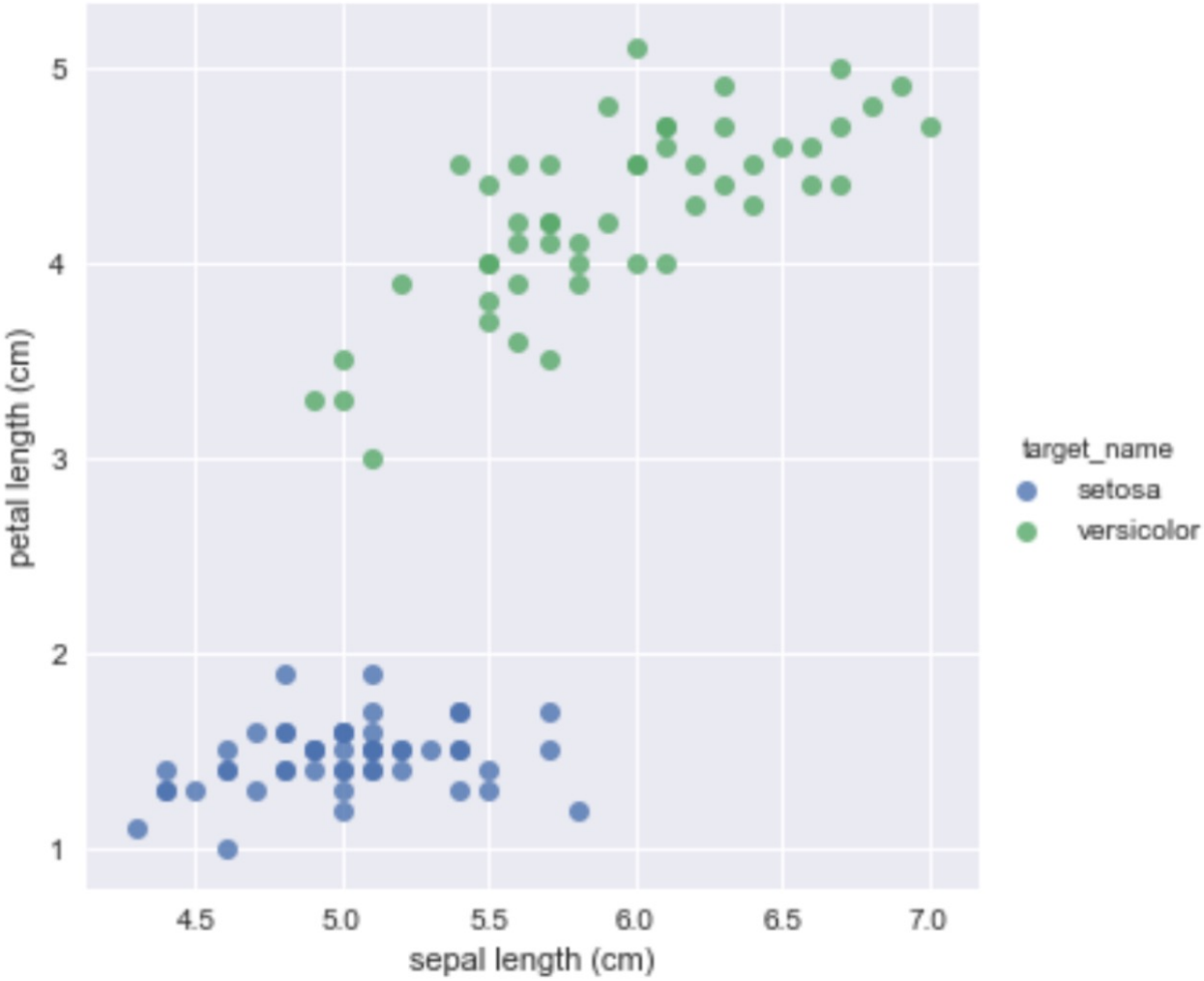



Perceptron（Perceptron Learning Algorithm, PLA）

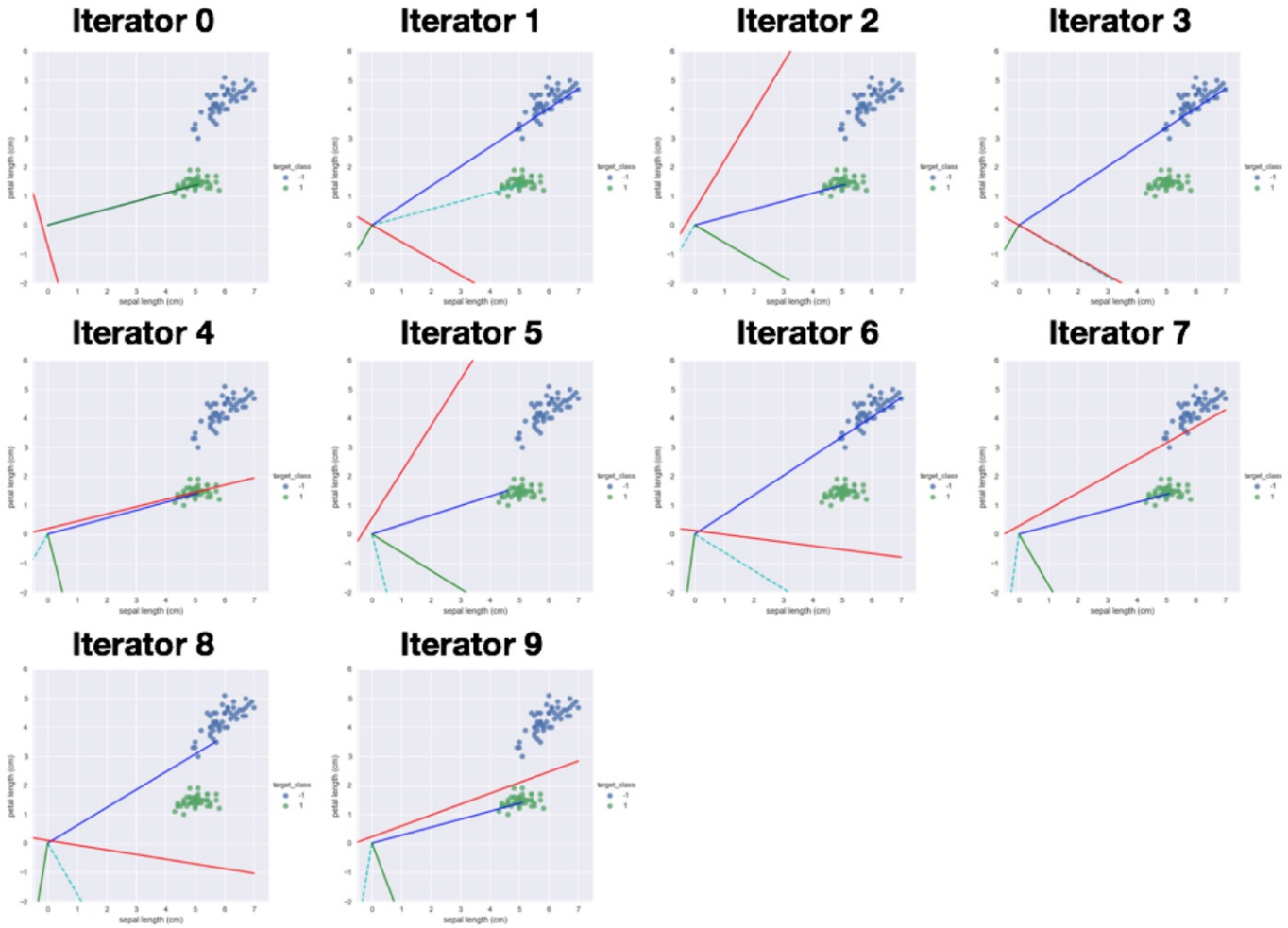
$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

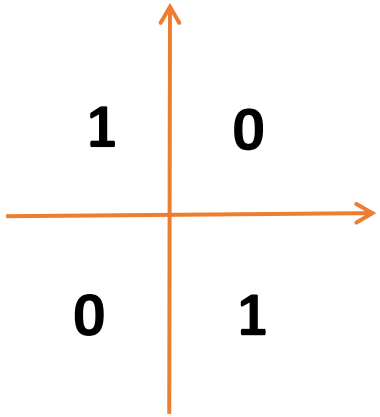
$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

