CSE 590
Data Science Fundamentals

Classifiers

Klaus Mueller

Computer Science Department
Stony Brook University and SUNY Korea

Lecture	Topic	Projects
1	Intro, schedule, and logistics	
2	Data Science components and tasks	
3	Data types	Project #1 out
4	Introduction to R, statistics foundations	
5	Introduction to D3, visual analytics	
6	Data preparation and reduction	
7	Data preparation and reduction	Project #1 due
8	Similarity and distances	Project #2 out
9	Similarity and distances	
10	Cluster analysis	
11	Cluster analysis	
12	Pattern mining	Project #2 due
13	Pattern mining	
14	Outlier analysis	
15	Outlier analysis	Final Project proposal due
16	Classifiers	
17	Midterm	
18	Classifiers	
19	Optimization and model fitting	
20	Optimization and model fitting	
21	Causal modeling	
22	Streaming data	Final Project preliminary report due
23	Text data	
24	Time series data	
25	Graph data	
26	Scalability and data engineering	
27	Data journalism	
	Final project presentation	Final Project slides and final report due

Covered Topics

Decision Trees

Naïve Bayesian Classifiers

Support Vector Machines (SVM)

Neural Networks

Classification – A Two-Step Process

Model construction: describing a set of predetermined classes

- Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
- The set of tuples used for model construction: training set
- The model is represented as classification rules, decision trees, or mathematical formulae

Model usage: for classifying future or unknown objects

- Estimate accuracy of the model
  - the known label of test sample is compared with the classified result from the model
  - accuracy rate is the percentage of test set samples that are correctly classified by the model
  - test set is independent of training set, otherwise over-fitting will occur

Classifiers – By Model

Generative

- models how the data was generated in order to categorize the data
- asks the question: based on my generation assumptions, which category is most likely to generate this data?
- example: naïve Bayes
- explicitly models the joint probability distribution

Discriminative

- does not care about how the data was generated
- it simply categorizes a given data item
- examples: SVM, neural network, decision tree

Classifiers – By Degree of Supervision

Supervised learning (classification)

- Supervision: the training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
- New data is classified based on the training set

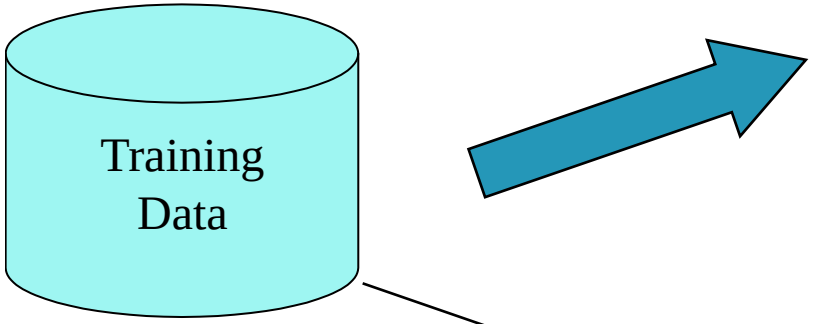
Unsupervised learning (clustering)

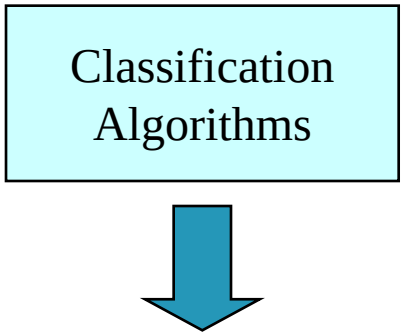
- The class labels of training data is unknown
- Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

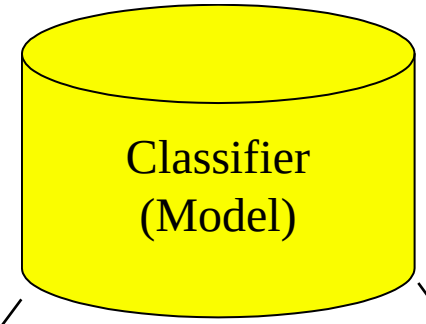
Active learning

- Hybrid – ask user to supervise only the “hard” cases

Example – Training



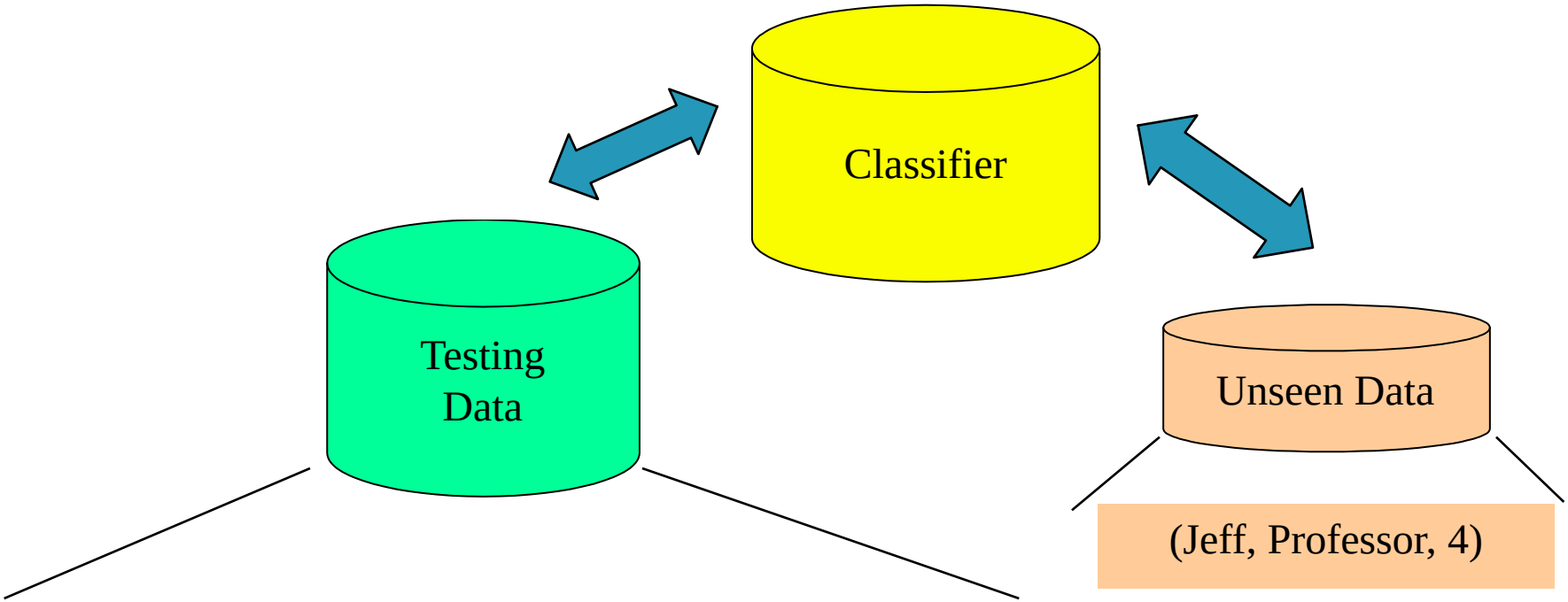




NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

IF rank = ‘professor’
OR years > 6
THEN tenured = ‘yes’

Example – Classification



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Classifier – Figures of Merit

Predictive accuracy

Speed and scalability

- time to construct the model
- time to use the model

Robustness

- handling noise and missing values

Scalability

- efficiency in disk-resident databases

Interpretability:

- understanding and insight provided by the model

Goodness of rules

- decision tree size
- compactness of classification rules

Decision Trees

Preparation

Data cleaning

- Preprocess data in order to reduce noise and handle missing values

Relevance analysis (feature selection)

- Remove the irrelevant or redundant attributes

Data transformation

- Generalize and/or normalize data

Decision Trees

Decision tree

- A flow-chart-like tree structure
- Internal node denotes a test on an attribute
- Branch represents an outcome of the test
- Leaf nodes represent class labels or class distribution

Decision tree generation consists of two phases

- Tree construction
  - at start, all the training examples are at the root
  - partition examples recursively based on selected attributes
- Tree pruning
  - identify and remove branches that reflect noise or outliers

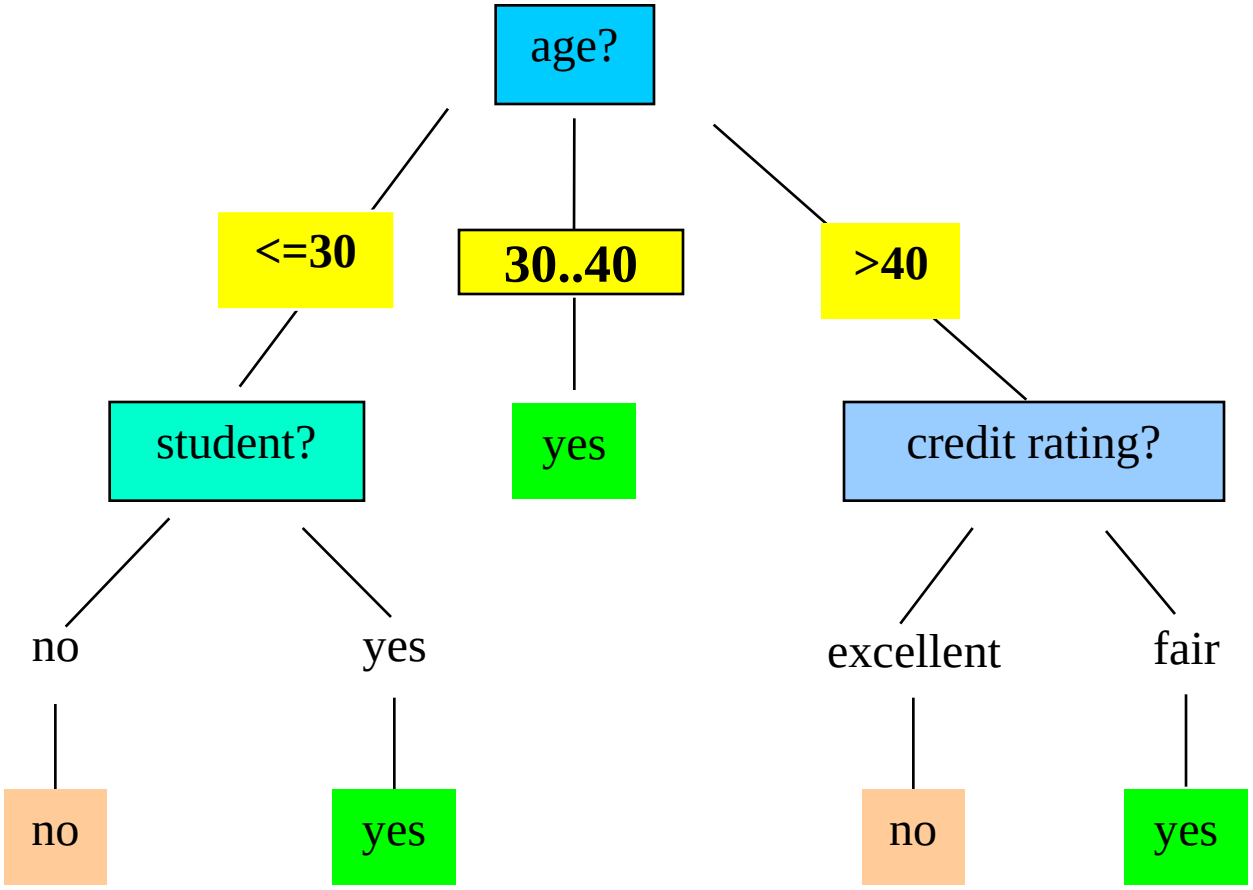
Use of decision tree: Classifying an unknown sample

- test the attribute values of the sample against the decision tree

Example – Consumer database

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

Decision Tree – Will They Buy A Computer?



