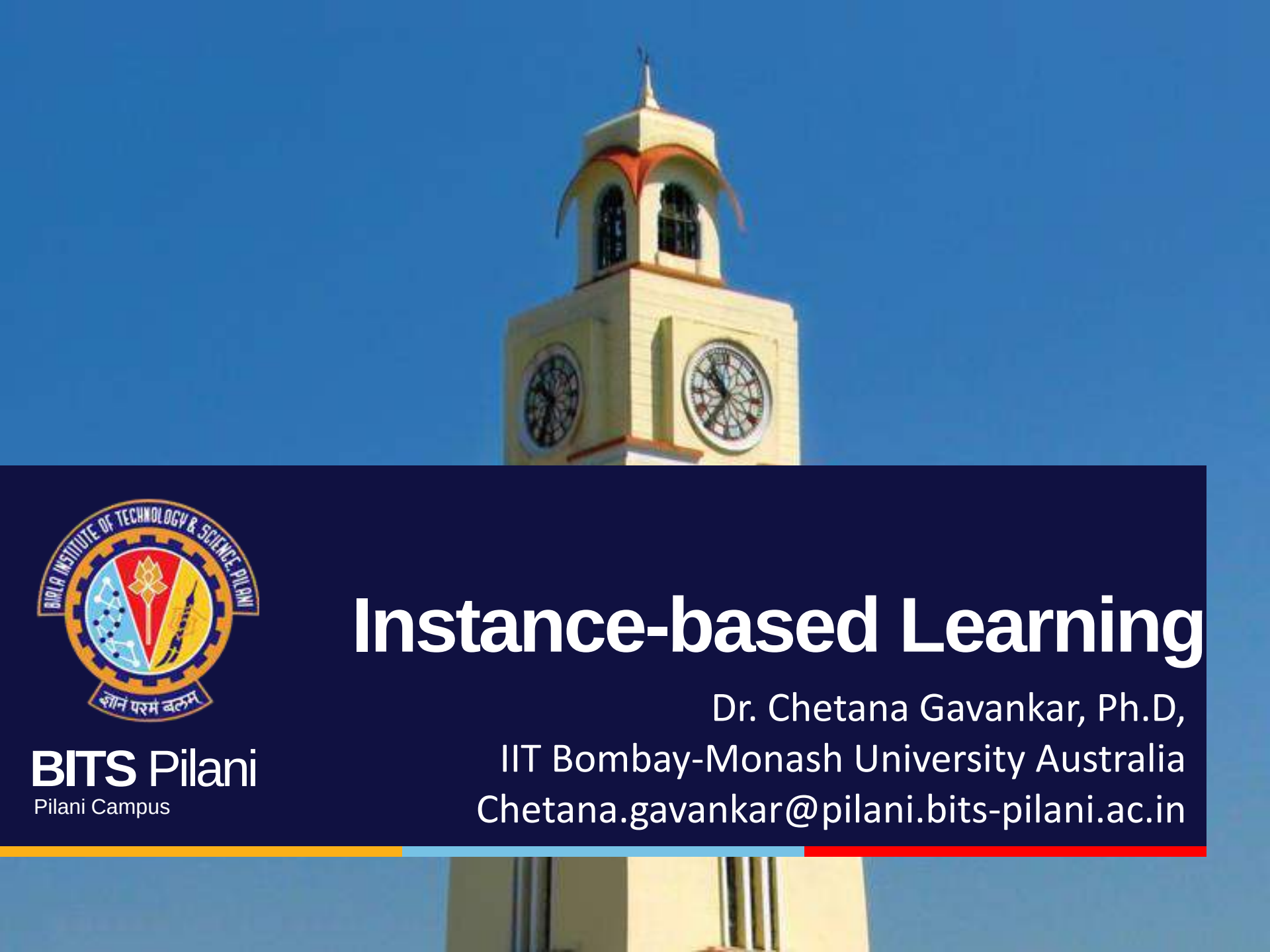


Instance-based Learning

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Topics to be covered

- Instance based learning
- K-Nearest Neighbour Learning
- Locally Weighted Regression (LWR) Learning
- Radial Basis Functions

Instance Based Learning

Key idea: just store all training examples $<x_i, f(x_i)>$

Nearest neighbor:

- Given query instance x_q, first locate nearest training example x_n, then estimate $f^*(x_q)=f(x_n)$

K-nearest neighbor:

- Given x_q, take vote among its k nearest neighbors (if discrete-valued target function)
- Take mean of f values of k nearest neighbors (if real-valued) $f^*(x_q)=\sum_{i=1}^{k} f(x_i)/k$

k-Nearest Neighbor Classifier

- Nearest Neighbour classifier is an instance based classifier
- 'lazy learning', as learning is postponed until a new instance is encountered
- Constructs a local approximation to the target function, applicable in the neighbourhood of new instance
- Suitable in cases where target function is complex over the entire input space, but easily describable in local approximations
- Real world applications found in recommendation systems (amazon).
- Caveat is the high cost of classification, which happens at the time of processing rather than before hand (there's no training phase)

When to Consider Nearest Neighbors

- Instances map to points in R^N
- Less than 20 attributes per instance
- Lots of training data

Advantages:

- Training is very fast
- Learn complex target functions
- Do not lose information

Disadvantages:

- Slow at query time
- Easily fooled by irrelevant attributes

k-Nearest Neighbor Classifier

- Considers all instances as members of n-dimensional space
- Nearest neighbours of an instance is determined based on Euclidean distance
- Distance between two n-dimensional instances x_i and x_j is given by:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

- For NN classifier, target function can be discrete or continuous

Discrete and Continuous-valued function

- **discrete-valued target function:**
  - $f : \Re^n \rightarrow V$ where V is the finite set $\{v_1, v_2, ..., v_s\}$
  - the target function value is the most common value among the k nearest training examples

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = (a == b)$

- **continuous-valued target function:**
  - algorithm has to calculate the mean value instead of the most common value
  - $f : \Re^n \rightarrow \Re$

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

k-Nearest Neighbor Classifier

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

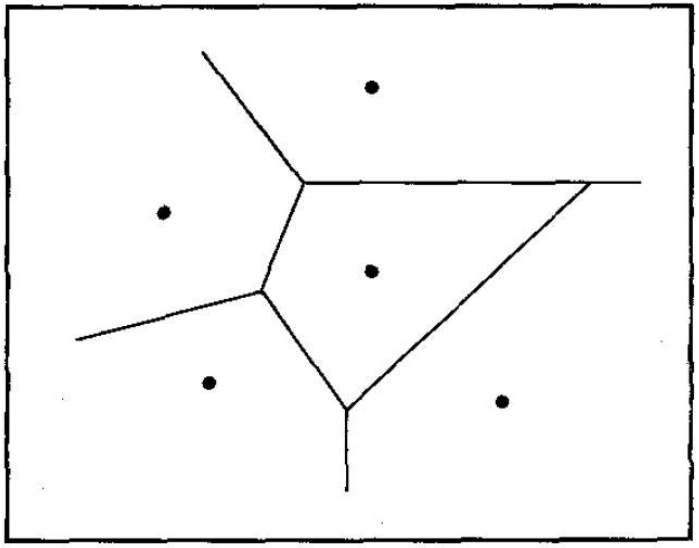
- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

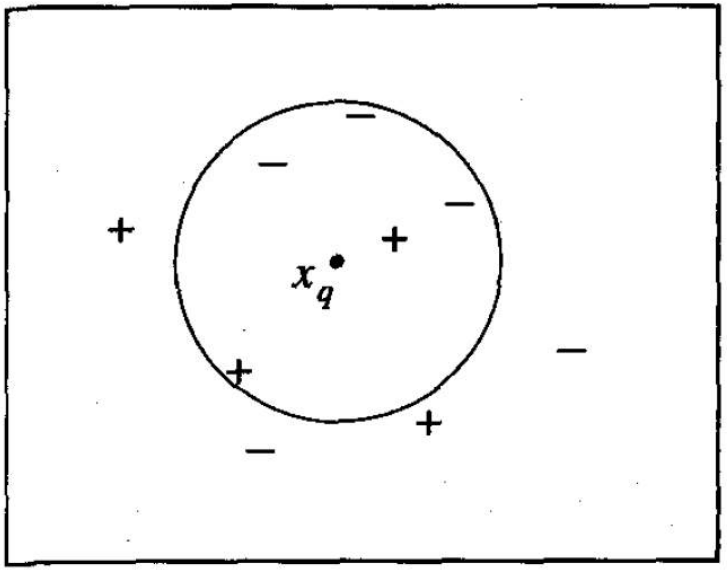
where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

* It can be used for Regression as well.