# Perception Example



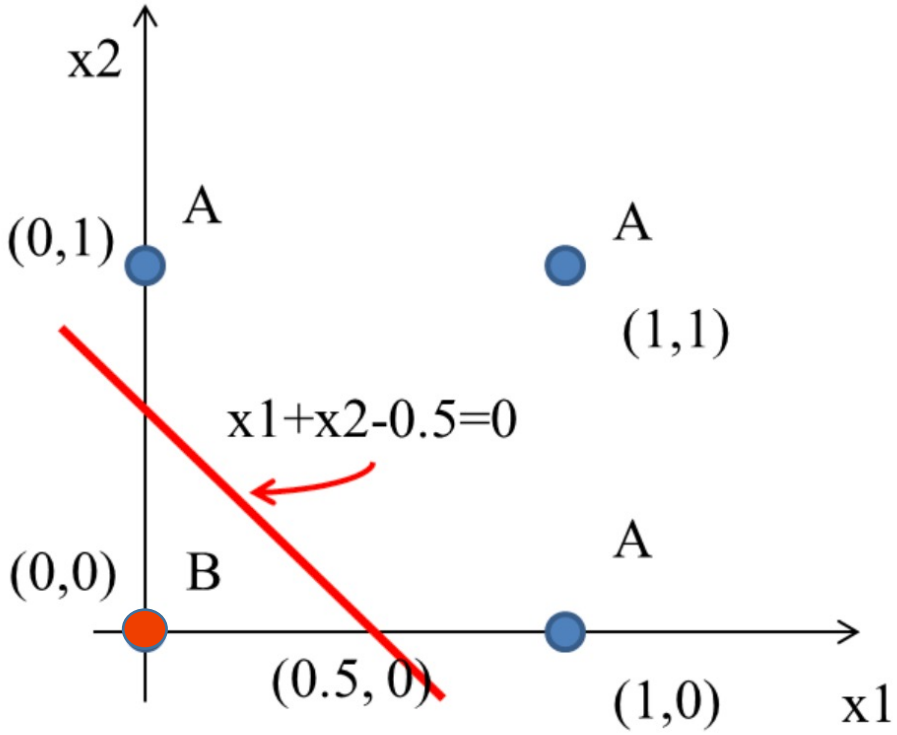
# Perception Example

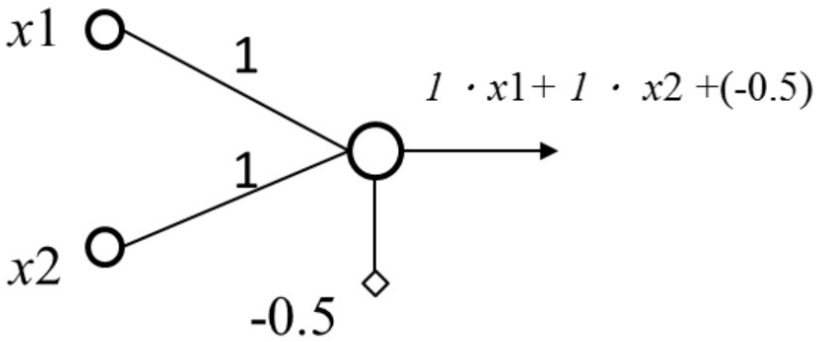


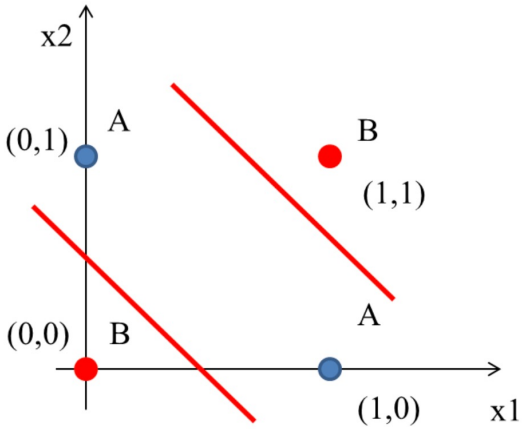

| X1 | X2 | C |
|---|---|---|
| T | T | 0 |
| F | T | 1 |
| T | F | 1 |
| F | F | 0 |

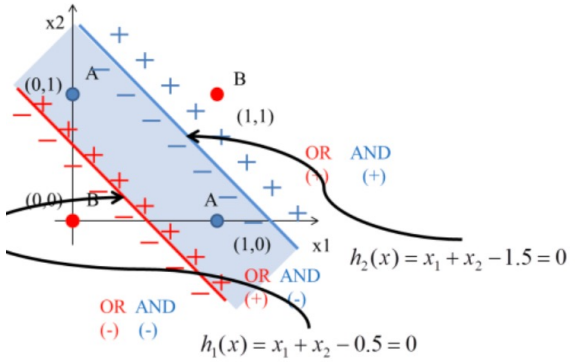
# How does Perceptron work?