Extracting Classification Rules from Trees

Represent the knowledge in the form of IF-THEN rules

- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand

Example

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

Algorithm For Decision Tree Construction

Basic algorithm (greedy algorithm)

- tree is constructed in a top-down recursive divide-and-conquer manner
- at start, all the training examples are at the root
- attributes are categorical (if continuous-valued, they are discretized in advance)
- examples are partitioned recursively based on selected attributes
- test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

Conditions for stopping partitioning

- all samples for a given node belong to the same class
- there are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
- there are no samples left

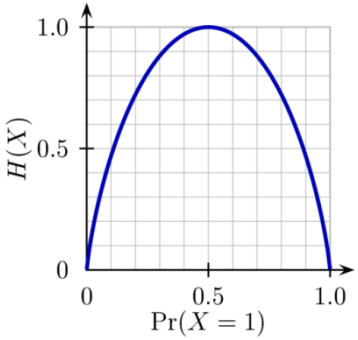
Attribute Selection Measures (1)

Information gain

- all attributes are assumed to be categorical
- can be modified for continuous-valued attributes – discretize

Uses an entropy measure

$$E(v_i) = -\sum_{j=1}^{k} p_j \log_2(p_j).$$



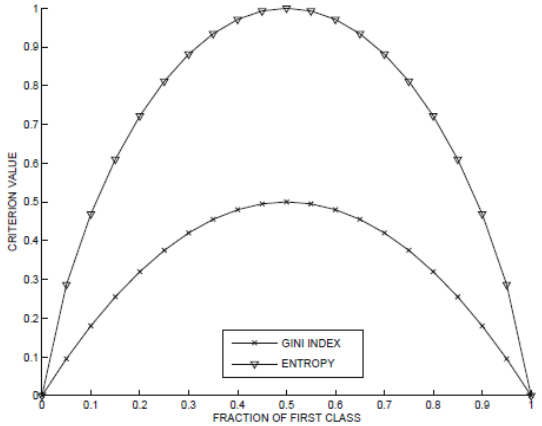
- p_j is the frequency of class j for attribute value v_i
- higher values of the entropy imply greater "mixing" of different classes in that attribute value/level
- a value of 0 implies perfect separation, and, therefore, the largest possible discriminative power of that value
- overall entropy of an attribute for selection is

$$E = \sum_{i=1}^{r} n_i E(v_i)/n.$$

Attribute Selection Measures (2)

Gini index

- definition $G(v_i) = 1 - \sum_{j=1}^{k} p_j^2.$
- p_j is the frequency of class j for attribute value v_i
- the index is highest when the distribution is random (1/k)
- zero when the value is an optimal discriminant
- select the attribute with lowest Gini index $G = \sum_{i=1}^{r} n_i G(v_i)/n.$
- similar to information gain



Expected Information Gain

The change in information entropy H from a prior state to a state that adds some information

$$IG(T, a) = H(T) - H(T|a)$$

- H(T): entropy of prior state without new information
- H(T/a): entropy of new information given prior state
- IG(T,a): information gain from new information a

Example – Computer Purchase

Class P: buys_computer = "yes"

Class N: buys_computer = "no"

H(p, n) = I(9, 5) =0.940

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

Compute the entropy for age:

$$E(age) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.69$$

H(buys_computer/age) = Gain(age) = 0.940 - 0.690 = 0.250

Similarly

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

→ age is best

Why Decision Trees?

Fast and easy

- relatively faster learning speed (than other classification methods)
- convertible to simple and easy to understand classification rules
- can use SQL queries for accessing databases
- comparable classification accuracy with other methods

One more note – testing and training

- 2-to-1 rule: separate training (2/3) and testing (1/3) sets

Probabilistic Classifiers

Bayesian Theorem

Given training data *D*, the *a-posteriori probability of a hypothesis h, P(h|D)* follows the Bayes theorem

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

Naïve Bayes:

- attributes are conditionally independent

Example – Weather Prediction

Given a training set, we can compute the probabilities of the following attributes

Outlook	P	N
sunny	2/9	3/5
overcast	4/9	0
rain	3/9	2/5
Temperature		
hot	2/9	2/5
mild	4/9	2/5
cool	3/9	1/5

Humidity	P	N
high	3/9	4/5
normal	6/9	1/5
Windy		
true	3/9	3/5
false	6/9	2/5

A possible question is now

- given these observations $X=<x_1,\ldots,x_k>$, what is the probability of an event or class, such as P(play-tennis|X)

Bayesian classification

The classification problem may be formalized using a-posteriori probabilities:

P(C|X) = prob. that the sample tuple $X=<x_1,\ldots,x_k>$ is of class C.

- for example. P(class=N | outlook=sunny, windy=true,…)

Idea:

- assign to sample X the class label C such that P(C|X) is maximal

Estimating A-Posteriori Probabilities

Bayes theorem:

$$P(C|X) = P(X|C) \cdot P(C) / P(X)$$

P(X) is constant for all classes

P(C) = relative frequency of class C samples

C such that P(C|X) is maximum =

C such that P(X|C)·P(C) is maximum

Problem: computing P(X|C) is unfeasible!

Naïve Bayesian Classification

Make the naïve assumption:

- attribute independence $P(x_1,\ldots,x_k|C) = P(x_1|C)\cdot\ldots\cdot P(x_k|C)$

If i-th attribute is categorical:

- $P(x_i|C)$ is estimated as the relative frequency of samples having value x_i as i-th attribute in class C

If i-th attribute is continuous:

- $P(x_i|C)$ is estimated through a Gaussian density function

Computationally easy in both cases

Play-tennis Example – Estimate $P(x_i|C)$

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

$P(x_i|C)$

P(p) = 9/14
P(n) = 5/14

outlook	
P(sunny|p) = 2/9	**P(sunny|n) = 3/5**
P(overcast|p) = 4/9	**P(overcast|n) = 0**
P(rain|p) = 3/9	**P(rain|n) = 2/5**
temperature	
P(hot|p) = 2/9	**P(hot|n) = 2/5**
P(mild|p) = 4/9	**P(mild|n) = 2/5**
P(cool|p) = 3/9	**P(cool|n) = 1/5**
humidity	
P(high|p) = 3/9	**P(high|n) = 4/5**
P(normal|p) = 6/9	**P(normal|n) = 2/5**
windy	
P(true|p) = 3/9	**P(true|n) = 3/5**
P(false|p) = 6/9	**P(false|n) = 2/5**

Play-tennis example – classifying X

Wish to find out P(p|X) given an unseen sample condition
X = <rain, hot, high, false>

P(X|p)·P(p) =
P(rain|p)·P(hot|p)·P(high|p)·P(false|p)·P(p) =
3/9·2/9·3/9·6/9·9/14 = 0.010582

P(X|n)·P(n) =
P(rain|n)·P(hot|n)·P(high|n)·P(false|n)·P(n) =
2/5·2/5·4/5·2/5·5/14 = 0.018286

Sample X is classified in class n (don't play)

Some Caution

Assume you (age 40) are told that you have a positive mammogram finding M+ for breast cancer C+

The probability for actually having breast cancer C+ is

- $P(C+|M+) = P(M+|C+)\, P(C+) / P(M+)$
- $P(C+|M+) = P(M+|C+)\, P(C+) / (P(M+|C+) + P(M+|C-)$

Using the probabilities:

- p for having breast cancer at age 40 is $P(C+) = 0.01$
- p for correct detection with M (TP) is $P(M+|C+) = 0.8$ (sensitivity)
- p for wrong detection (FP) is $P(M+|C-) = 0.096$ (1-specificity)

Via Bayes' rule p for actually having breast cancer C+ is

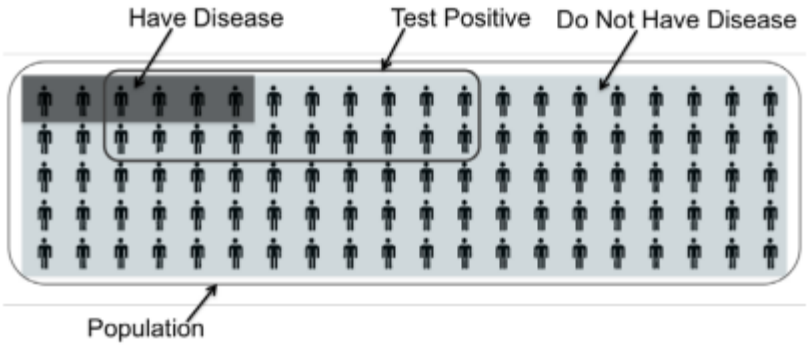
- $P(C+|M+) = 0.8 \cdot 0.01 / (0.8 \cdot 0.01 + 0.096 \cdot 0.99) = 0.078$ (7.8%)
- turns out 95 out of 100 doctors estimated this probability to be between 70% and 80%
- solution: use more than one test (here, e.g., use ultrasound, too)

Bayes' Rule Can Be Confusing

The probability was small since the FP was multiplied by a large population

- people also do poorly with measure of uncertainty

Visualization can help here



"Improving Bayesian Reasoning: The Effects of Phrasing, Visualization, and Spatial Ability" by Alvitta Ottley, Evan M. Peck, Lane T. Harrison, Daniel Afergan, Caroline Ziemkiewicz, Holly A. Taylor, Paul K. J. Han, and Remco Chang, IEEE TVCG, January 2016

The Independence Hypothesis…

… makes computation possible

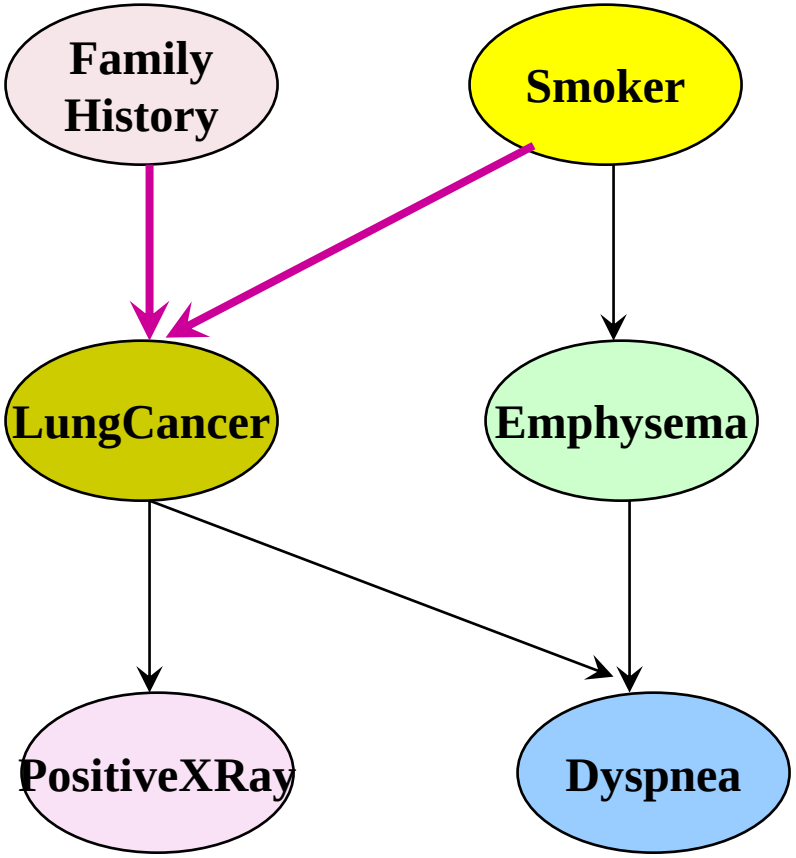
… yields optimal classifiers when satisfied

… but is seldom satisfied in practice, as attributes (variables) are often correlated.

Attempts to overcome this limitation:

- Bayesian networks, that combine Bayesian reasoning with causal relationships between attributes
- Decision trees, that reason on one attribute at the time, considering most important attributes first

Bayesian Belief Networks



Bayesian Belief Networks

	(FH, S)	(FH, ~S)	(~FH, S)	(~FH, ~S)
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

The conditional probability table for the variable LungCancer

Support Vector Machines

Linear classifiers: Which Hyperplane?