k-NN examples



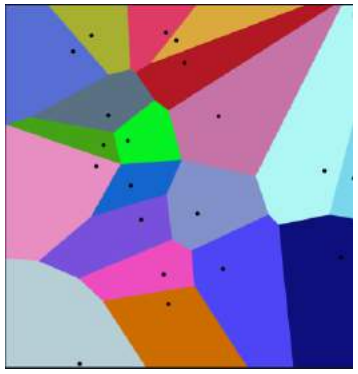
K=1



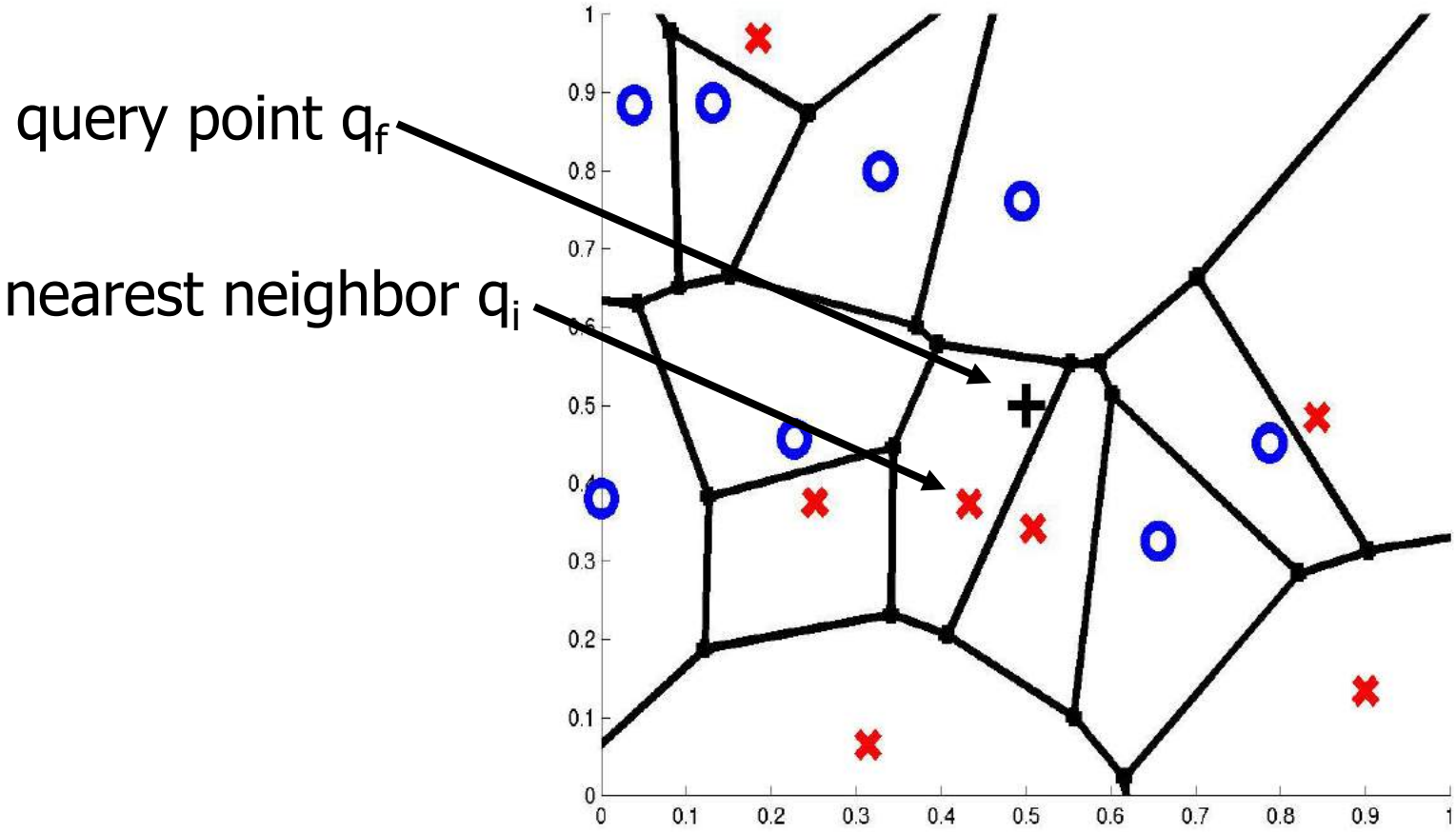
K=5

Voronoi Diagram

- It is a partition of a plane into regions close to each of a given set of objects.



Voronoi Diagram

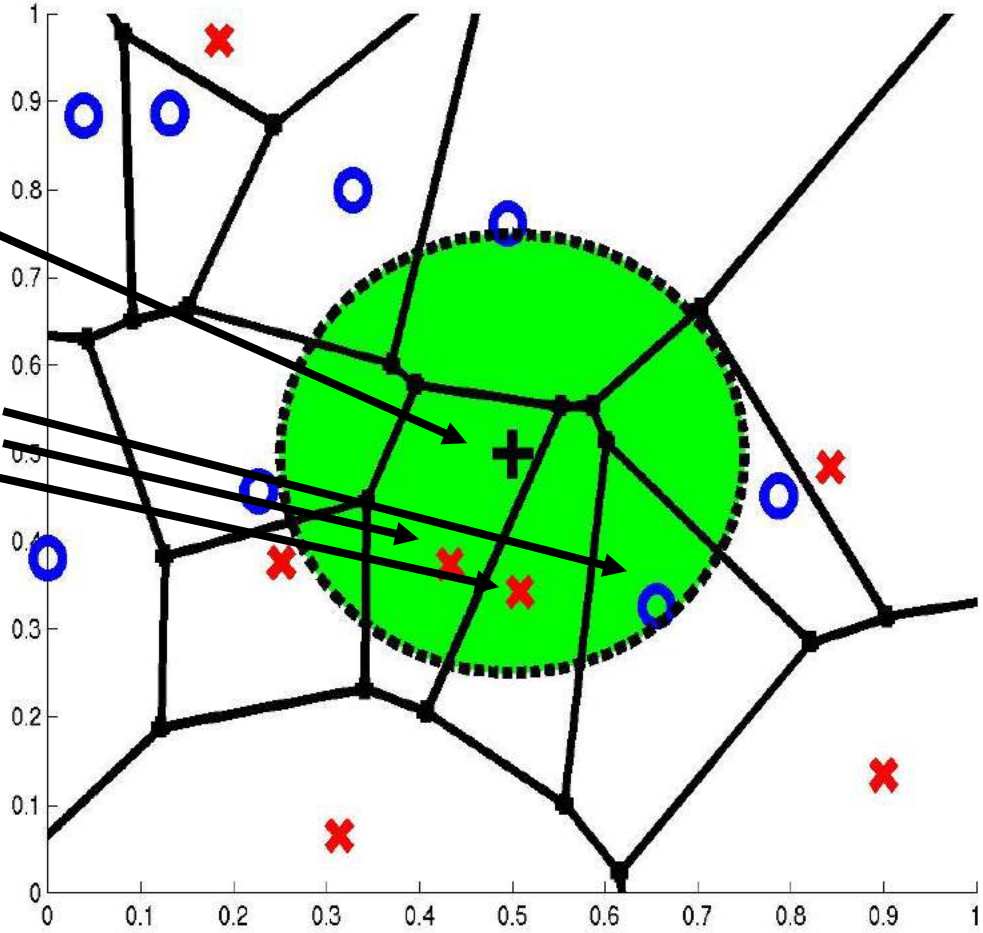


3-Nearest Neighbors

query point q_f

3 nearest neighbors

2x,1o

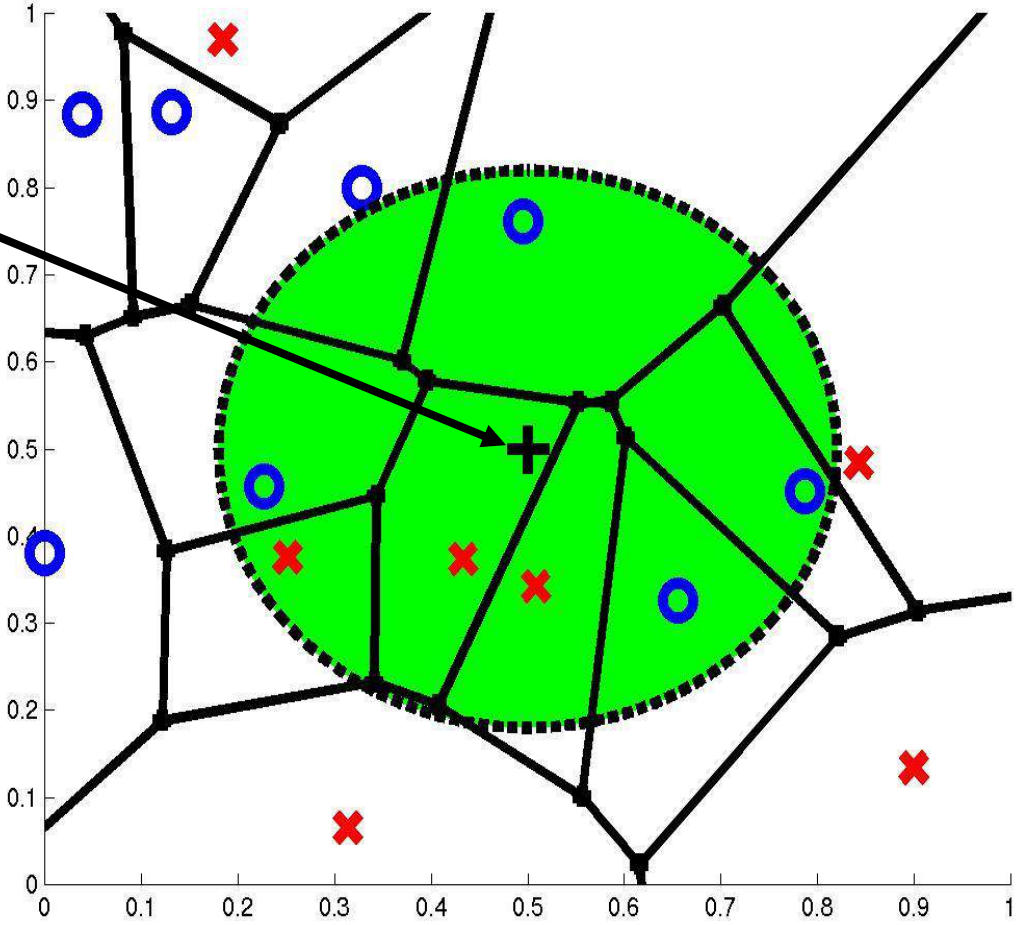


7-Nearest Neighbors

query point q_f

7 nearest neighbors

3x,4o



Distance weighted nearest neighbor

- contribution of each of the k nearest neighbors is weighted accorded to their distance to x_q
  - **discrete-valued target functions**

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where $w_i \equiv \frac{1}{d(x_q, x_i)^2}$ and $\hat{f}(x_q) = f(x_i)$ if $x_q = x_i$