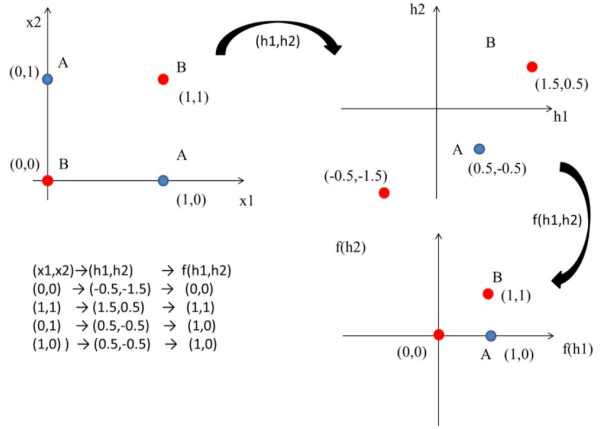


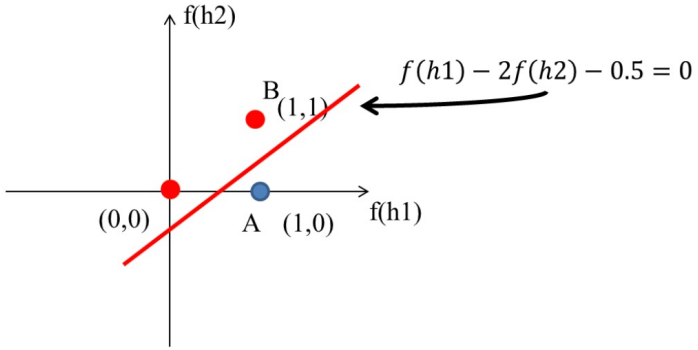



# How does Perceptron work?

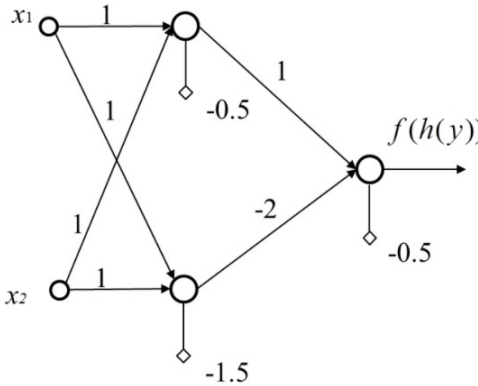



$$y_1 = f(h_1(x)) = f(x_1 + x_2 - 0.5)$$



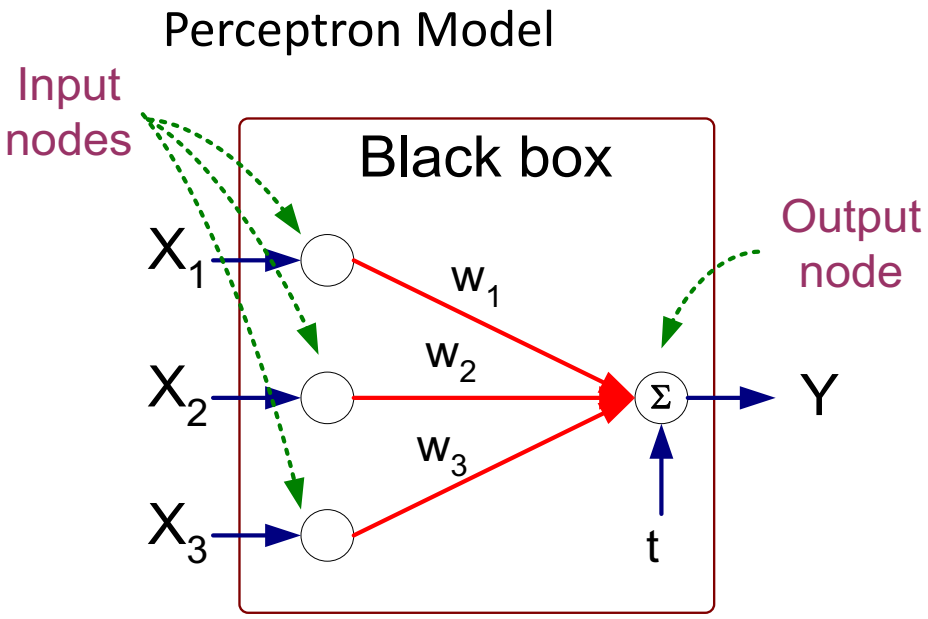

$$y_2 = f(h_2(x)) = f(x_1 + x_2 - 1.5)$$

$$f(v) = \begin{cases} 1 & if\ v \geq 0 \\ o & if\ v < 0 \end{cases}$$

Step activation function

# Artificial Neural Networks (ANN)

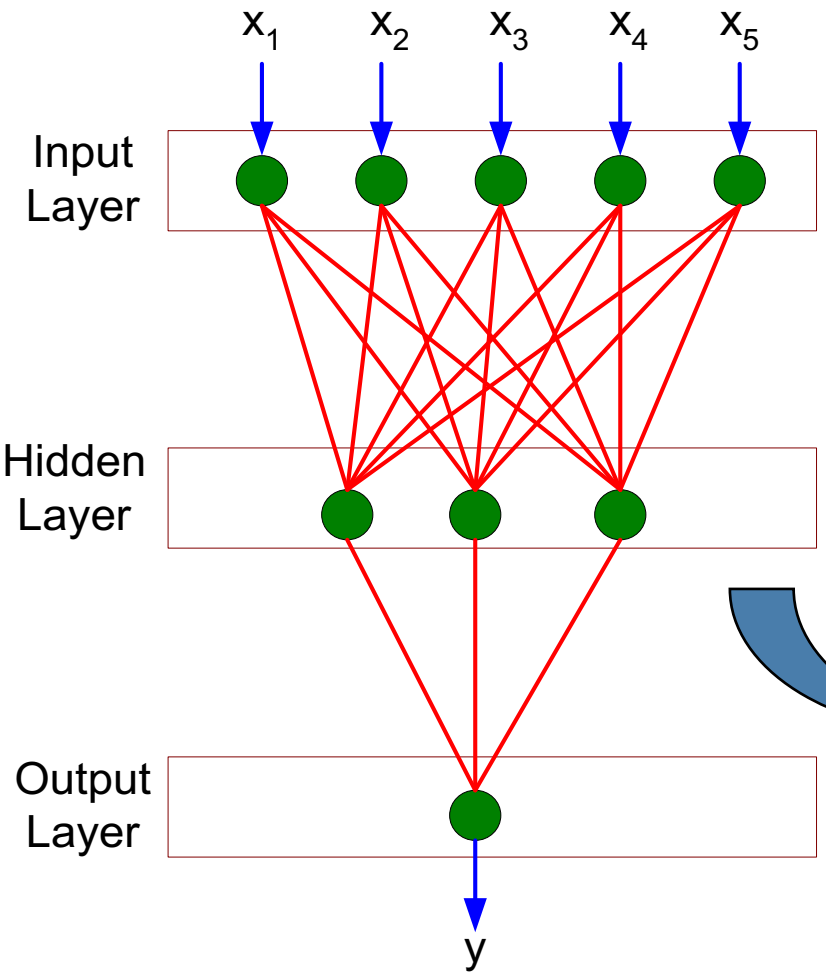
- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t

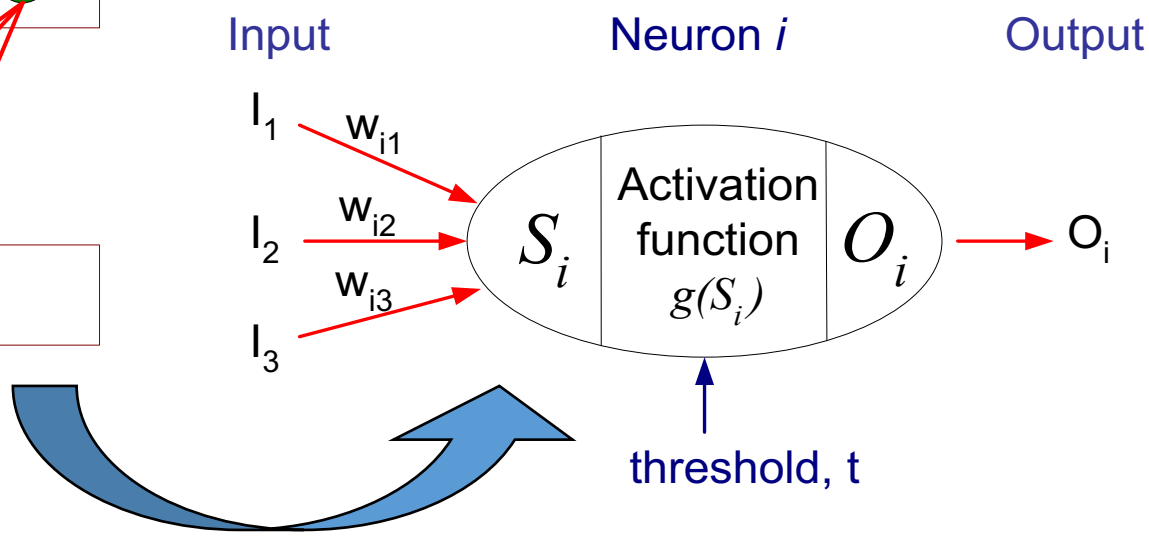



$$Y = I(\sum_i w_i X_i - t) \quad \text{or}$$

$$Y = sign(\sum_i w_i X_i - t)$$

# General Structure of ANN

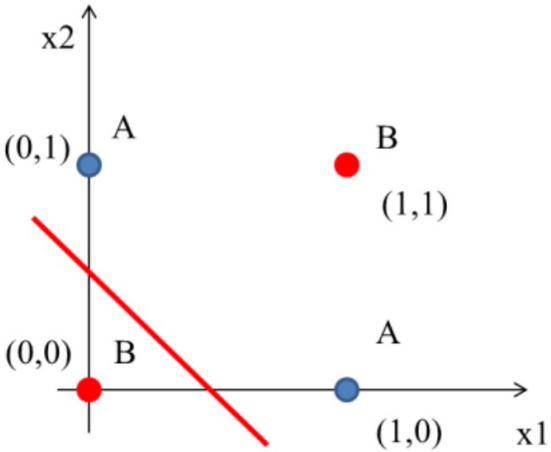


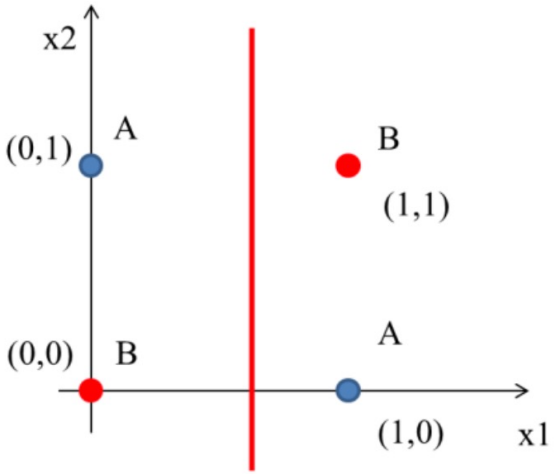



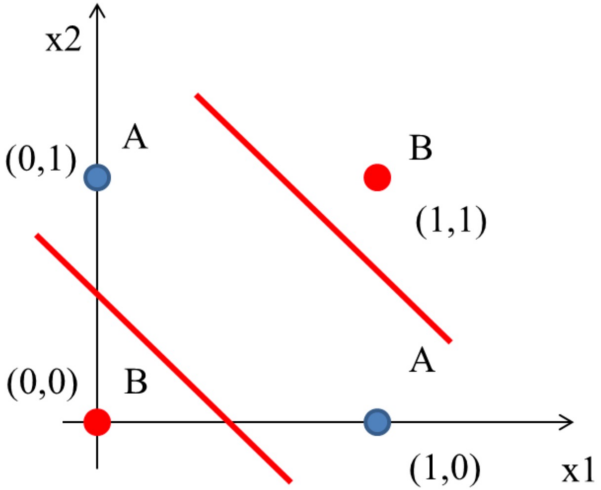

Training ANN means learning the weights of the neurons

# MLP: Multi-layer Perceptron

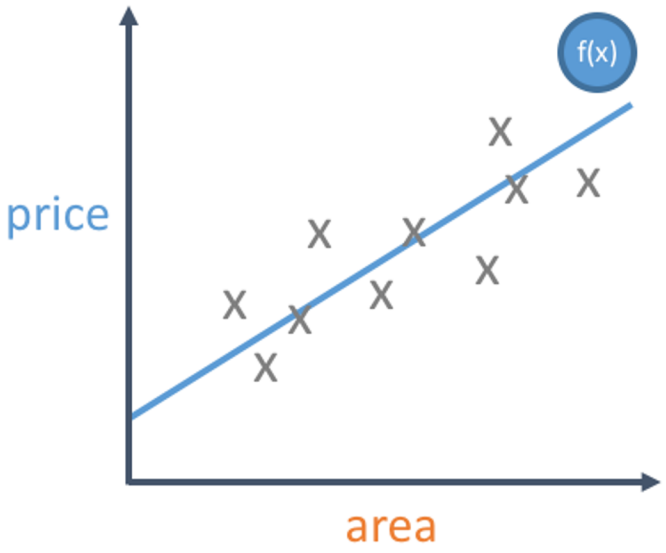
| X1 | X2 | C |
|---|---|---|
| T | T | 0 |
| F | T | 1 |
| T | F | 1 |
| F | F | 0 |