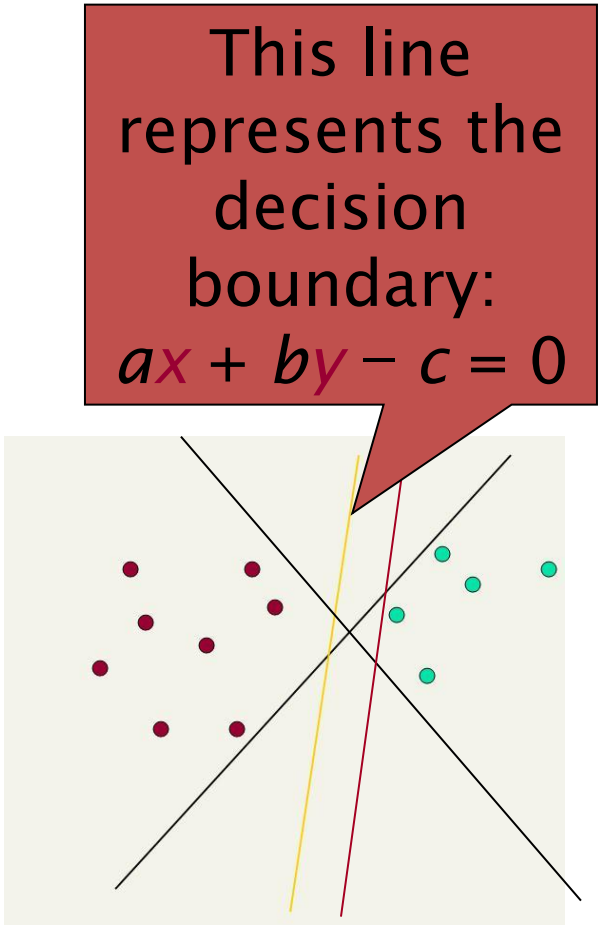
Lots of possible solutions for *a*, *b*, *c*.

Some methods find a separating hyperplane, but not the optimal one [according to some criterion of expected goodness]

- E.g., perceptron

Support Vector Machine (SVM) finds an optimal* solution.

- Maximizes the distance between the hyperplane and the "difficult points" close to decision boundary
- One intuition: if there are no points near the decision surface, then there are no very uncertain classification decisions



Support Vector Machine (SVM)

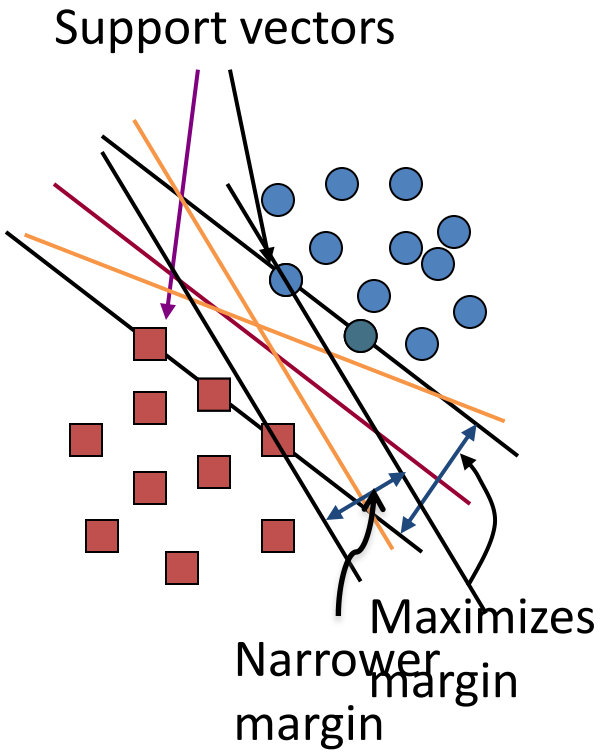
SVMs maximize the *margin* around the separating hyperplane.

- A.k.a. large margin classifiers

The decision function is fully specified by a subset of training samples, *the support vectors*.

Solving SVMs is a *quadratic programming* problem

Seen by many as the most successful current text classification method*



*but other discriminative methods often perform very similarly

Maximum Margin: Formalization

w: decision hyperplane normal vector

$\mathbf{x}_i$: data point i

y_i: class of data point i (+1 or -1) NB: Not 1/0

Classifier is: $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T\mathbf{x}_i + b)$

Functional margin of $\mathbf{x}_i$ is: $y_i(\mathbf{w}^T\mathbf{x}_i + b)$

- But note that we can increase this margin simply by scaling **w**, **b**….

Functional margin of dataset is twice the minimum functional margin for any point

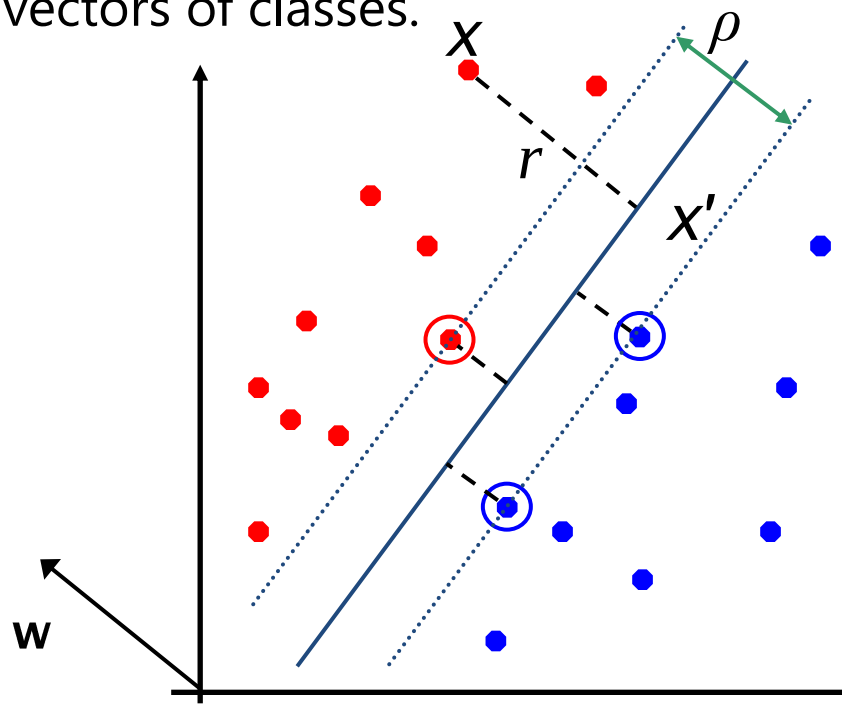
- The factor of 2 comes from measuring the whole width of the margin

Geometric Margin

Distance from example to the separator is $r = y\dfrac{\mathbf{w}^T\mathbf{x}+b}{\|\mathbf{w}\|}$

Examples closest to the hyperplane are ***support vectors***.

Margin ρ of the separator is the width of separation between support vectors of classes.



Derivation of finding *r:*
Dotted line **x'**−**x** is perpendicular to decision boundary so parallel to **w**.
Unit vector is **w**/|**w**|, so line is r**w**/|**w**|.
x' = **x** – yr**w**/|**w**|.
x' satisfies $\mathbf{w}^T\mathbf{x}'+b = 0$.
So $\mathbf{w}^T(\mathbf{x} - yr\mathbf{w}/|\mathbf{w}|) + b = 0$
Recall that $|\mathbf{w}| = \sqrt{\mathbf{w}^T\mathbf{w}}$.
So $\mathbf{w}^T\mathbf{x} - yr|\mathbf{w}| + b = 0$
So, solving for r gives:
$r = y(\mathbf{w}^T\mathbf{x} + b)/|\mathbf{w}|$

Linear SVM Mathematically

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

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

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

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 Support Vector Machine (SVM)

Hyperplane

$\mathbf{w}^T \mathbf{x} + b = 0$

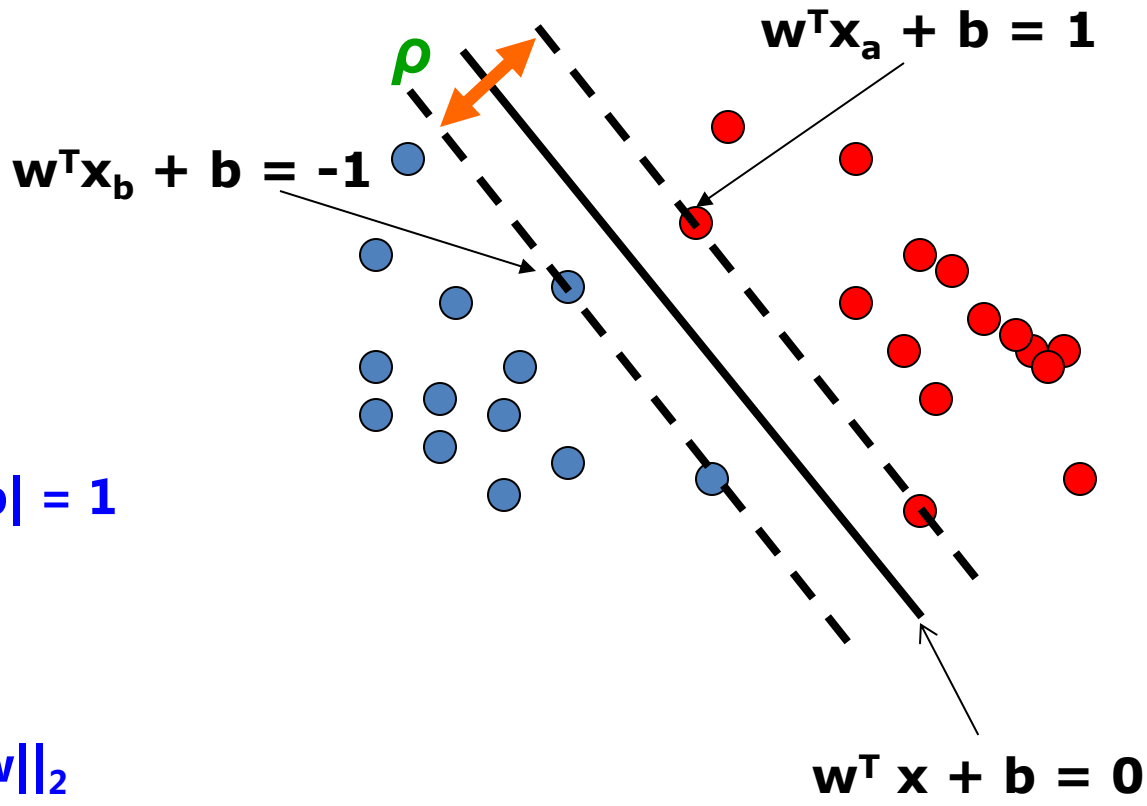
Extra scale constraint:

$\mathbf{min}_{i=1,\ldots,n} |\mathbf{w}^T\mathbf{x}_i + b| = 1$

This implies:

$w^T(x_a - x_b) = 2$

$\rho = ||x_a - x_b||_2 = 2/||w||_2$



Linear SVMs Mathematically (cont.)

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

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

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

The Optimization Problem Solution

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:

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

Notice that it relies on an *inner product* between the test point **x** and the support vectors $\mathbf{x_i}$

- We will return to this later.

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.

Classification with SVMs

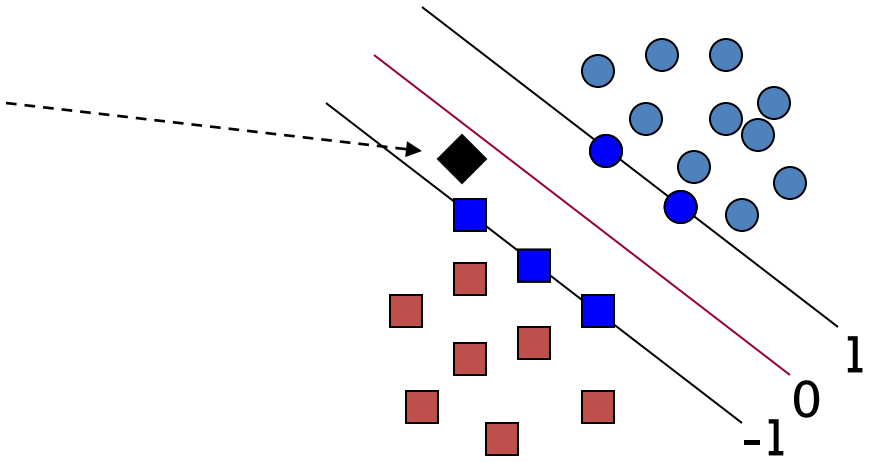
Given a new point **x**, we can score its projection onto the hyperplane normal:

- I.e., compute score: $\mathbf{w}^T\mathbf{x} + b = \Sigma\alpha_i y_i \mathbf{x_i}^T\mathbf{x} + b$
  - Decide class based on whether < or > 0
- Can set confidence threshold *t*.