  - **continuous-valued target function:**

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

Distance Weighted k-NN

Give more weight to neighbors closer to the query point

– $f^*(x_q) = \Sigma_{i=1}^{k} w_i f(x_i) / \Sigma_{i=1}^{k} w_i$
; $w_i = K(d(x_q, x_i))$

and

– $d(x_q, x_i)$ is the distance between x_q and x_i

Variation: Instead of only k-nearest neighbors use all training examples (Shepard's method)

Distance Weighted Average

Weighting the data

- $f^*(x_q) = \Sigma_i f(x_i) K(d(x_i,x_q))/ \Sigma_i K(d(x_i,x_q))$

Relevance of a data point $(x_i,f(x_i))$ is measured by calculating the distance $d(x_i,x_q)$ between the query x_q and the input vector x_i

Weighting the error criterion

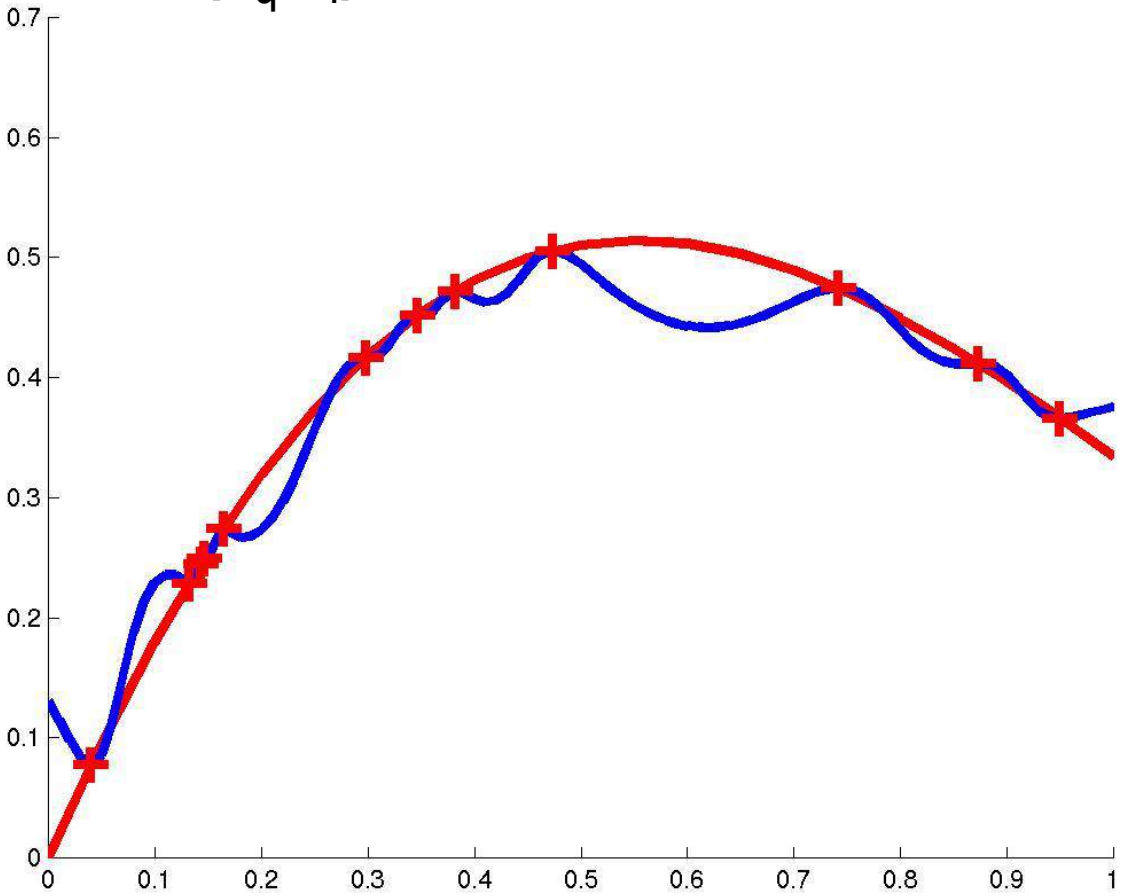
- $E(x_q) = \Sigma_i (f^*(x_q)-f(x_i))^2 K(d(x_i,x_q))$

Best estimate $f^*(x_q)$ will minimize the cost $E(x_q)$, therefore

$$\partial E(x_q)/\partial f^*(x_q)=0$$

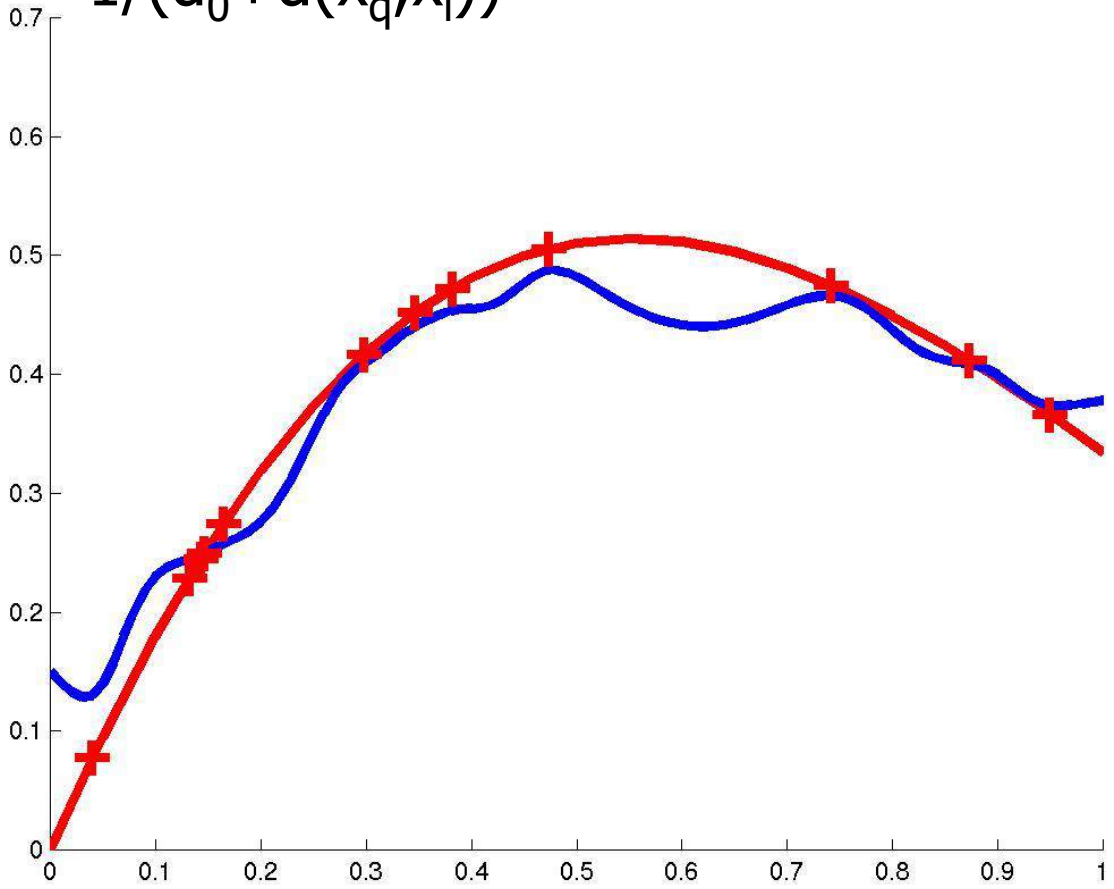
Distance Weighted NN

$K(d(x_q,x_i)) = 1/\ d(x_q,x_i)^2$



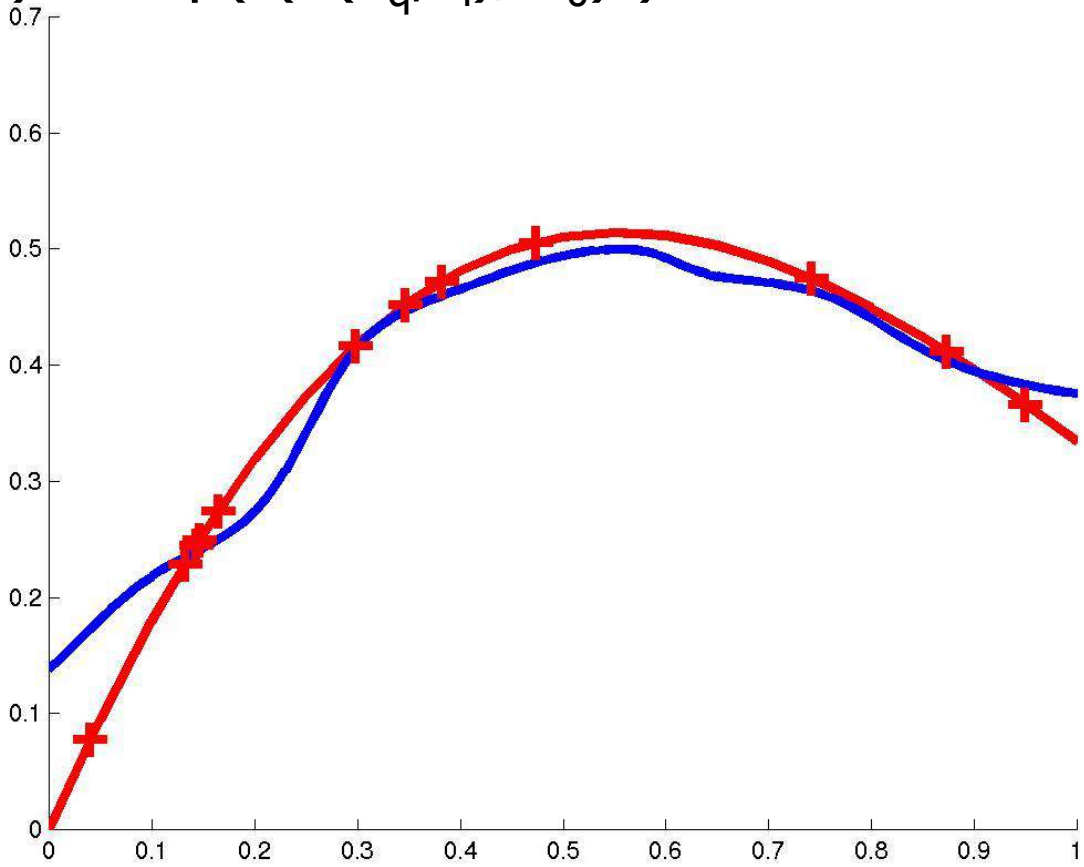
Distance Weighted NN

$$K(d(x_q,x_i)) = 1/(d_0+d(x_q,x_i))^2$$



Distance Weighted NN

$$K(d(x_q,x_i)) = \exp(-(d(x_q,x_i)/\sigma_0)^2)$$



Curse of Dimensionality

Imagine instances described by 20 attributes but only a few are relevant to target function