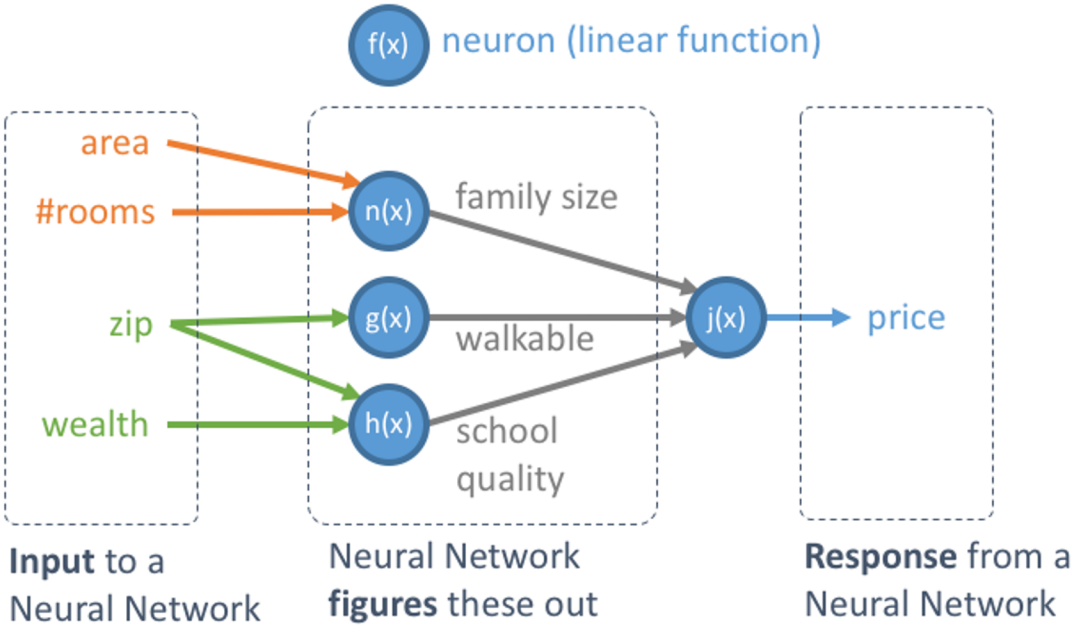


# Feature Learning




A typical Machine Learning problem and a simple Neural Network Andrew Ng Stanford GSB lecture on Artificial Intelligence is the New Electricity

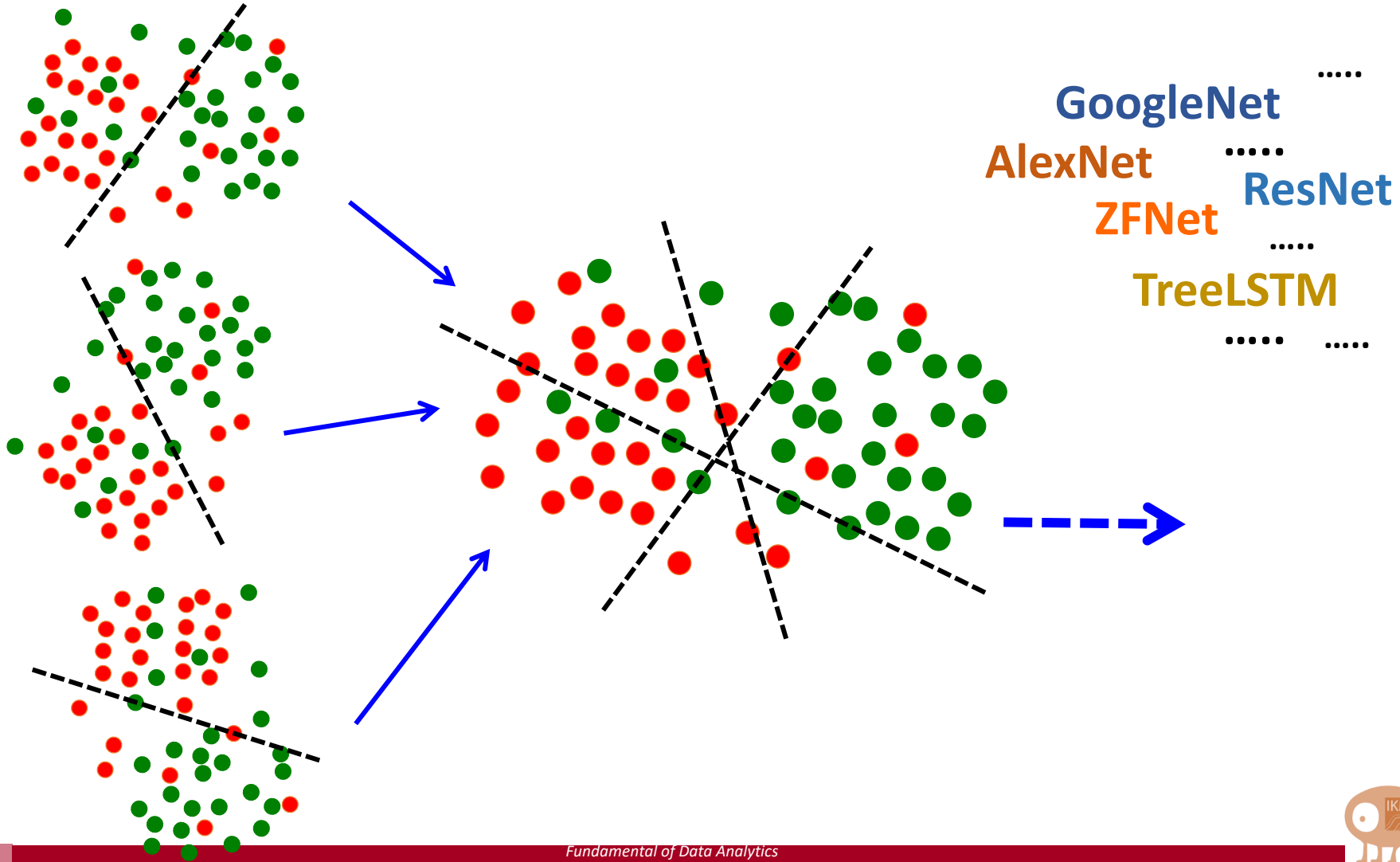
Read more at
**manavsehgal.com**

# Algorithm for learning ANN

- Initialize the weights ($w_0$, $w_1$, …, $w_k$)

- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples
  - Objective function: $E = \sum_i [Y_i - f(w_i, X_i)]^2$
  - Find the weights $w_i'$ s that minimize the above objective function
    - e.g., backpropagation algorithm

# ANN

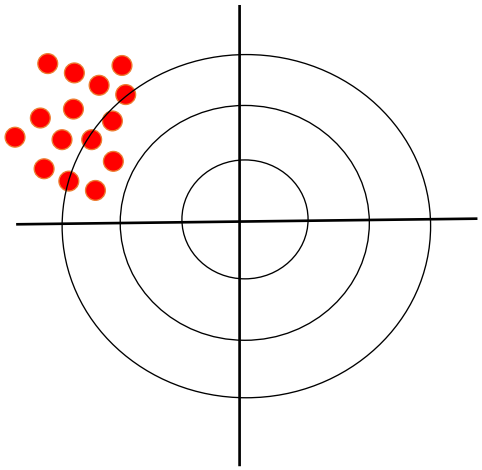


# ANN Practice

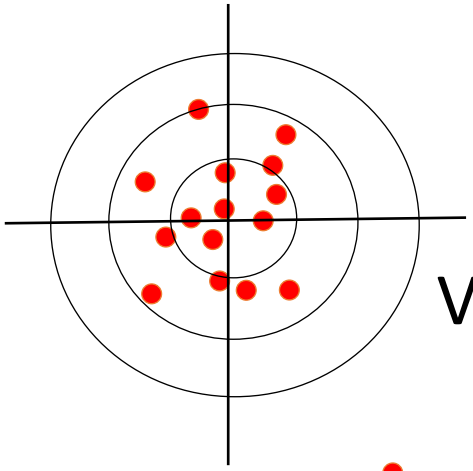
- https://github.com/IKMLab/Feedforward-Tutorial

# Bias & Variance

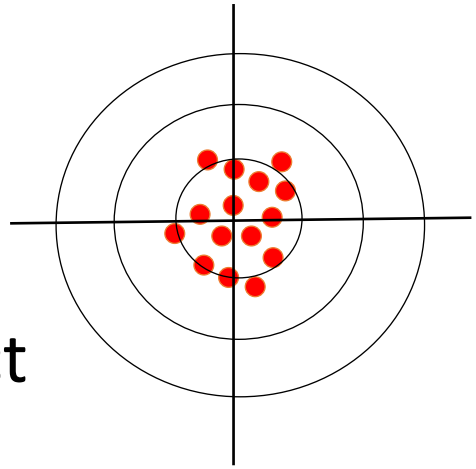
Bias



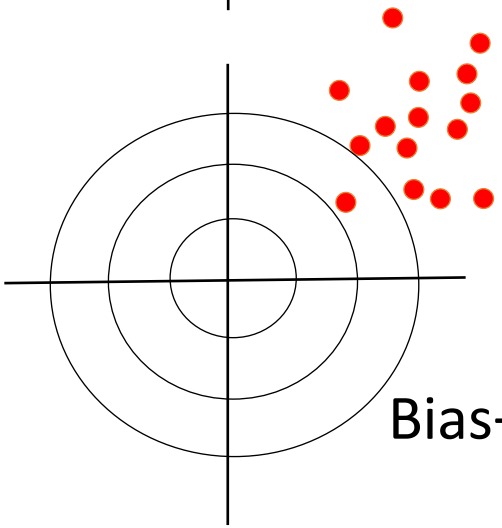
Variance



Perfect



Bias+Variance



# Bias-Variance Trade-Off

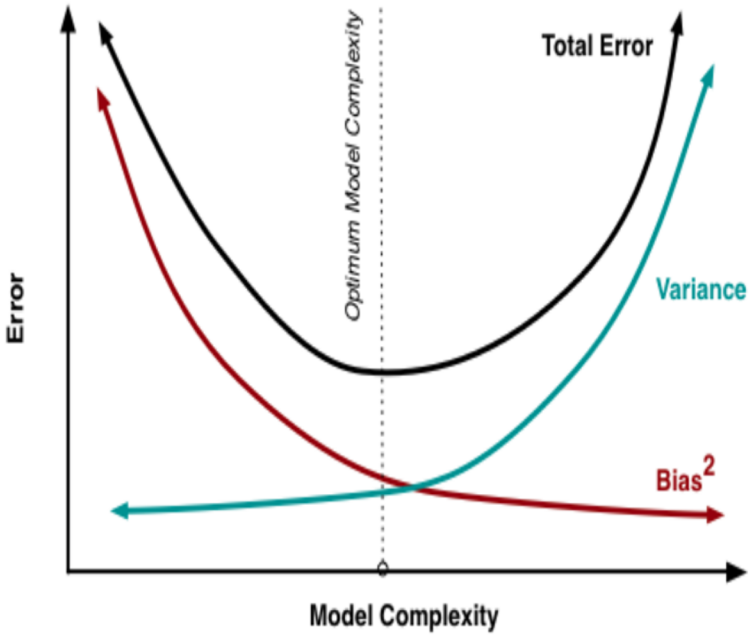
As model complexity increases we see the characteristic U-shape of total error