Score > *t*: yes

Score < -*t*: no

Else: don't know



Linear SVMs: Summary

The classifier is a *separating hyperplane*.

The most "important" training points are the support vectors; they define the hyperplane.

Quadratic optimization algorithms can identify which training points $\mathbf{x_i}$ are support vectors with non-zero Lagrangian multipliers α_i.

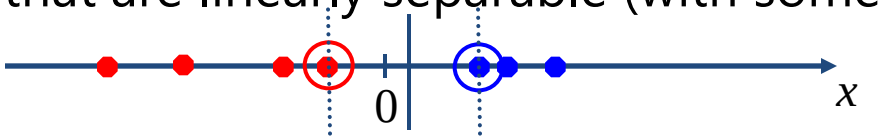
Both in the dual formulation of the problem and in the solution, training points appear only inside inner products:

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) $0 \le \alpha_i \le C$ for all α_i

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

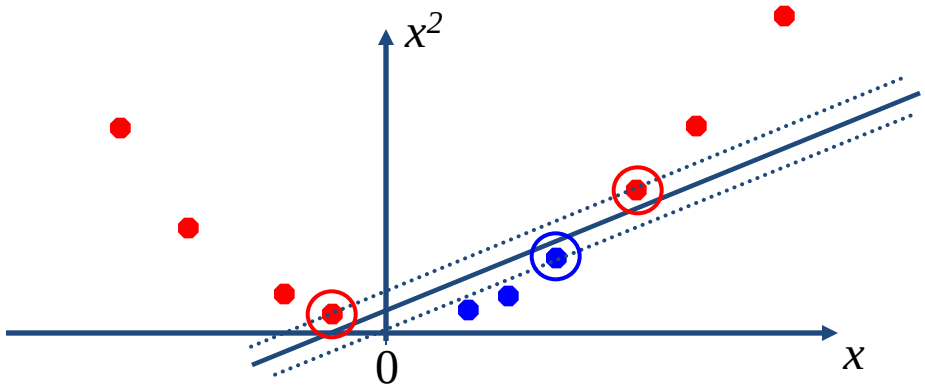
Non-Linear SVMs

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



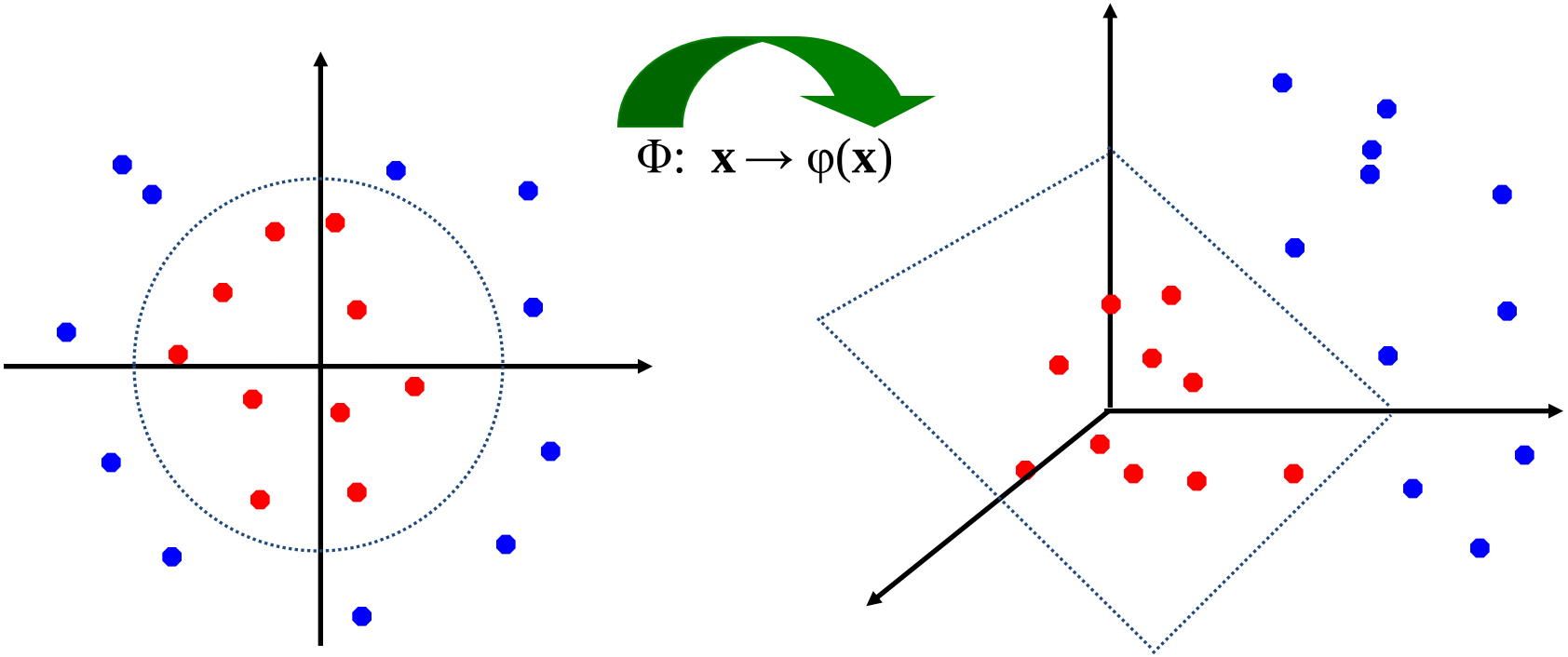
But what are we going to do if the dataset is just too hard?

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:



The "Kernel Trick"

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in \text{SV}} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

- No need to know this mapping explicitly, because we only use the dot product of feature vectors.

- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Unattainable Decision Boundary in 2D

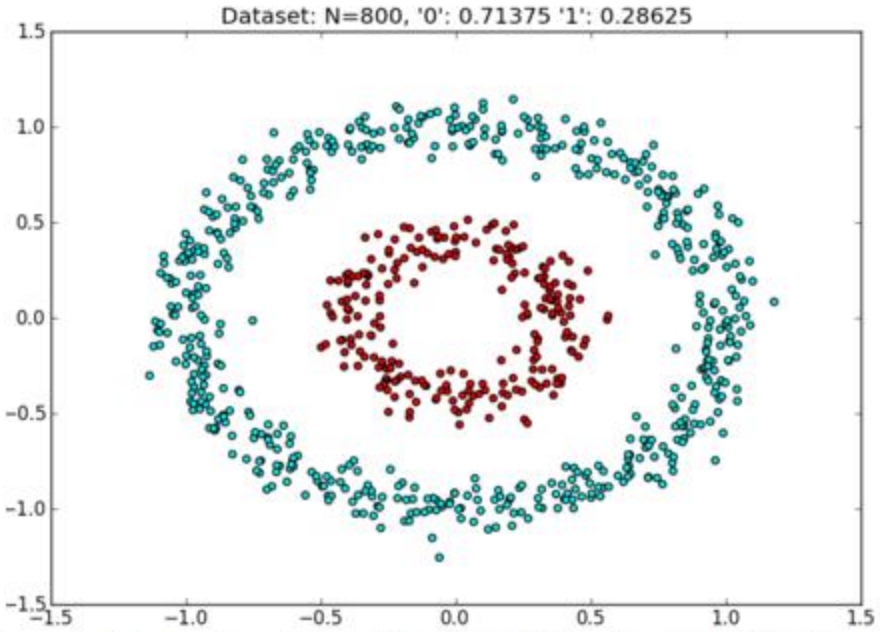
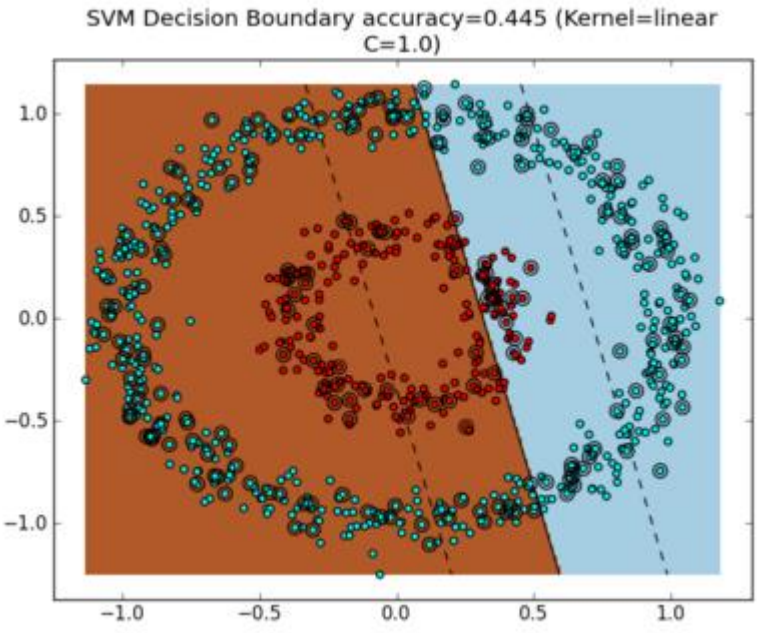


Figure 3: A two-class dataset that is not linearly separable. The outer ring (cyan) is class '0', while the inner ring (red) is class '1'.



http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

The Kernel Trick – Adding a Third Dimension

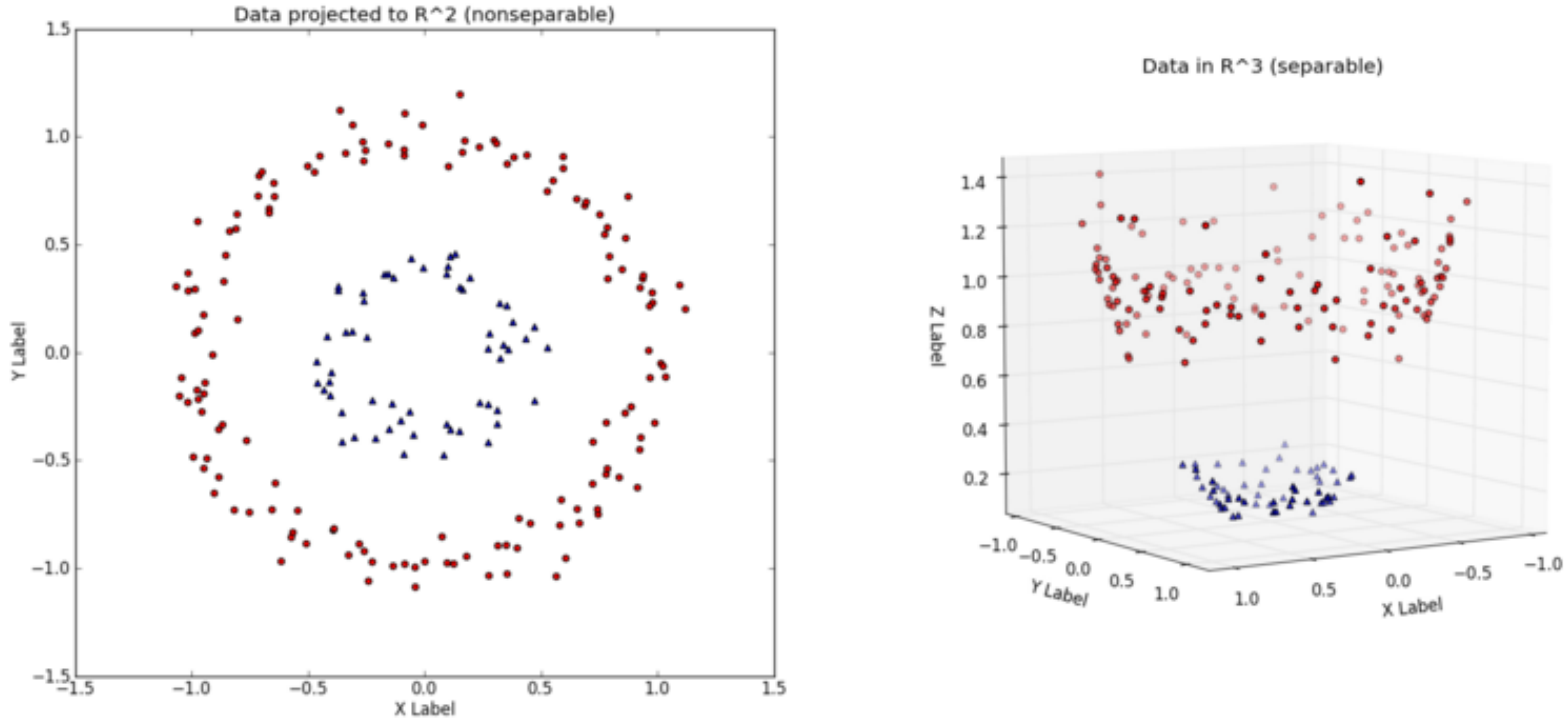


Figure 5: (Left) A dataset in $\mathbb{R}^2$, not linearly separable. (Right) The same dataset transformed by the transformation: $[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2]$.