Curse of dimensionality: nearest neighbor is easily misled when instance space is high-dimensional

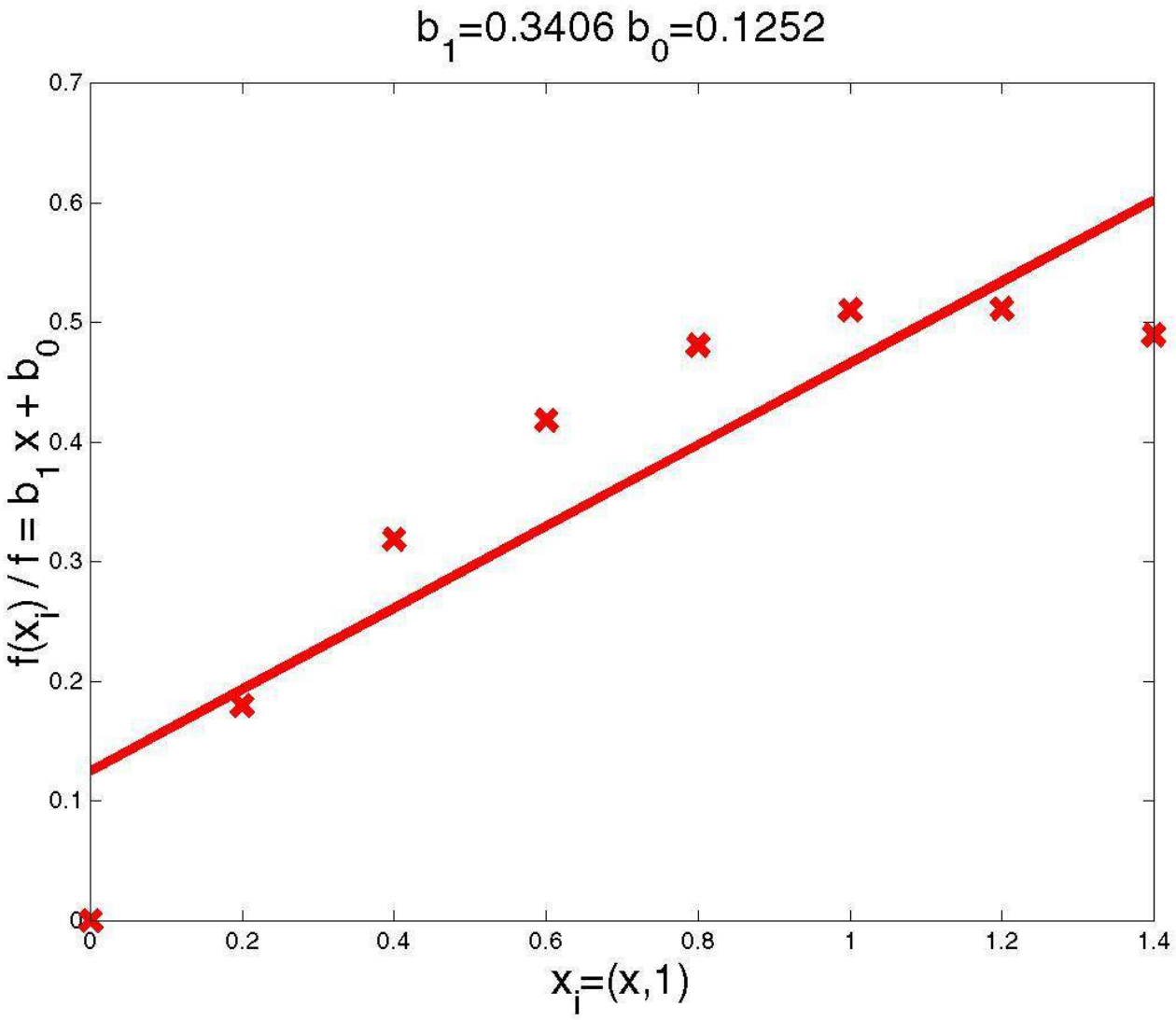
One approach:
Stretch j-th axis by weight z_j, where $z_1,\ldots,z_n$ chosen to minimize prediction error
Use cross-validation to automatically choose weights $z_1,\ldots,z_n$
Note setting z_j to zero eliminates this dimension alltogether (feature subset selection)

Locally Weighted Regression

Locally Weighted Regression

- Locally – Function approximated based on data near query point
- Weighted – Contribution by each training example is weighted by its distance from query point
- Regression- Approximates real-valued target function

Linear Regression Example



Locally weighted regression

- a note on terminology:
  - *Regression* means approximating a real-valued target function
  - *Residual* is the error $\hat{f}(x) - f(x)$ in approximating the target function
  - *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$
- nearest neighbor approaches can be thought of as approximating the target function at the single query point x_q
- locally weighted regression is a generalization to this approach, because it constructs an explicit approximation of f over a local region surrounding x_q

Locally weighted regression

- target function is approximated using a **linear function**

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \ldots + w_n a_n(x)$$

- methods like **gradient descent** can be used to calculate the coefficients $w_0, w_1, \ldots, w_n$ to minimize the error in fitting such linear functions

- ANNs require a global approximation to the target function
- here, just a local approximation is needed

$\Rightarrow$ the error function has to be redefined

Locally weighted regression

- possibilities to redefine the error criterion E
  1. Minimize the squared error over just the k nearest neighbors

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neigbors}} (f(x) - \hat{f}(x))^2$$

  2. Minimize the squared error over the entire set D, while weighting the error of each training example by some decreasing function K of its distance from x_q

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

  3. Combine 1 and 2

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neighbors}} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

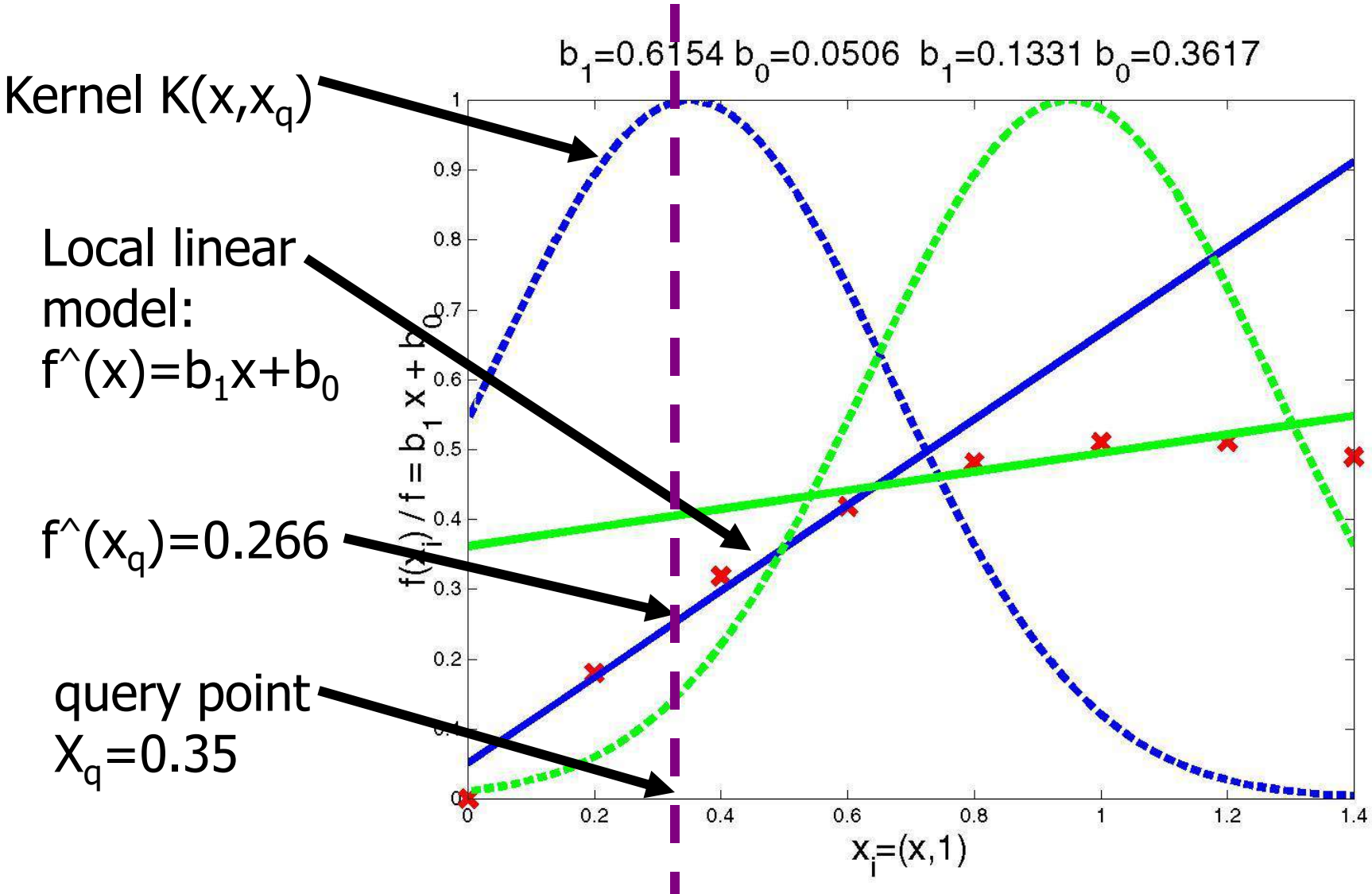
Locally weighted regression

- choice of the error criterion
  - E_2 is the most esthetically criterion, because it allows every training example to have impact on the classification of x_q
  - however, computational effort grows with the number of training examples
  - E_3 is a good approximation to E_2 with constant effort