We reduce bias to an optimum where increasing variance starts to dominate

Finding the right balance is a key machine learning skill

High bias, low variance = **underfitting**

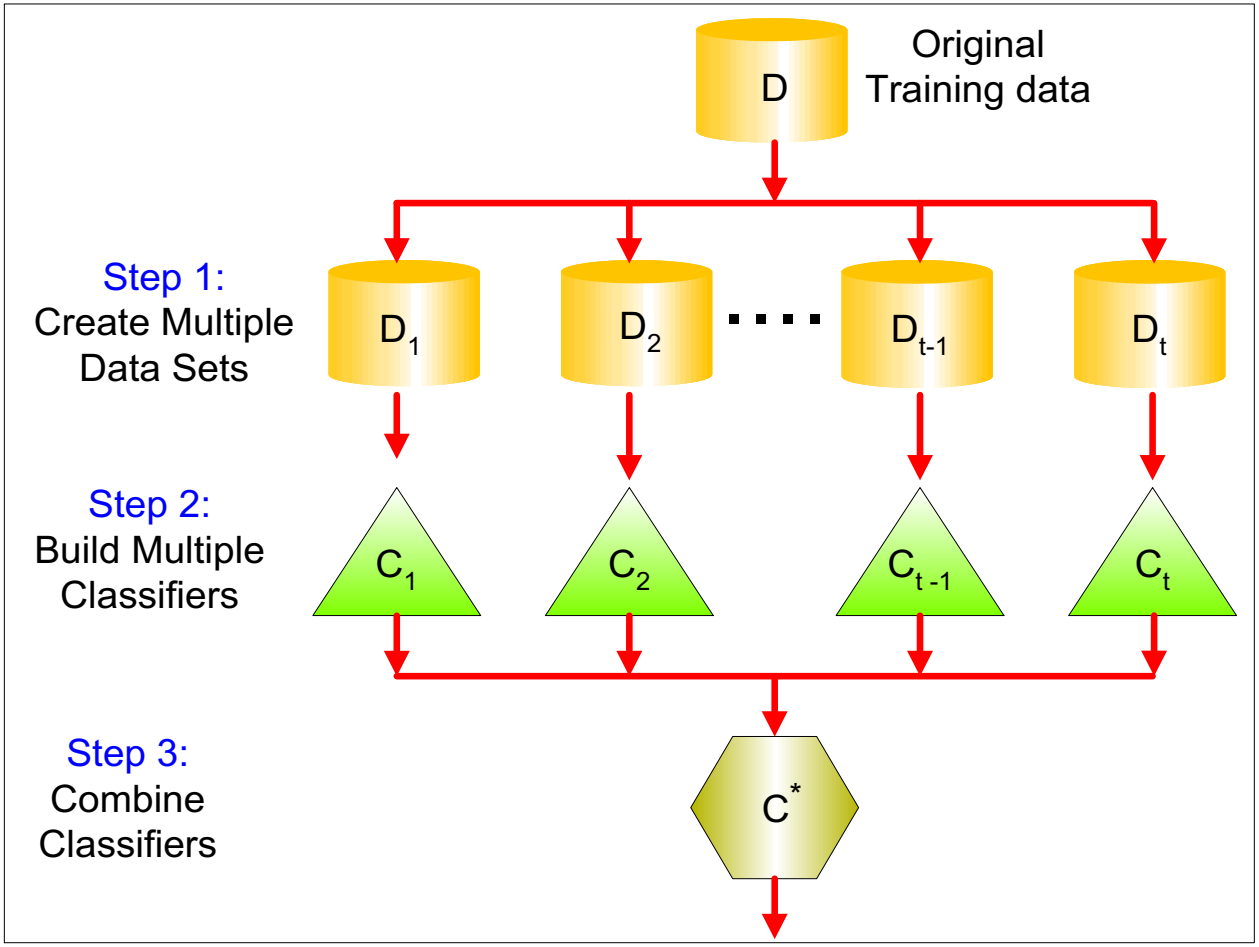
Low bias, high variance = **overfitting**



# Ensemble Methods

- Construct a set of classifiers from the training data
- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers

# General Idea



# Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

# Examples of Ensemble Methods

- How to generate an ensemble of classifiers?
  - Bagging
    - Breiman, 1996
    - Bootstrap Aggregating = Bagging
  - Boosting

# Bagging

- Training
  - Create $k$ bootstrap samples $S[1], S[2], \ldots, S[k]$
  - Build a distinct classifier on each $S[i]$ to produce $k$ classifiers, using the same learning algorithm.
- Testing
  - Classify each new instance by voting of the $k$ classifiers (equal weights)

# Bagging (cont.)

- When does it help?
  - When learner is unstable
    - Small change to training set causes large change in the output classifier
    - True for decision trees, neural networks; not true for *k*-nearest neighbor, naïve Bayesian, class association rules
  - Experimentally, bagging can help substantially for **unstable learners**, may somewhat degrade results for stable learners

Bagging Predictors, Leo Breiman, 1996

# Boosting

- A family of methods:
  - **AdaBoost (Adaptive Boosting)** (Freund & Schapire, 1996)
    - sensitive to noisy data and outliers
- Training
  - Produce a sequence of classifiers (the same base learner)
  - Each classifier is dependent on the previous one, and focuses on the previous one's errors
  - Examples that are incorrectly predicted in previous classifiers are given **higher weights**
- Testing
  - For a test case, the results of the series of classifiers are combined to determine the final class of the test case.

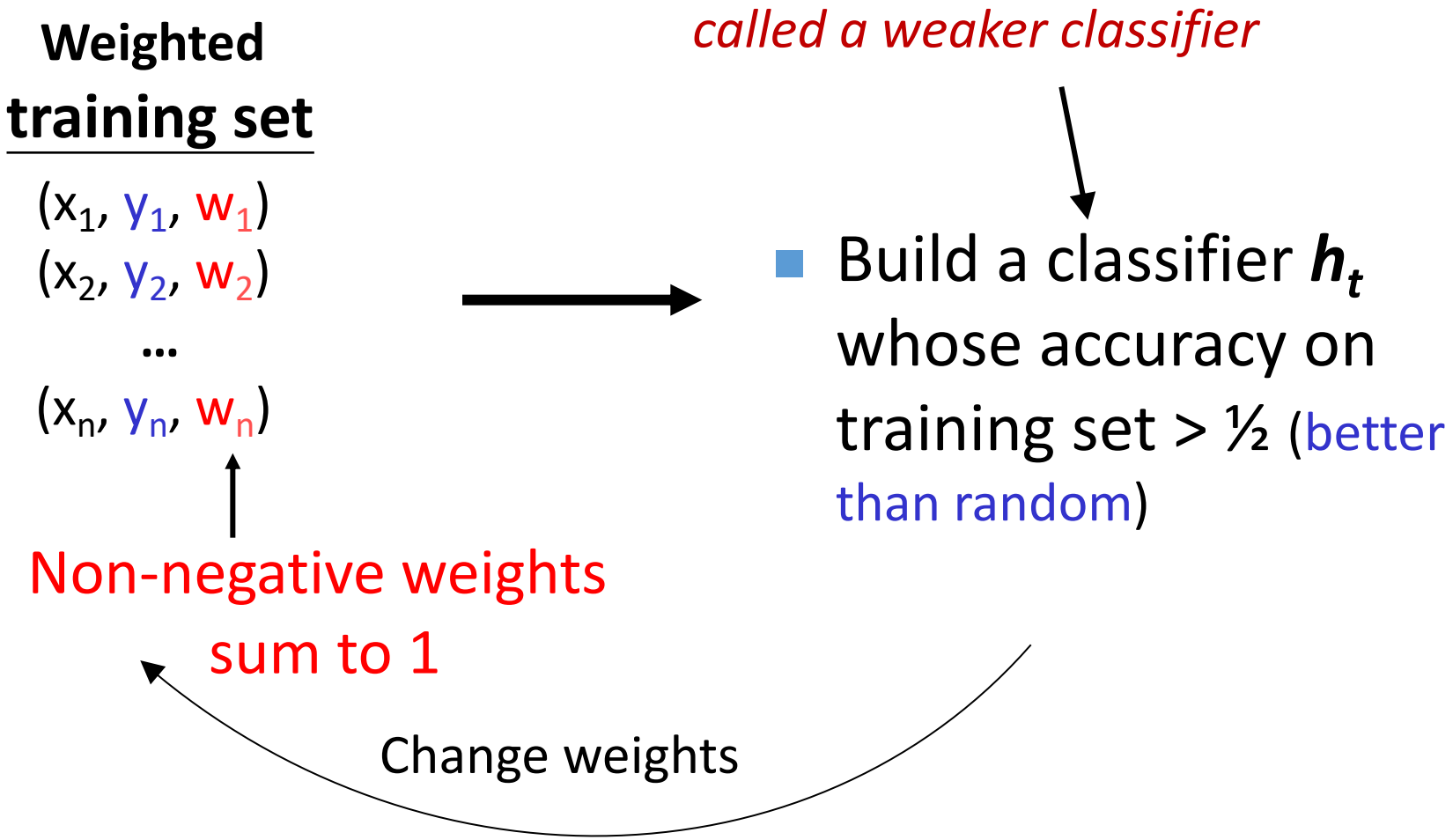
# Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