http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

Decision Boundary in 3D and Projected Down into 2D

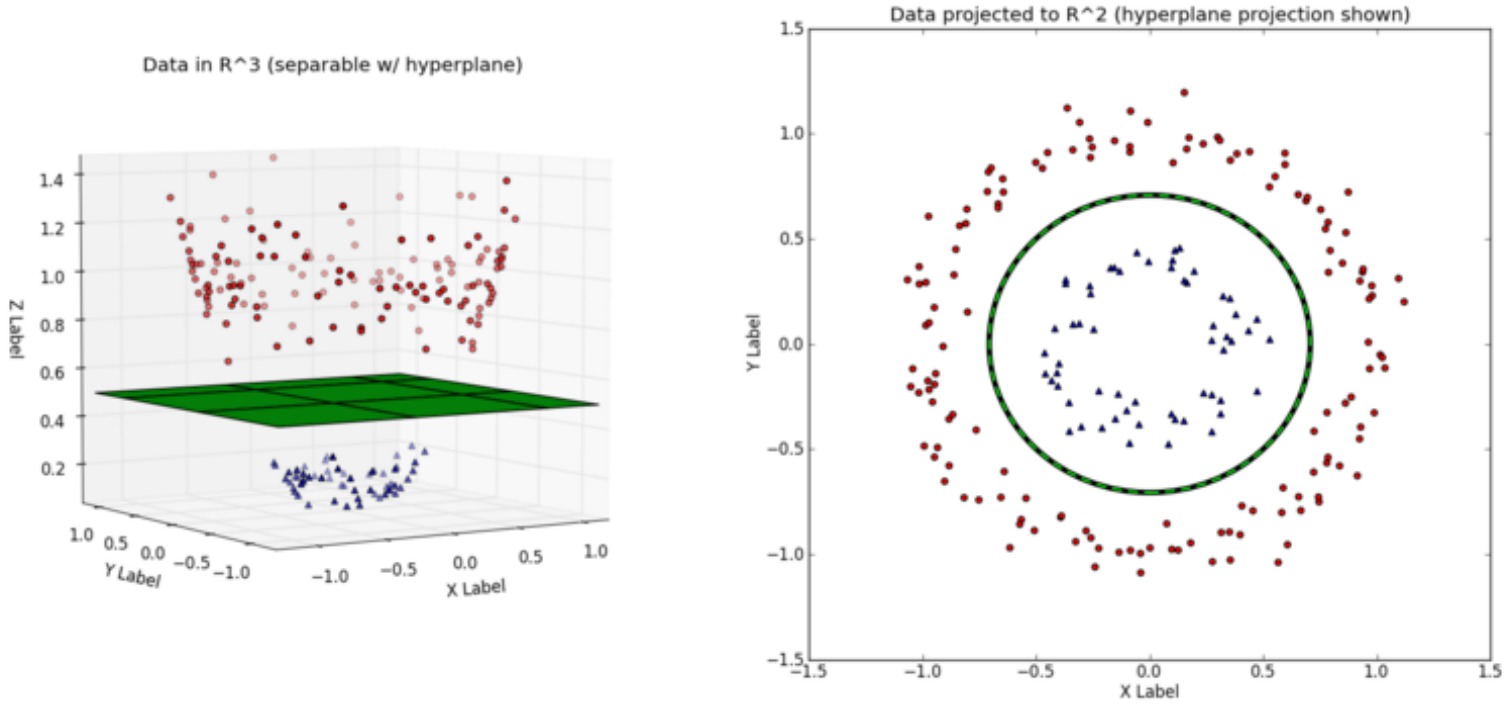


Figure 6: (Left) The decision boundary $\vec{w}$ shown to be linear in $\mathbb{R}^3$. (Right) The decision boundary $\vec{w}$, when transformed back to $\mathbb{R}^2$, is nonlinear.

http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

The "Kernel Trick" – Derivation

- The linear classifier relies on an inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i,\mathbf{x}_j)= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$$

- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.
- Example:

  2-dimensional vectors $\mathbf{x}=[x_1 \;\; x_2]$; let $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^T\mathbf{x}_j)^2$,

  Need to show that $K(\mathbf{x}_i,\mathbf{x}_j)= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$:

  $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^T\mathbf{x}_j)^2 = 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}=$

  $= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\, x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}x_{i1} \;\; \sqrt{2}x_{i2}]^T [1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}]$

  $= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ where $\phi(\mathbf{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$

The "Kernel Trick" – Other Kernels

- Examples of commonly-used kernel functions:
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
  - Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right)$$

  - Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

- In general, functions that satisfy *Mercer's condition* can be kernel functions.

Neural Networks

Pros and Cons

Pros

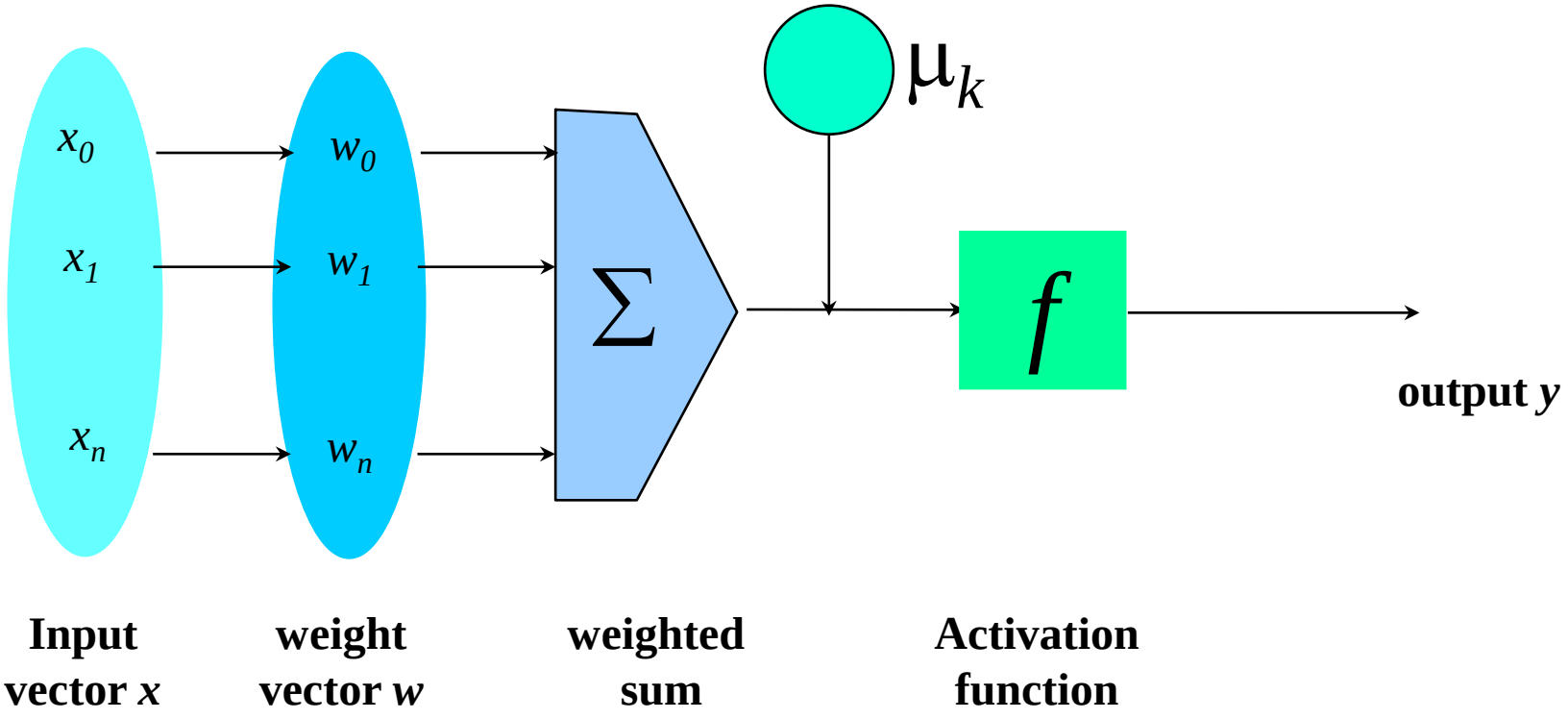
- prediction accuracy is generally high
- robust, works when training examples contain errors
- output may be discrete, real-valued, or a vector of several discrete or real-valued attributes
- fast evaluation of the learned target function

Cons

- long training time
- difficult to understand the learned function (weights)
- not easy to incorporate domain knowledge

Concept

The *n*-dimensional input vector *x* is mapped into variable *y* by means of the scalar product and a nonlinear function mapping



Training

The ultimate objective of training

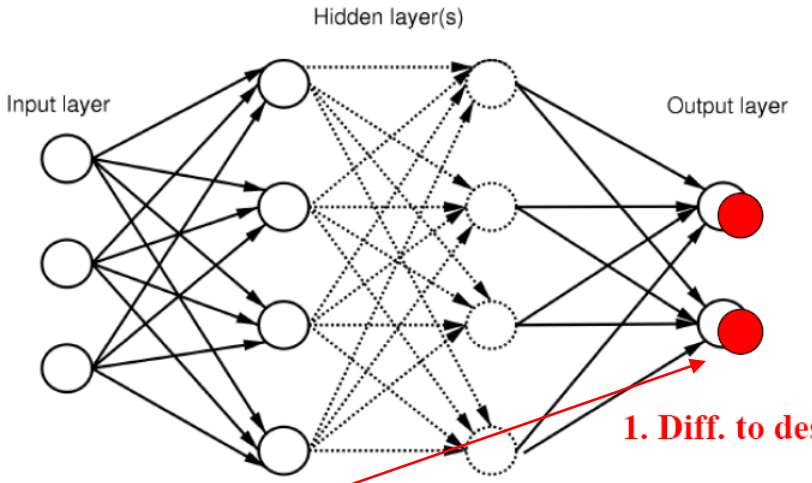
- obtain a set of weights that makes almost all the tuples in the training data classified correctly

Steps

- Initialize weights with random values
- Feed the input tuples into the network one by one
- For each unit
  - compute the net input to the unit as a linear combination of all the inputs to the unit
  - compute the output value using the activation function
  - compute the error
  - update the weights and the bias