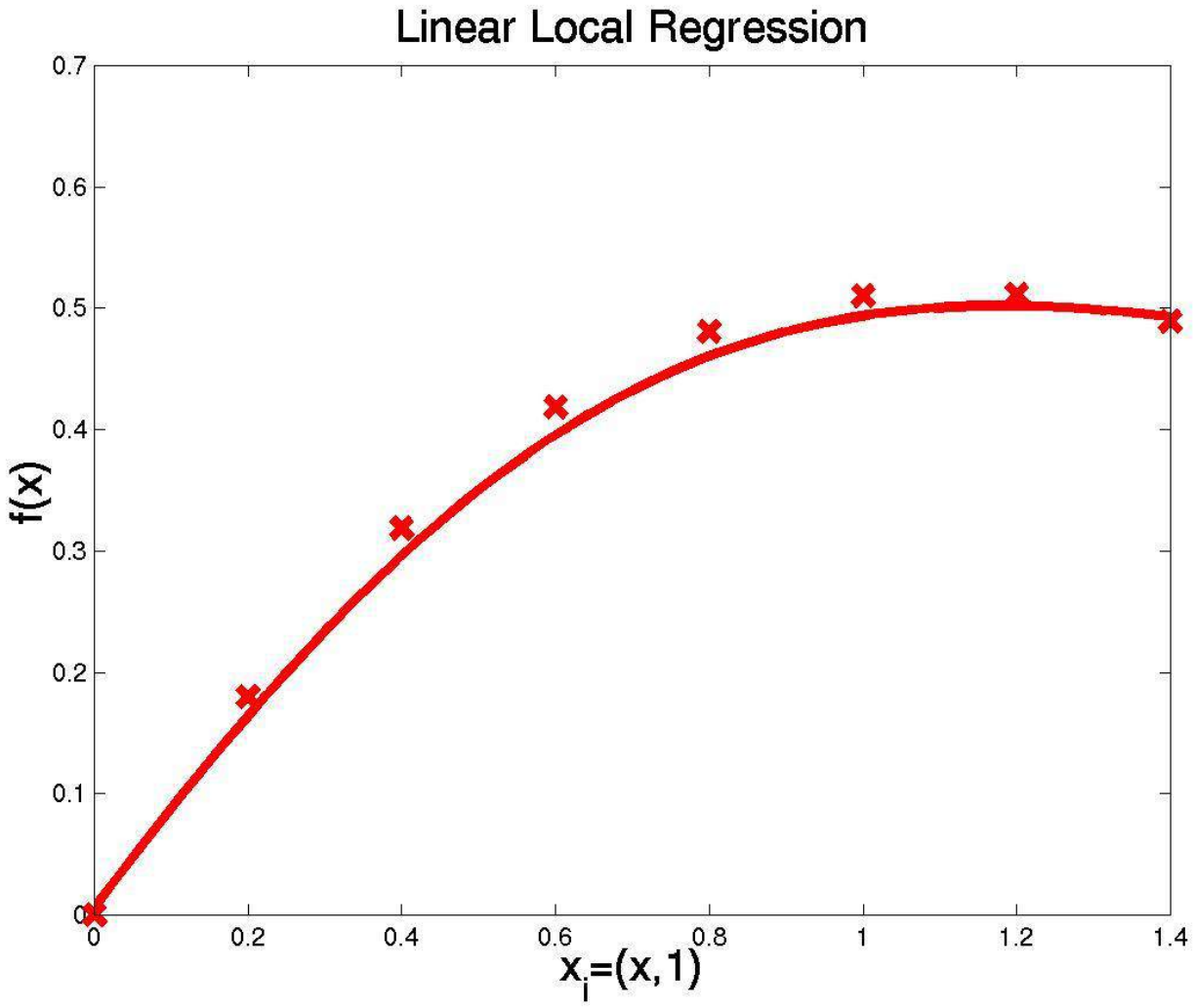
$$\Delta w_j = \eta \sum_{x \in k \text{ nearest neighbors}} K(d(x_q, x))(f(x) - \hat{f}(x))a_j$$

- Remarks on locally weighted linear regression:
  - in most cases, constant, linear or quadratic functions are used
  - costs for fitting more complex functions are prohibitively high
  - simple approximations are good enough over a sufficiently small subregion of X

Linear Local Model Example



Linear Local Model Example



Design Issues in Local Regression

- Local model order (constant, linear, quadratic)
- Distance function d
  - feature scaling: $d(x,q)=(\Sigma_{j=1}^{d} m_j(x_j-q_j)^2)^{1/2}$
  - irrelevant dimensions $m_j=0$
- kernel function K

See paper by Atkeson [1996] "Locally Weighted Learning"

Radial Basis Function

Radial Basis Function

- closely related to distance-weighted regression and to ANNs
- learned hypotheses have the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u \cdot K_u(d(x_u, x))$$

where

  - each x_u is an instance from X and
  - $K_u(d(x_u, x))$ decreases as $d(x_u, x)$ increases and
  - k is a user-provided constant

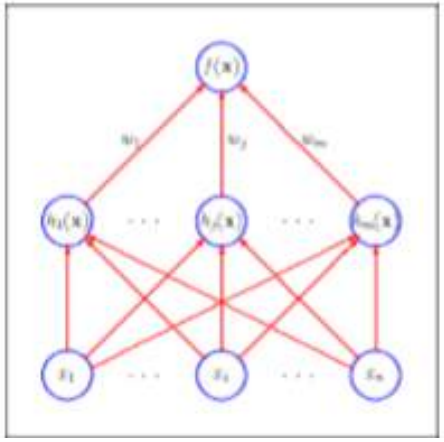
- though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution of each of the K_u terms is localized to a region nearby the point x_u

Radial Basis Function

- it is common to choose each function $K_u(d(x_u, x))$ to be a Gaussian function centered at x_u with some variance σ^2

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

- the function of $\hat{f}(x)$ can be viewed as describing a two-layer network where the first layer of units computes the various $K_u(d(x_u, x))$ values and the second layer a linear combination of the results



Training Radial Basis Function Networks

How to choose the center x_n for each Kernel function K_n?

- scatter uniformly across instance space
- use distribution of training instances (clustering)

How to train the weights?

- Choose mean x_n and variance σ_n for each K_n
  non-linear optimization or EM
- Hold K_n fixed and use local linear regression to compute the optimal weights w_n

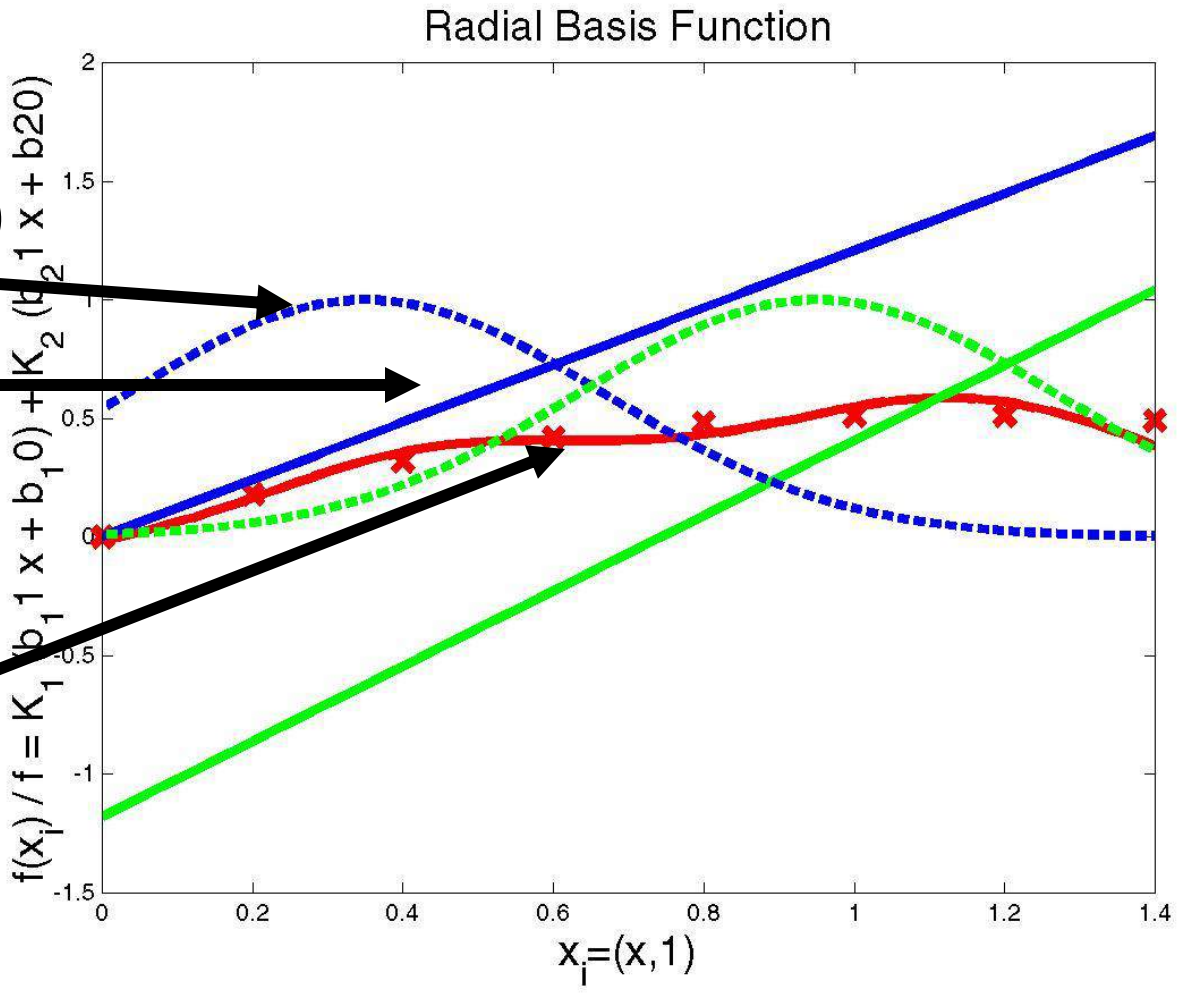
Radial Basis Network Example

$K_1(d(x_1,x))=$

$\exp(-1/2\ d(x_1,x)^2/\sigma^2)$

$w_1\ x+\ w_0$

$\hat{f}(x) = K_1\ (w_1\ x+\ w_0) + K_2\ (w_3\ x\ +\ w_2)$



and Eager Learning

Lazy: wait for query before generalizing

- k-nearest neighbors, weighted linear regression

Eager: generalize before seeing query

- Radial basis function networks, decision trees, back-propagation
- Eager learner must create global approximation