| Original Data | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Boosting (Round 1) | 7 | 3 | 2 | 8 | 7 | 9 | 4 | 10 | 6 | 3 |
| Boosting (Round 2) | 5 | 4 | 9 | 4 | 2 | 5 | 1 | 7 | 4 | 2 |
| Boosting (Round 3) | 4 | 4 | 8 | 10 | 4 | 5 | 4 | 6 | 3 | 4 |

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

# AdaBoost


Weighted training set
$(x_1, y_1, w_1)$
$(x_2, y_2, w_2)$
…
$(x_n, y_n, w_n)$
Non-negative weights sum to 1
called a weaker classifier
Build a classifier $h_t$ whose accuracy on training set > ½ (better than random)
Change weights

# Does AdaBoost always work?

- The actual performance of boosting depends on the data and the base learner.
  - It requires the base learner to be unstable as bagging.
- Boosting seems to be susceptible to noise.
  - When the number of outliners is very large, the emphasis placed on the hard examples can hurt the performance.

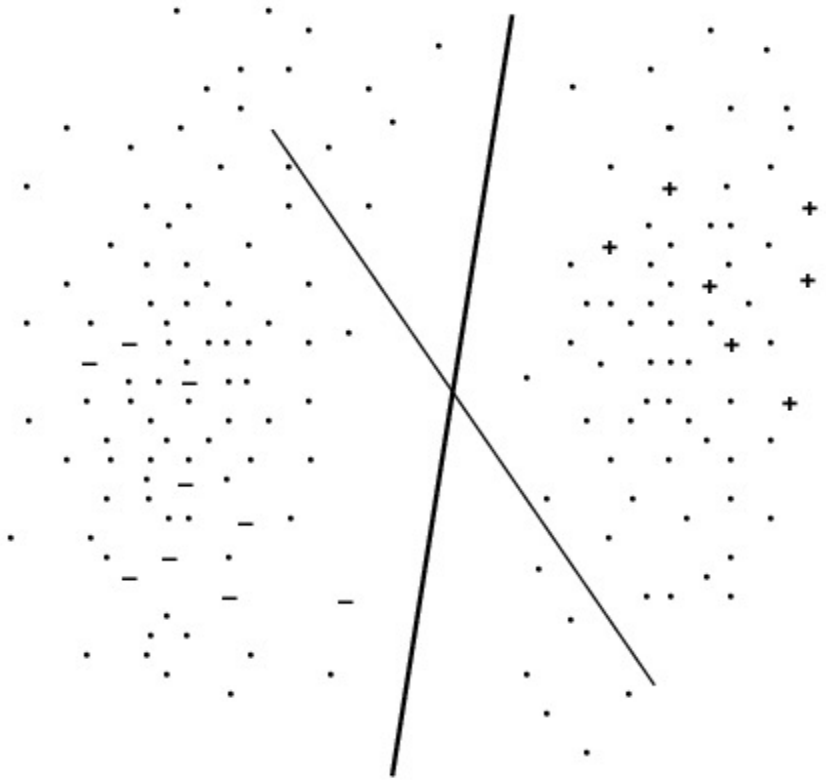
# Supervised Learning with Unlabeled Data

- *Supervised v.s. Unsupervised*
- Assigning labels to training set is
  - expensive
  - time consuming
- Abundance of unlabeled data
  - suggests possible use to improve learning
  - Learning with a small set of labeled examples and a large set of unlabeled examples (LU learning)
  - Learning with positive and unlabeled examples (no labeled negative examples) (PU learning).

# Why Unlabeled Data helpful?

- Consider positive and negative examples
  - as two separate distribution
  - with very large number of samples available parameters of distribution can be estimated well
  - needs only few labeled points to decide which Gaussian is associated with positive and negative class
- In text domains
  - categories can be guessed using term co-occurrences

# Why Unlabeled Data?



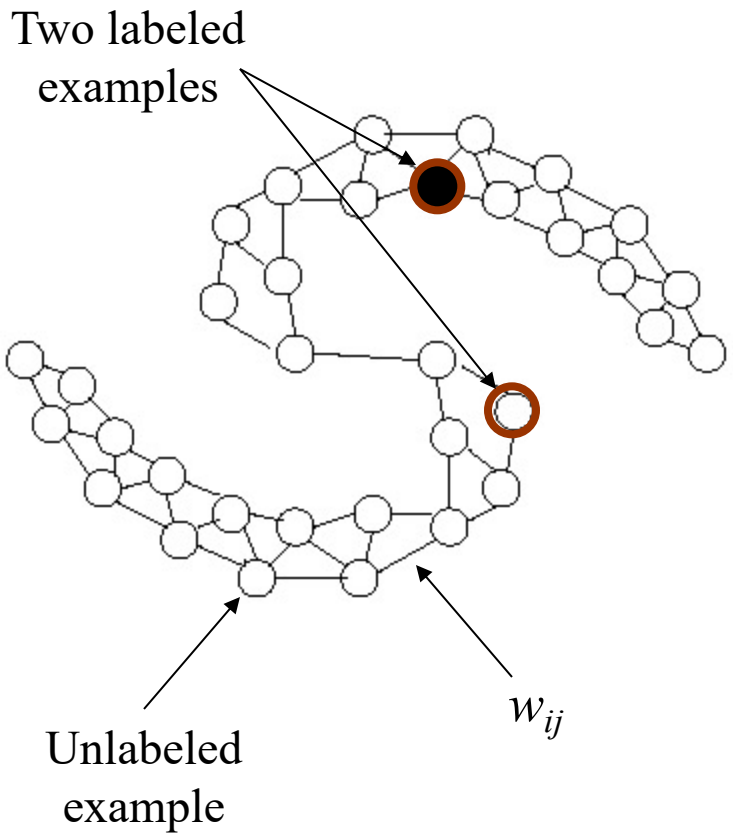
# Semi-supervised learning

| usage | supervised learning | semi-supervised learning | unsupervised learning |
|---|---|---|---|
| $\{(x, y)\}$ labeled data | yes | yes | no |
| $\{x\}$ unlabeled data | no | yes | yes |

- Label propagation
- Transductive learning
- Co-training
- Active learning
- Transfer learning?

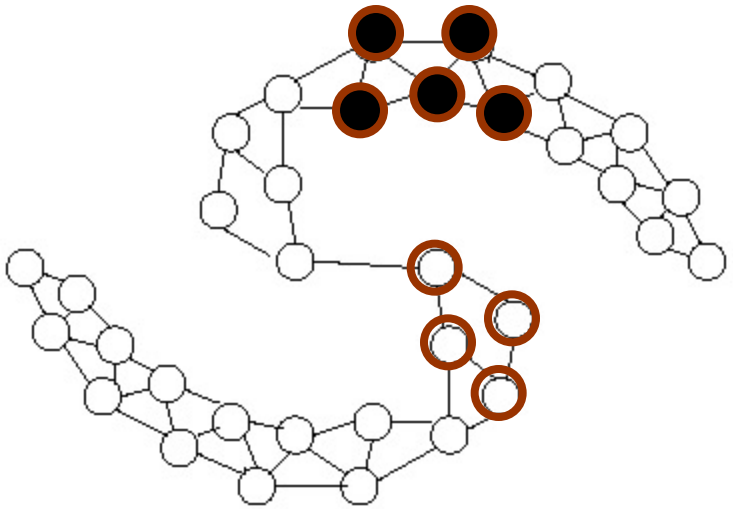
# Label Propagation

- A toy problem
  - Each node in the graph is an example
    - Two examples are labeled
    - Most examples are unlabeled
  - Compute the similarity between examples $S_{ij}$
  - Connect examples to their most similar examples
- How to predicate labels for unlabeled nodes using this graph?