Visualization of Backpropagation learning

Backpropagation Learning



Backpropagation Learning

$$E_{out\ i} = d_{out\ i} - out_i$$

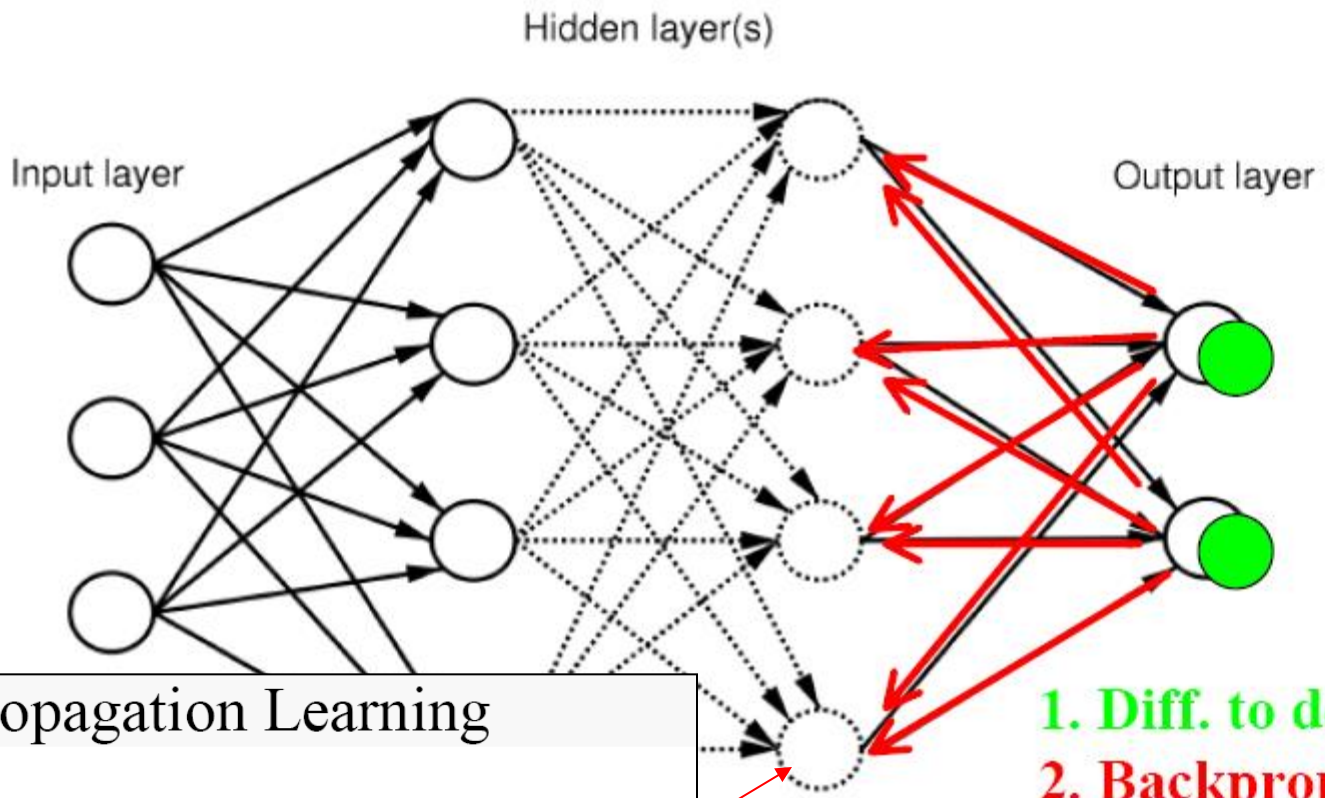
$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

Backprop output layer

Backpropagation Learning



1. Diff. to desired values
2. Backprop output layer

Backpropagation Learning

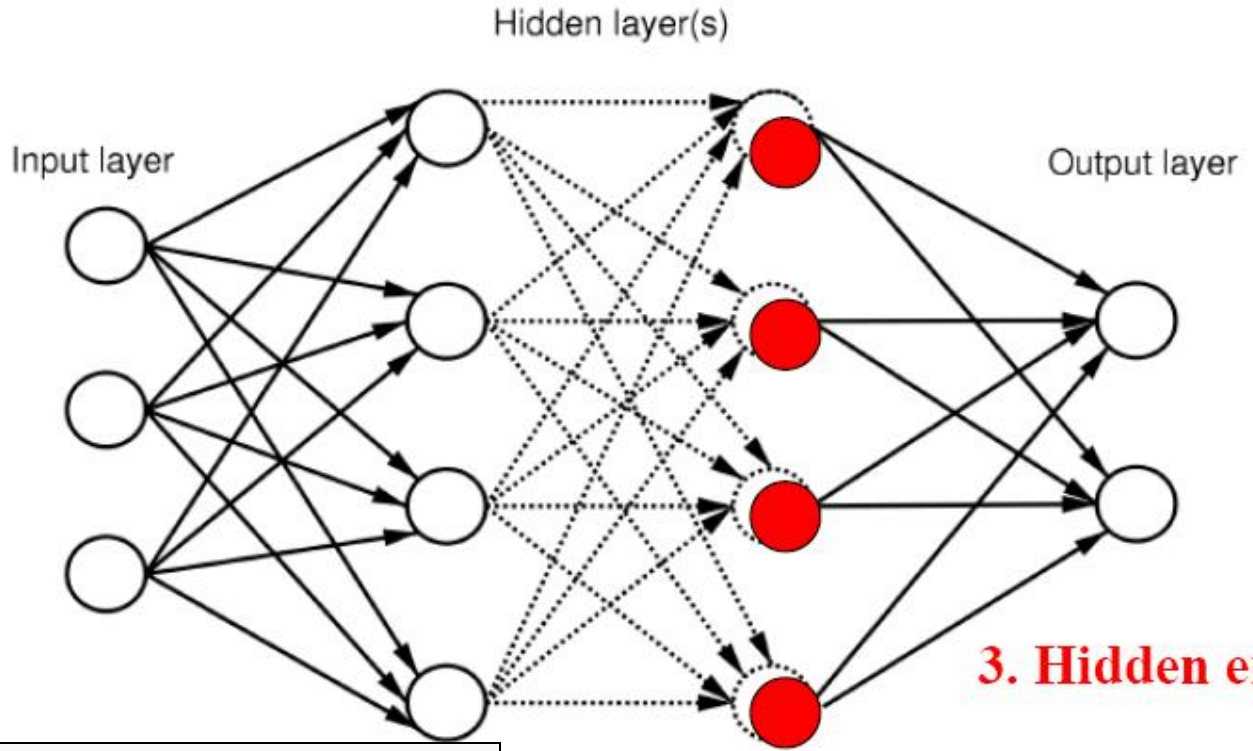
$$E_{out\ i} = d_{out\ i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

Backpropagation Learning



3. Hidden error values

Backpropagation Learning

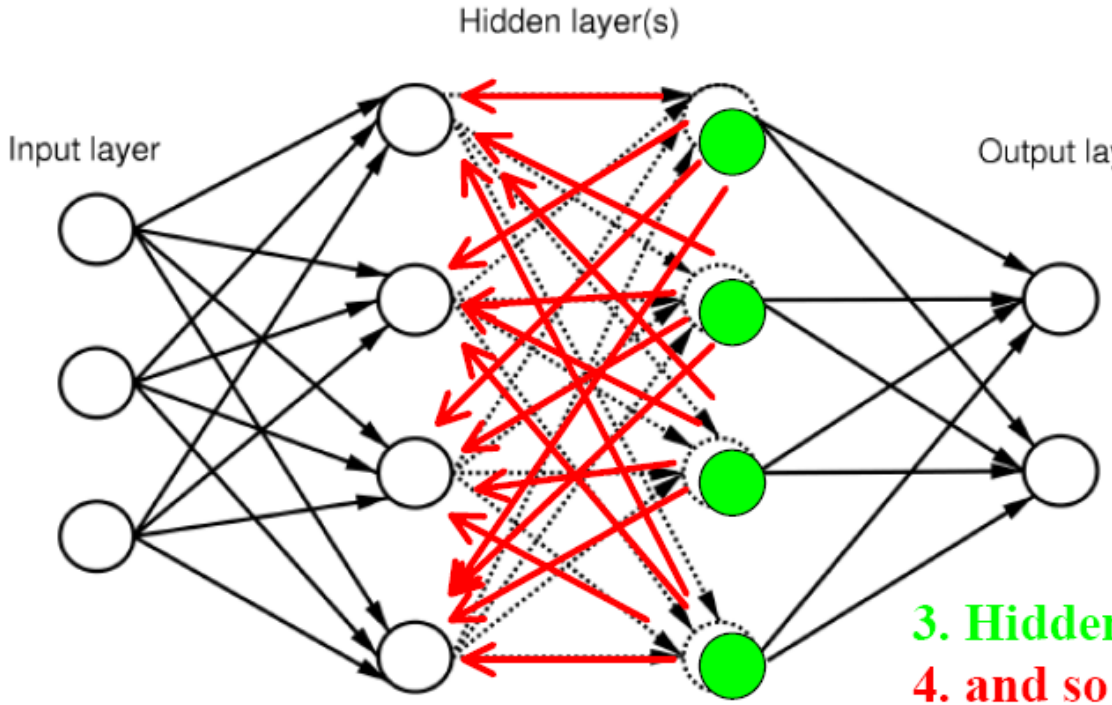
$$E_{out\ i} = d_{out\ i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

Backpropagation Learning



3. Hidden error values

4. and so on …

Backpropagation Learning

$$E_{out\ i} = d_{out\ i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

How Many Hidden Layers?

Usually just one (i.e., a 2-layer net)

How many hidden units in the layer?

- Too few → can't learn
- Too many → poor generalization (overfitting)

How Big A Training Set?

Determine your target error rate, e

Success rate is 1- e

Typical training set approx. n/e, where n is the number of weights in the net

Example:

- e = 0.1, n = 80 weights
- training set size 800
- trained until 95% correct training set classification
- should produce 90% correct classification on testing set (typical)

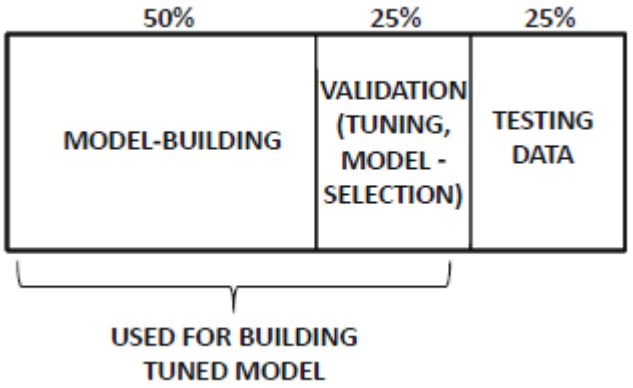
Evaluation

Need to evaluate classifier in terms of

- effectiveness
- comparing different models
- select the best one for a particular data set
- parameter tuning
- ensemble analysis (see later)

Evaluation is different than testing

- do not tune the parameters with the test data



How to Pick the Test Data

Holdout method

- divide the data into training data an test data at ratio 2 to 1
- *stratify* samples: sample each labeled class separately at the same % rate and pool the samples
- this way each class is represented both in training and testing
- will be fairer when classes are unevenly distributed
- do this multiple times

Cross-Validation

- divide the n samples into m bins (m is typically 10)
- train with m-1 bins = (m-1)n/m points, test with 1 bin = n/m points
- repeat this by picking different bins for training and testing
- stratify when classes are unevenly populated
- extreme case is *leave-one-out* cross-validation when m=n

How to Pick the Test Data

Bootstrap method

- training set samples the data *with replacement* and has size n
- so some samples may be duplicates and data might be missing
- testing set uses all data and so a sample may be contained in both the training and the test data
- hence testing will yield a highly optimistic score (the other methods were more pessimistic)

- probability a specific data point is not included in a sample is $(1-1/n)$
- probability a specific data point is not included in n samples is $(1-1/n)^n$
- for large n, this expression evaluates to about 1/e
- e is the base of the natural logarithm
- the fraction of the labeled data points included at least once in the training data is therefore $1-1/e \approx 0.632$

- note: bootstrap is best for small data

Multi-Class Learning

What to do when there are multiple classes, k?