Lazy learner can create local approximations
If they use the same hypothesis space, lazy can represent more complex functions (H=linear functions)

Literature & Software

- T. Mitchell, “Machine Learning”, chapter 8, “Instance-Based Learning”
- “Locally Weighted Learning”, Christopher Atkeson, Andrew Moore, Stefan Schaal
- R. Duda et al., “Pattern recognition”, chapter 4 “Non-Parametric Techniques”

Netlab toolbox

- k-nearest neighbor classification
- Radial basis function networks

Thank You

BITS Pilani
Pilani Campus

Ensemble Learning

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

innovate achieve lead

BITS Pilani

Pilani Campus

Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Contents

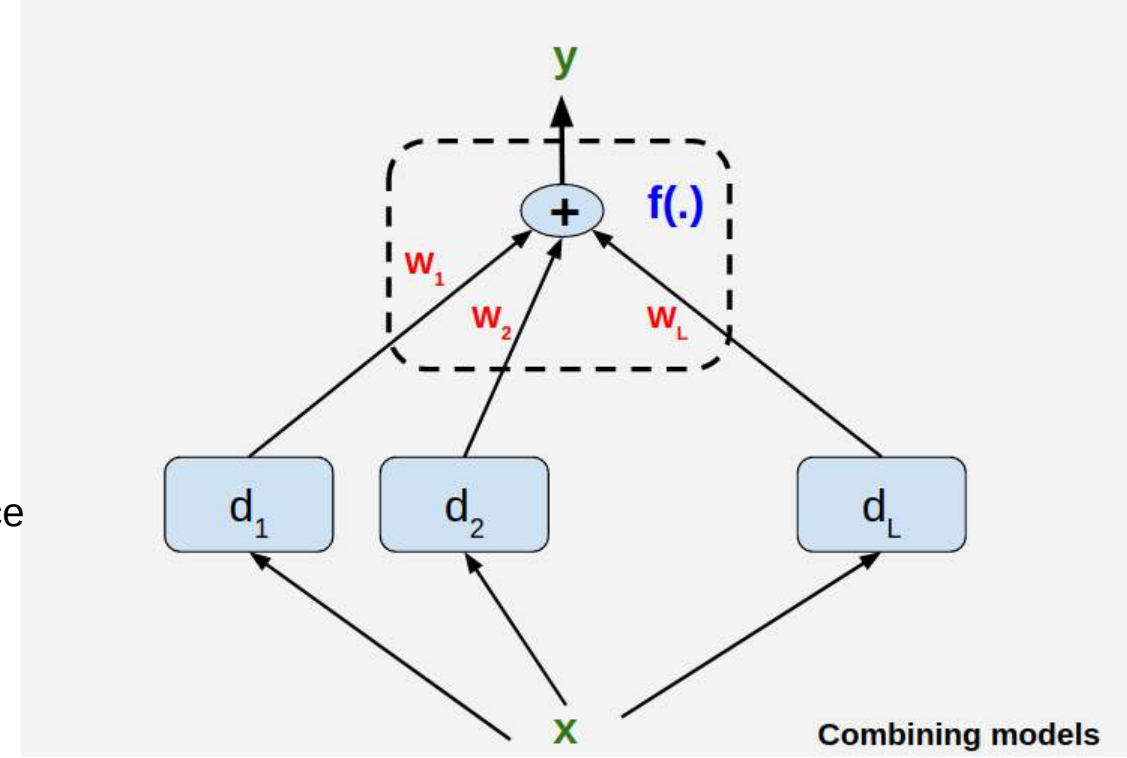
- Combining classifiers
- Bagging
- Boosting
- Random Forest Algorithm
- AdaBoost Algorithm
- Gradient Boosting

Getting Started

- No Free Lunch Theorem: There is no algorithm that is always the most accurate
- Each learning algorithm dictates a certain model that comes with a set of assumptions
  - Each algorithm converges to a different solution and fails under different circumstances
    - The best tuned learners could miss some or examples and there could be other learners which works better on (may be only) those !
  - In the absence of a single expert (*a superior model*) , a committee (*combinations of models*) can do better !
    - A committee can work in many ways ...

Committee of Models

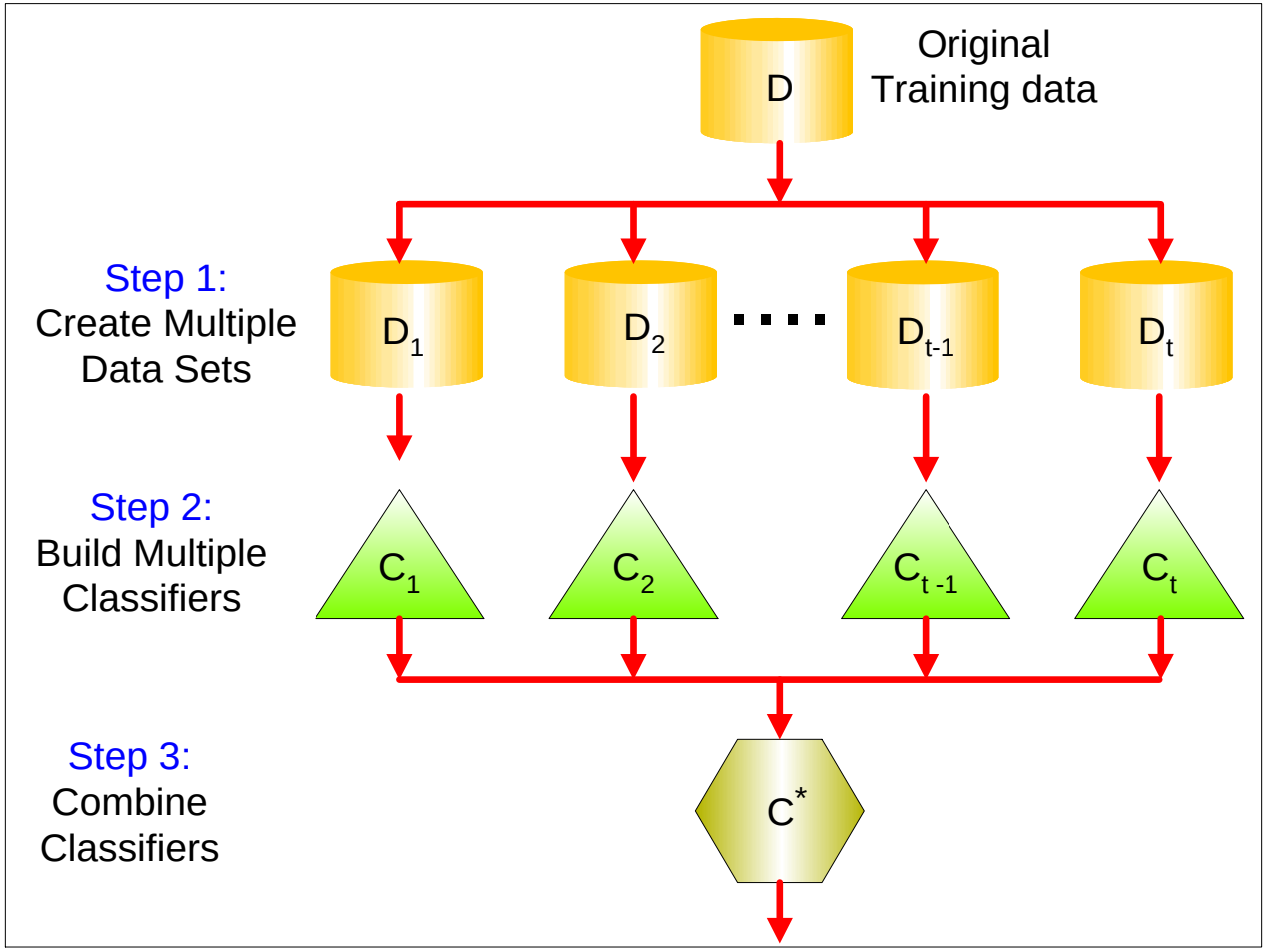
- Committee Members are base learners !
- Major challenges dealing with this committee
  - Expertise of each of the members (Does it help / not?)
  - Combining the results from the members for better performance