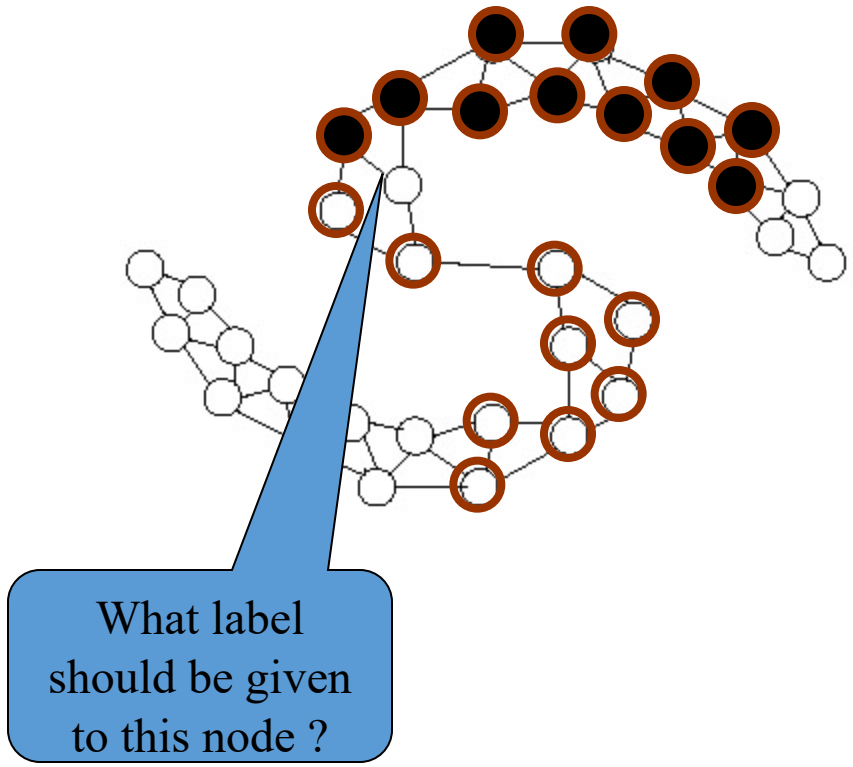
# Label Propagation

- Forward propagation



# Label Propagation

- Forward propagation
- Forward propagation
- Forward propagation
  - How to resolve conflicting cases



# Learning from Positive & Unlabeled data (PU learning)

- Positive examples: One has a set of examples of a class *P*, and
- Unlabeled set: also has a set *U* of unlabeled (or mixed) examples with instances from *P* and also not from *P* (*negative examples*).
- Build a classifier: Build a classifier to classify the examples in *U* and/or future (test) data.
- **Key feature of the problem**: no labeled negative training data.
- We call this problem, PU-learning.

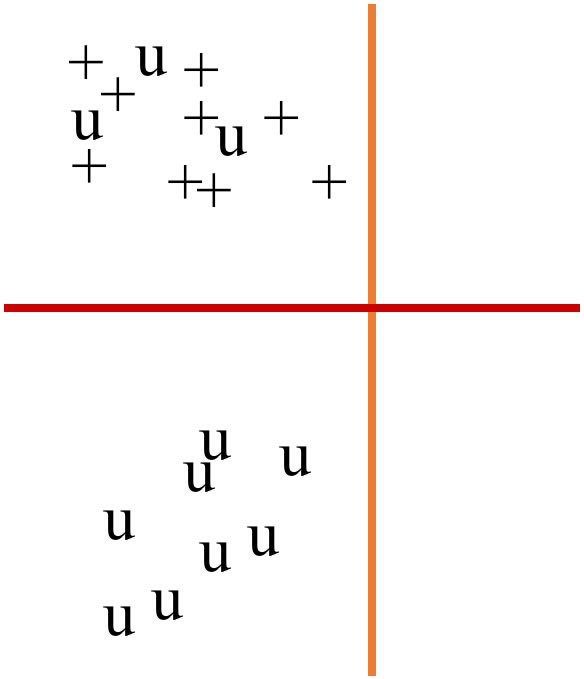
# Applications of the problem

- With the growing volume of online texts available through the Web and digital libraries, one often wants to find those documents that are related to one's work or one's interest.
- For example, given an ICML proceedings,
  - find all machine learning papers from AAAI, IJCAI, KDD
  - No labeling of negative examples from each of these collections.
- Similarly, given one's bookmarks (positive documents), identify those documents that are of interest to him/her from Web sources.

# Direct Marketing

- Company has database with details of its customer – positive examples, but no information on those who are not their customers, i.e., no negative examples.
- Want to find people who are similar to their customers for marketing
- Buy a database consisting of details of people, some of whom may be potential customers – unlabeled examples.

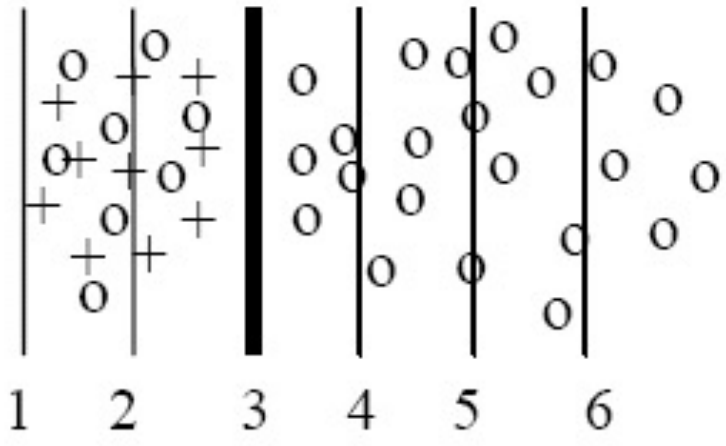
# Are Unlabeled Examples Helpful?



"Not learnable" with only positive examples. However, addition of unlabeled examples makes it learnable.

# An illustration

- Assume a linear classifier. Line 3 is the best solution.

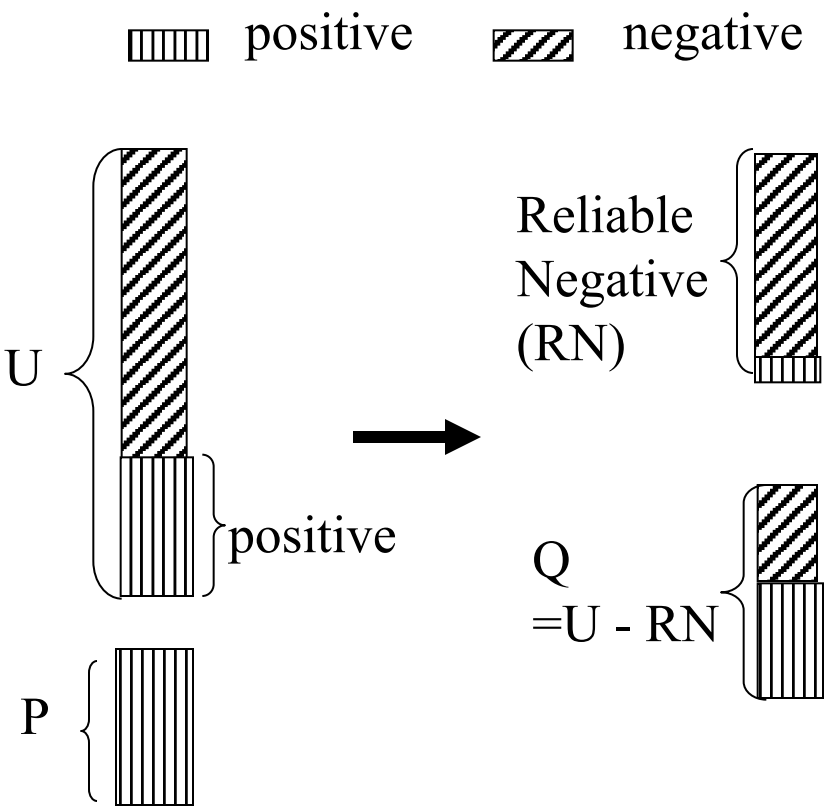



An illustration of the constrained optimization problem

# Existing 2-step strategy

- Step 1: Identifying a set of reliable negative documents from the unlabeled set.
  - S-EM [Liu et al, 2002] uses a Spy technique,
  - PEBL [Yu et al, 2002] uses a 1-DNF technique
  - Roc-SVM [Li & Liu, 2003] uses the Rocchio algorithm.
  - …
- Step 2: Building a sequence of classifiers by iteratively applying a classification algorithm and then selecting a good classifier.
  - S-EM uses the Expectation Maximization (EM) algorithm, with an error based classifier selection mechanism
  - PEBL uses SVM, and gives the classifier at convergence, i.e., no classifier selection.
  - Roc-SVM uses SVM with a heuristic method for selecting the final classifier.