- some classifiers like SVM are only defined for two classes

Strategy 1: One-against-the-rest

- form k classifiers, one for each class against all others
- classify a query point for each class j
- if class j wins then it gets a point, else all other classes get a vote
- the class with the highest overall #votes wins
- optionally scale each vote by the classifier's score and sum

Strategy 2: One-against-one

- train $k \cdot (k-1)/2$ pairwise classifiers
- class with highest number of votes wins
- needs more comparisons
- training will be similar since the data is smaller per classifier

Rare Class Learning

Rare classes are usually more expensive when they are missed

- fraud in credit card uses
- highly profitable stock
- but vanilla classifiers usually return the normal class
- need a strategies for better rare class detection
- incorporate the cost of misclassifying a rare class into the classifier

These methods bias the classifier towards the rare class

- that is why you get these annoying credit card blocks

Rare Class Learning

Example reweighting

- reweigh training examples according to class misclassification costs

Example resampling

- oversample rare classes in proportion to class cost
- or, undersample frequent classes and keep all rare examples
- the latter is more efficient for training

SMOTE algorithm – copes better with bias

- introduces synthetic oversampling
- instead of using only the rare examples which adds too much bias
- generate synthetic data examples on the line segment connecting each minority example to its nearest normal class neighbor

Scalable Classification

Size of the data can lead to significant computational problems in the training phase

- create multiple models based on smaller data and merge them
- eliminate non-important data not relevant to the model early

Decision trees

- *Bootstrapped Optimistic Algorithm for Tree* construction (BOAT)
- uses b sets of bootstrapped samples
- constructs b decision trees
- checks their splitting criteria and merge them

Scalable SVM

- keep reducing the data during iterative decision boundary construction
- only keep those somewhat close to the current boundary

Ensemble Methods

Motivated by

- different classifiers may make different predictions on test instances due to the specific characteristics of the classifier
- different classifiers may also have varied sensitivity to the random artifacts in the training data

Ensemble methods increase the prediction accuracy by combining the results from multiple classifiers

- each of the k classifiers has a higher bias
- but for the overall classifier the bias as well as the variance σ^2 is reduced to σ^2/k

Ensemble Methods – Overall Idea

Algorithm *EnsembleClassify*(Training Data Set: $\mathcal{D}$
Base Algorithms: $\mathcal{A}_1 \ldots \mathcal{A}_r$, Test Instances: $\mathcal{T}$)
begin
$j = 1$;
repeat
Select an algorithm $\mathcal{Q}_j$ from $\mathcal{A}_1 \ldots \mathcal{A}_r$;
Create a new training data set $f_j(\mathcal{D})$ from $\mathcal{D}$;
Apply $\mathcal{Q}_j$ to $f_j(\mathcal{D})$ to learn model $\mathcal{M}_j$;
$j = j + 1$;
until(termination);
report labels of each $T \in \mathcal{T}$ based on combination of
predictions from all learned models $\mathcal{M}_j$;
end

Bagging

Takes k different bootstrapped samples each of size n

- drawn independently

The classifier is trained on each of them

- for a given test instance, the predicted class label is reported by the ensemble as the *majority vote* of the different classifiers

Reduces model variance

- but does not reduce the model bias

Random forests

- generalization of the basic bagging method applied to decision trees.
- defined as an ensemble of decision trees each generated by bootstrapped samples

Boosting

Create a sequence of classifiers with changing sample weights

- modify the weights for a future classifier based on classifier performance of previous classifiers

→ future models constructed are dependent on the results from previous models

Use the same algorithm A on a weighted training data set

- incorrectly classified instances in future iterations increase the relative weight of these instances in the training of the next classifier
- hypothesize that the errors in these misclassified instances are caused by classifier bias

→ increasing the instance weight of misclassified instances will result in a new classifier that corrects for the bias on these particular instances

ADABOOST

Well known boosting algorithm

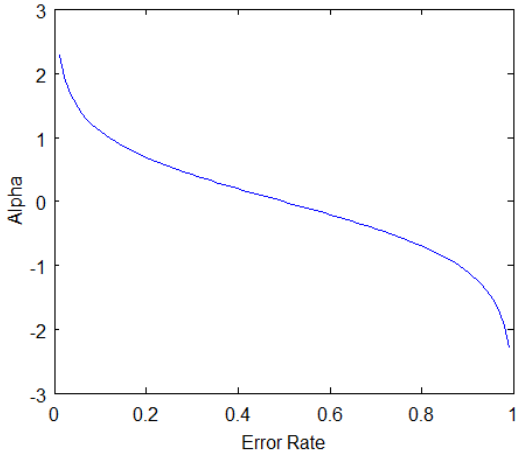
- puts forward the idea of *weak classifier*

Algorithm *AdaBoost*(Data Set: $\mathcal{D}$, Base Classifier: $\mathcal{A}$, Maximum Rounds: T)
begin
  $t = 0$;
  for each i initialize $W_1(i) = 1/n$;
  repeat
    $t = t + 1$;
    Determine weighted error rate ϵ_t on $\mathcal{D}$ when base algorithm $\mathcal{A}$
      is applied to weighted data set with weights $W_t(\cdot)$;
    $\alpha_t = \frac{1}{2}\log_e((1-\epsilon_t)/\epsilon_t)$;
    for each misclassified $\overline{X_i} \in \mathcal{D}$ **do** $W_{t+1}(i) = W_t(i)e^{\alpha_t}$;
      else (correctly classified instance) **do** $W_{t+1}(i) = W_t(i)e^{-\alpha_t}$;
    for each instance $\overline{X_i}$ **do** normalize $W_{t+1}(i) = W_{t+1}(i)/[\sum_{j=1}^{n} W_{t+1}(j)]$;
  until $((t \geq T)$ OR $(\epsilon_t = 0)$ OR $(\epsilon_t \geq 0.5))$;
  Use ensemble components with weights α_t for test instance classification;
end

random chance

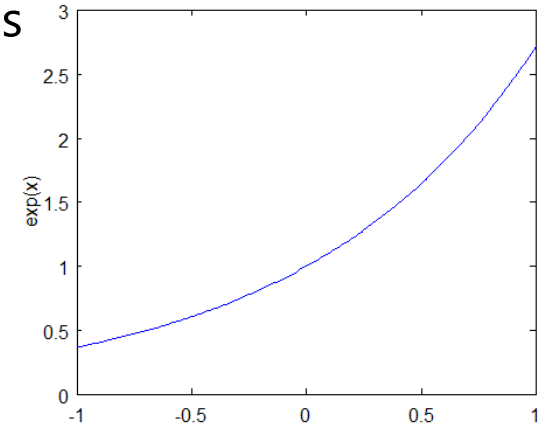
ε^t is the fraction of incorrectly predicted training instances

AdaBoost Weights



The weight function α_τ

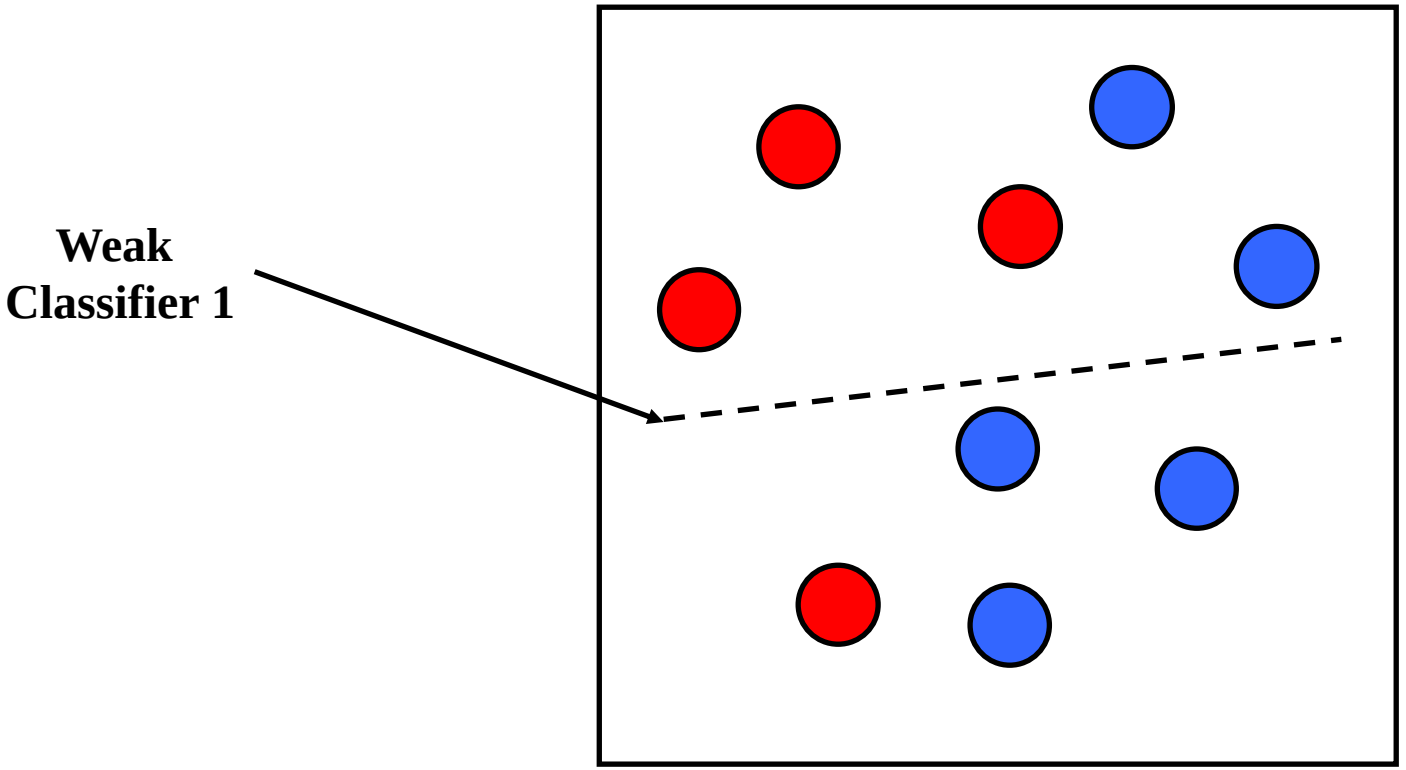
- The classifier weight grows exponentially as the error approaches 0. Better classifiers are given exponentially more weight.
- The classifier weight is zero if the error rate is 0.5. A classifier with 50% accuracy is no better than random guessing, so we ignore it.
- The classifier weight grows exponentially negative as the error approaches 1. We give a negative weight to classifiers with worse than 50% accuracy. "Whatever that classifier says, do the opposite!".
- use these weights for subsequent classifications



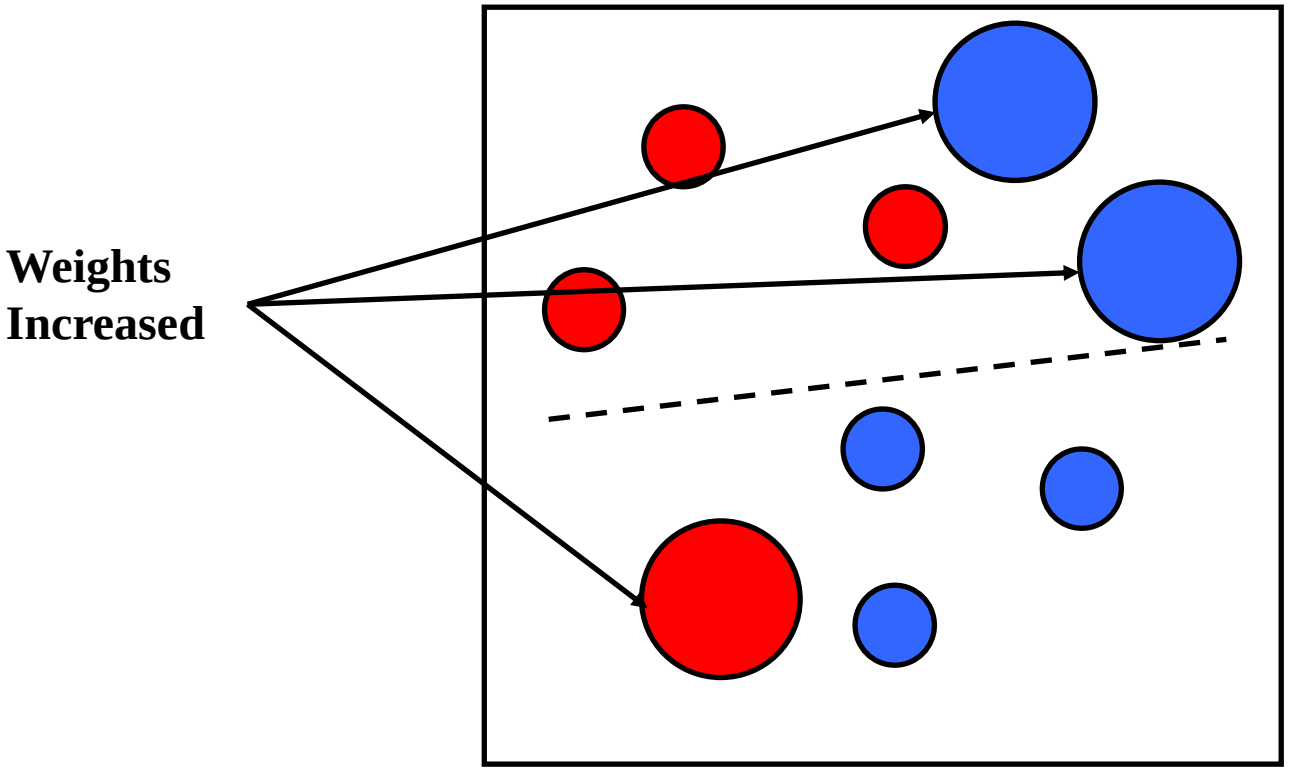
Use the exponential function

- positive exponent for misclassified samples
- negative exponent for correctly classified samples