Ensemble Methods

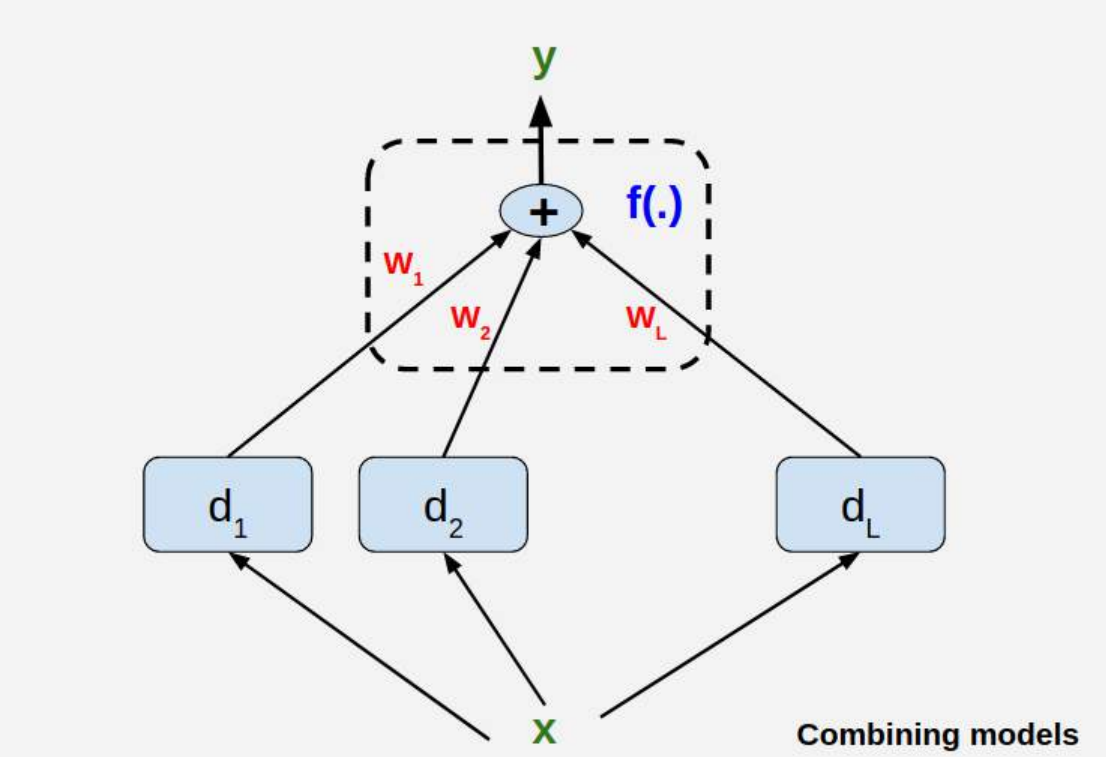
- **Ensemble methods** use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone
- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers
- Tend to reduce problems related to over-fitting of the training data.

General Approach



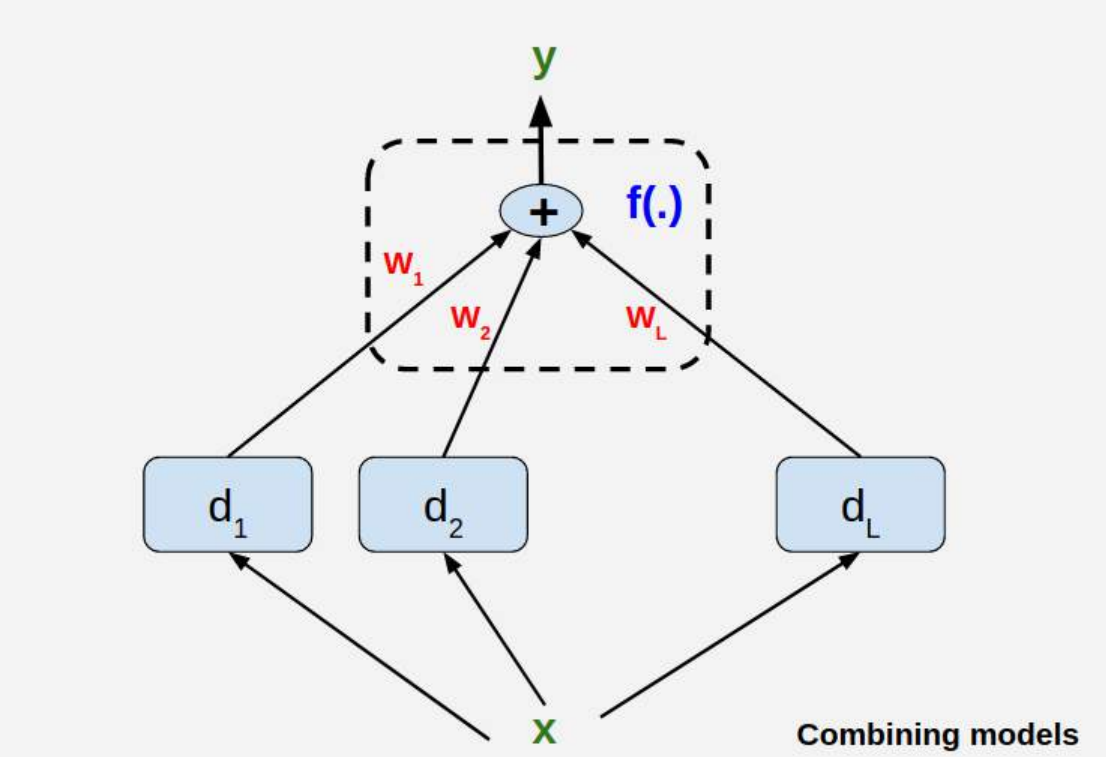
Issue 1 : On the members (Base Learners)

- It does not help if all learners are good/bad at roughly same thing
  - Need Diverse Learners



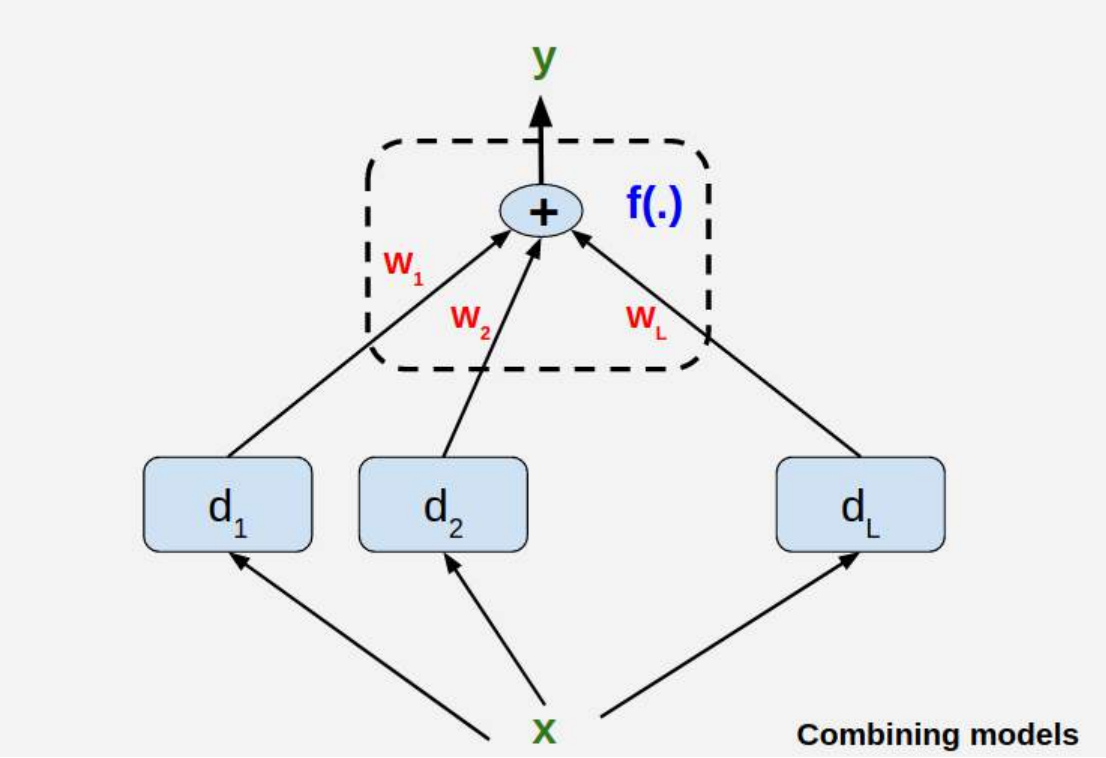
Issue 1 : On the members (Base Learners)

- Use Different Algorithms
  - Different algorithms make different assumptions
- Use Different Hyperparameters, that is ,
  - vary the structure of neural nets



Issue 1 : On the members (Base Learners)

- Different input representations
  - Uttered words + video information of speakers clips
  - image + text annotations
- Different training sets
  - Draw different random samples of data
  - Partition data in the input space and have learners specialized in those spaces (mixture of experts)