# Step 1




# Step 2

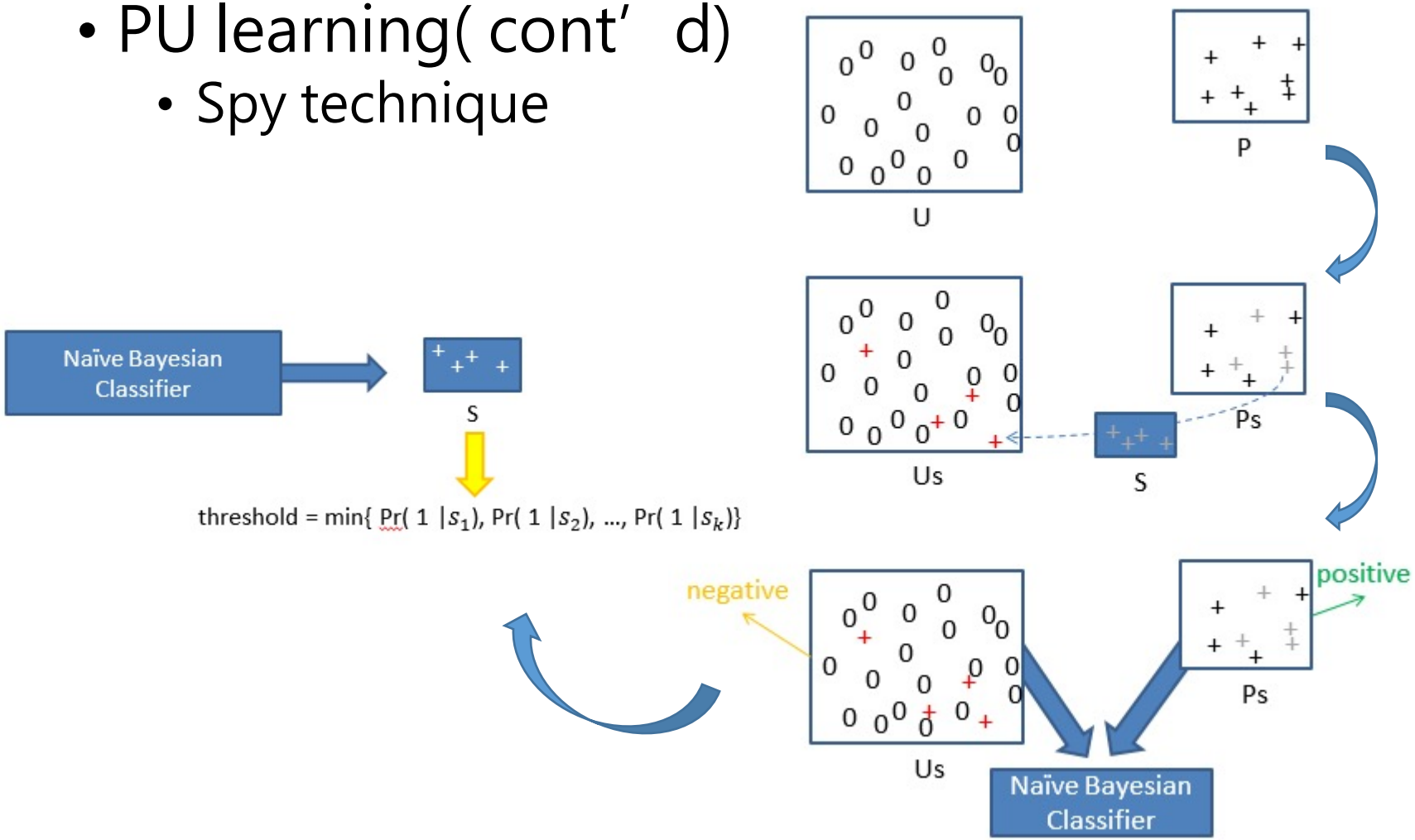
Using P, RN and Q to build the final classifier iteratively

or

Using only P and RN to build a classifier

# Spy

- PU learning( cont’ d)
  - Spy technique



# Step1: The Spy technique

- Sample a certain % of positive examples and put them into unlabeled set to act as "spies".
- Run a classification algorithm *assuming all unlabeled examples are negative*,
  - we will know the behavior of those actual positive examples in the unlabeled set through the "spies".
- We can then extract reliable negative examples from the unlabeled set more accurately.

# Step 1: Other methods

- 1-DNF method:
  - Find the set of words W that occur in the positive documents more frequently than in the unlabeled set.
  - Extract those documents from unlabeled set that do not contain any word in W. These documents form the reliable negative documents.

- Rocchio method from information retrieval.
- Naïve Bayesian method.

# Do they follow the theory?

- Yes, heuristic methods because
  - Step 1 tries to find some initial reliable negative examples from the unlabeled set.
  - Step 2 tried to identify more and more negative examples iteratively.
- The two steps together form an iterative strategy of increasing the number of unlabeled examples that are classified as negative while maintaining the positive examples correctly classified.

# Co-training Algorithm
[Blum and Mitchell, 1998]

Given: labeled data L,

unlabeled data U

Loop:

Train h1 (e.g., hyperlink classifier) using L

Train h2 (e.g., page classifier) using L

Allow h1 to label *p* positive, *n* negative examples from U

Allow h2 to label *p* positive, *n* negative examples from U

Add these most confident self-labeled examples to L

# Co-training Algorithm

- Labeled data are used to infer two Naïve Bayes classifiers, one for each view
- Each classifier will
  - examine unlabeled data
  - pick the most confidently predicted positive and negative examples
  - add these to the labeled examples
- Classifiers are now retrained on the augmented set of labeled examples
- *As well as the text on the pages being relevant to the classification of the page, the hyperlinks also contain useful information*

# Co-training: Experimental Results

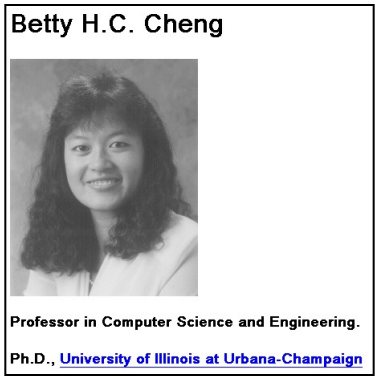
- begin with 12 labeled web pages (academic course)
- provide 1,000 additional unlabeled web pages
- average error: learning from labeled data 11.1%
- average error: co-training 5.0%

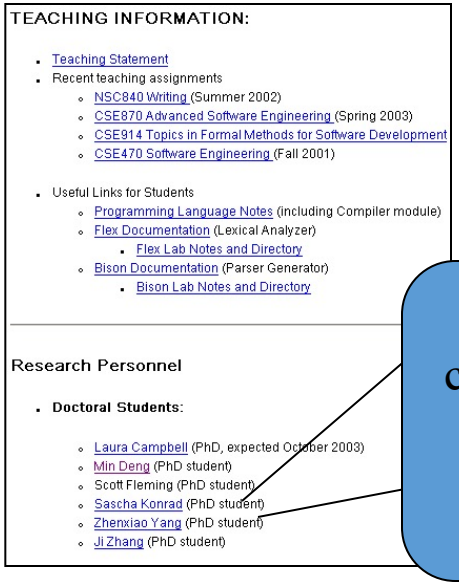
| | Page-base classifier | Link-based classifier | Combined classifier |
|---|---|---|---|
| Supervised training | 12.9 | 12.4 | 11.1 |
| Co-training | 6.2 | 11.6 | 5.0 |

# An example: Co-training for Semi-supervised Learning

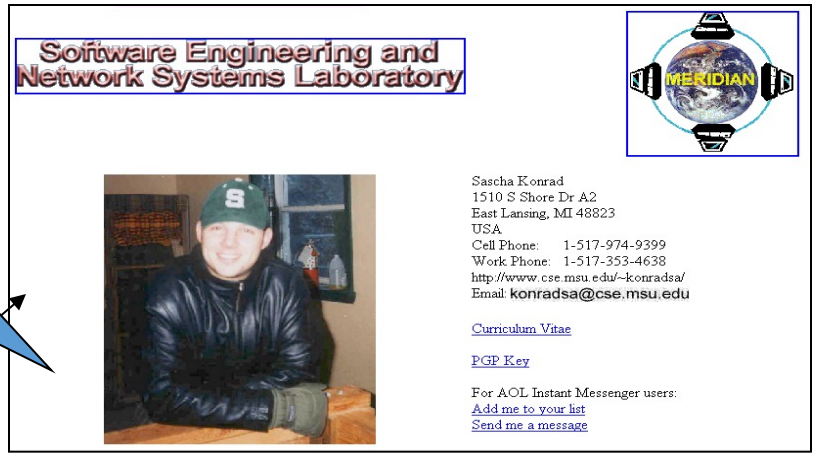
- Consider the task of classifying web pages into two categories: category for students and category for professors
- Two aspects of web pages should be considered
  - Content of web pages
    - "I am currently the second year Ph.D. student …"
  - Hyperlinks
    - "My advisor is …"
    - "Students: …"

# Co-training for Semi-Supervised Learning




It is easier to classify this web page using hyperlinks




It is easy to classify the type of this web page based on its content

# Co-training

- Two representation for each web page


**Content representation**:

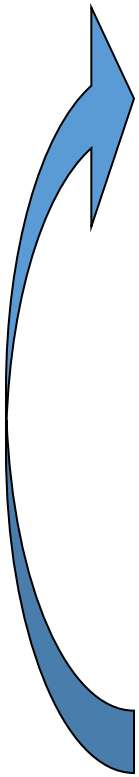
(doctoral, student, computer, university…)

**Hyperlink representation**:

Inlinks: Prof. Cheng

Oulinks: Prof. Cheng

# Co-training: Classification Scheme



1. Train a content-based classifier using labeled web pages
2. Apply the content-based classifier to classify unlabeled web pages
3. Label the web pages that have been confidently classified
4. Train a hyperlink based classifier using the web pages that are initially labeled and labeled by the classifier
5. Apply the hyperlink-based classifier to classify the unlabeled web pages
6. Label the web pages that have been confidently classified

# Learning flow