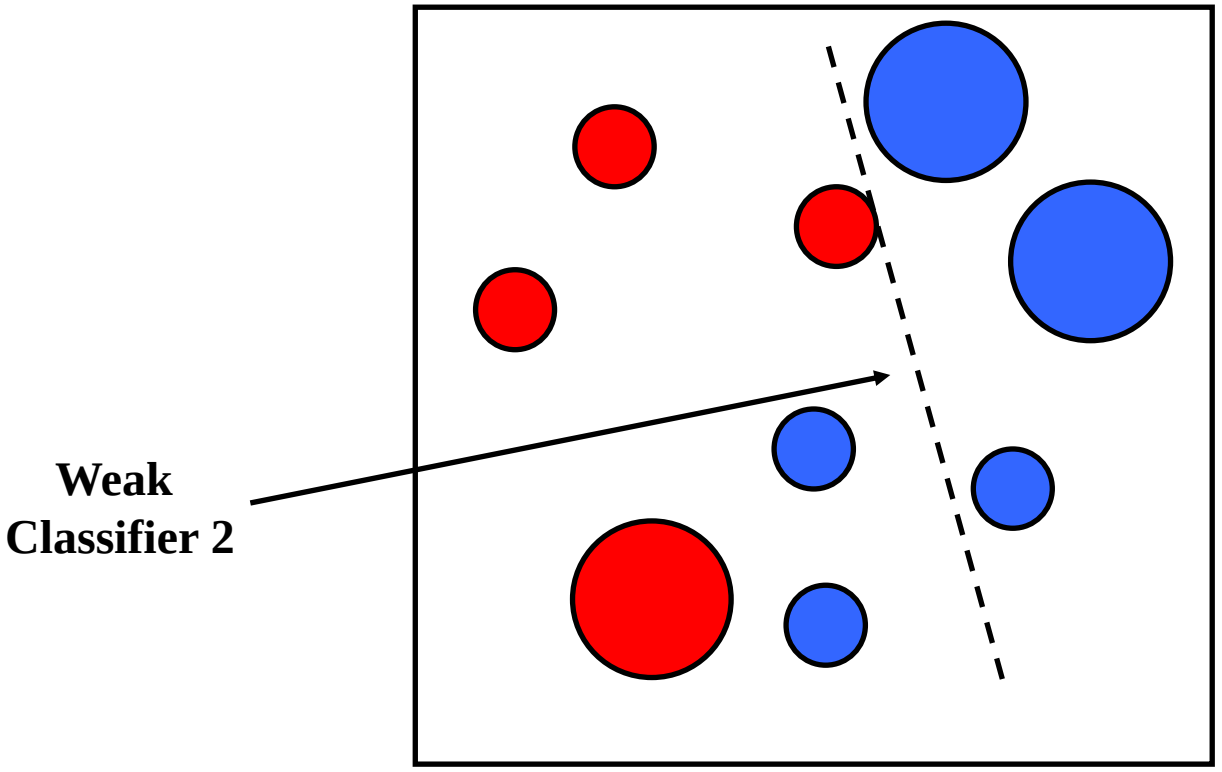
Boosting Illustration



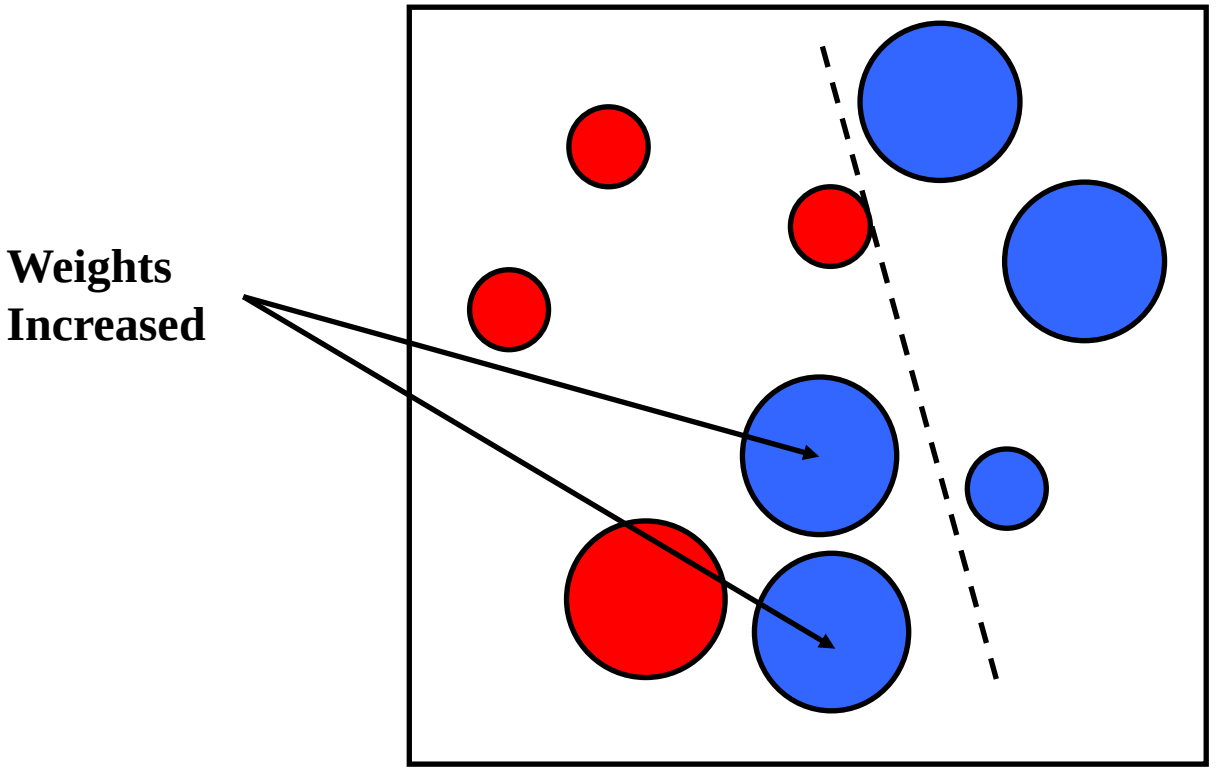
Boosting Illustration



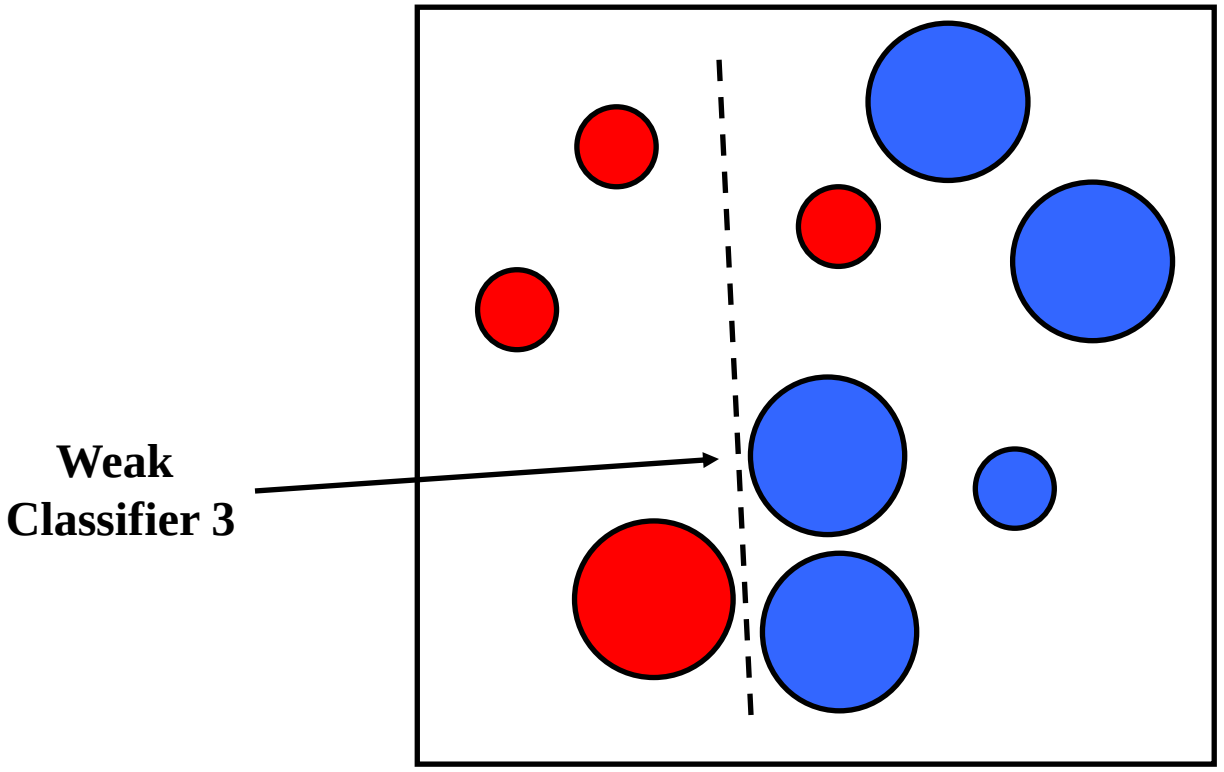
Boosting Illustration



Boosting Illustration

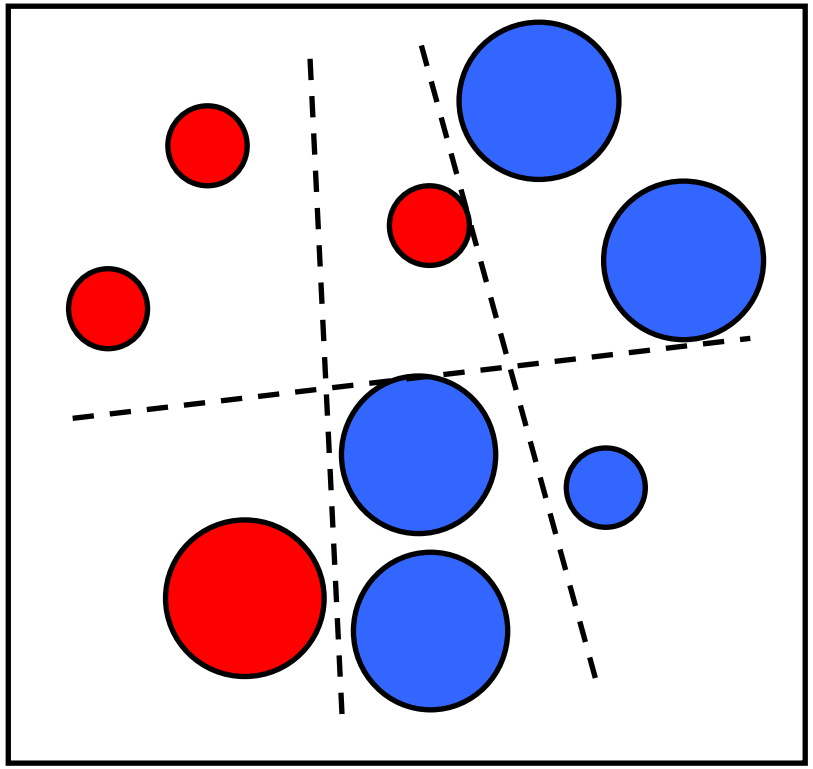


Boosting Illustration



Boosting Illustration

Final classifier is a combination of weak classifiers



Some Adaboost Decision Boundary Examples