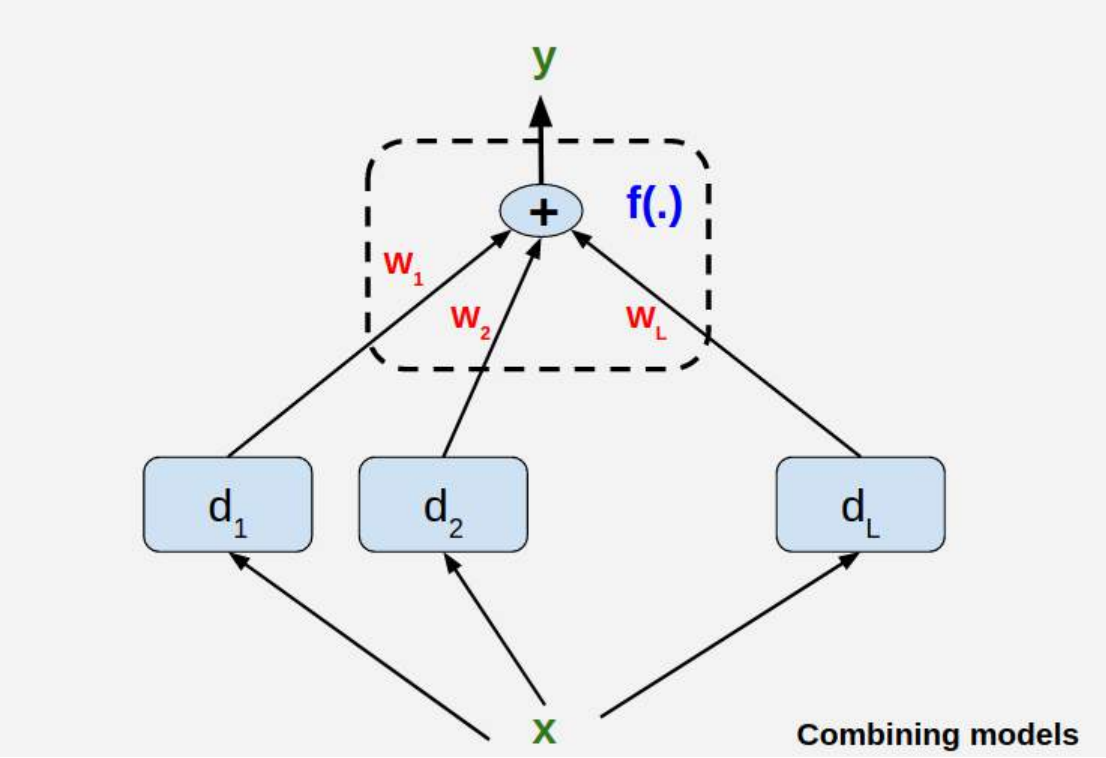
Issue -2 : Combining Results

$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

A Simple Combination Scheme:

$$y = \sum_{j=1}^{L} w_j d_j$$

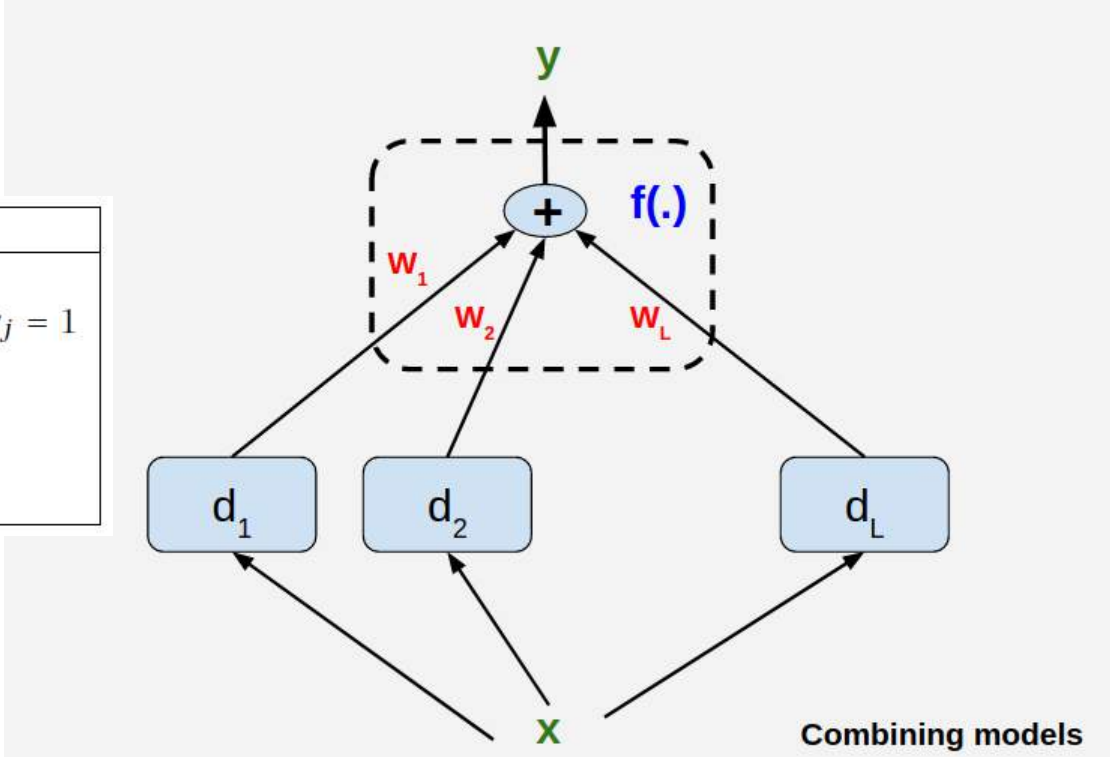
$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1$$



Issue -2 : Combining Results

$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

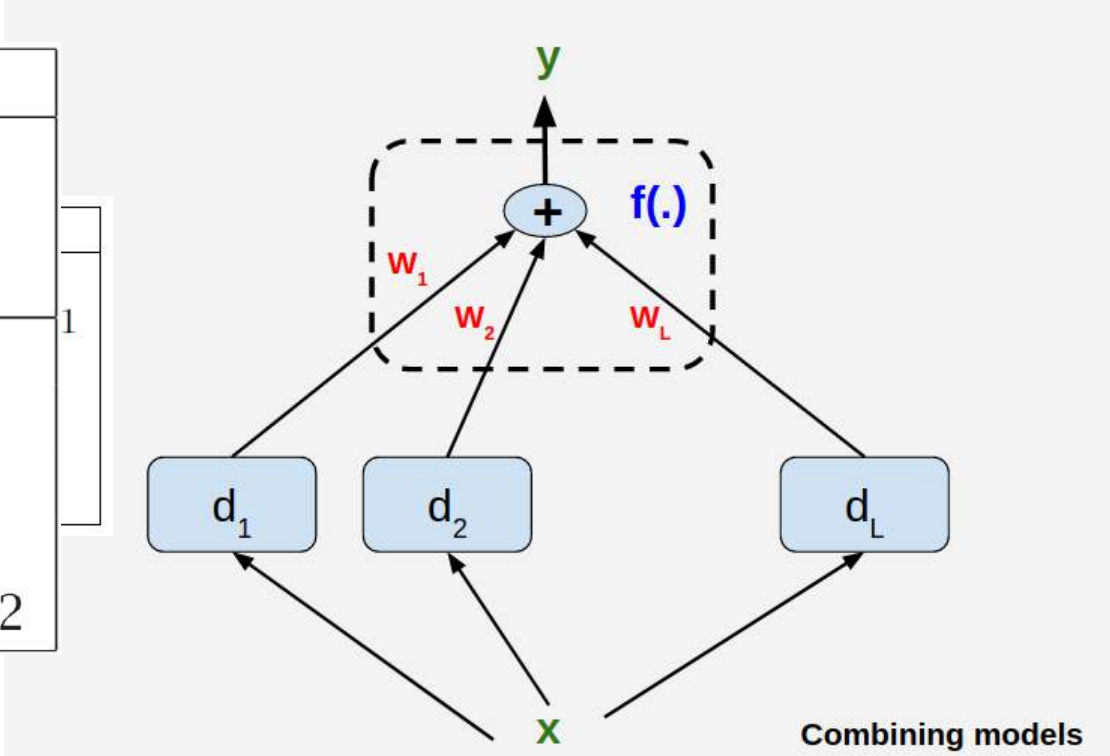
Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$



Issue -2 : Combining Results

	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
Sum	0.2	**0.5**	0.3
Median	0.2	**0.5**	0.4
Minimum	0.0	**0.4**	0.2
Maximum	0.4	**0.6**	0.4
Product	0.0	**0.12**	0.032

1

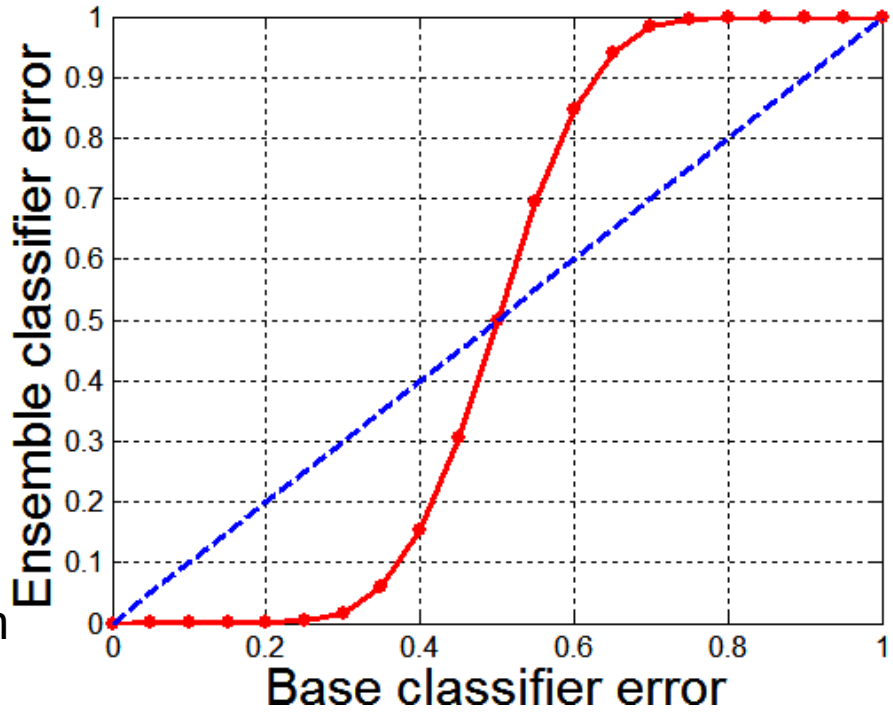


When does Ensemble work?

- Ensemble classifier performs better than the base classifiers when e is smaller than 0.5
- Necessary conditions for an ensemble classifier to perform better than a single classifier:
  - Base classifiers should be independent of each other
  - Base classifiers should do better than a classifier that performs random guessing

Why Ensemble Methods work?

- 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- If base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers depicted by dotted line
- Assume errors made by classifiers are uncorrelated
- Ensemble makes a wrong prediction only if base classifiers error is more than 0.5



Types of Ensemble Methods

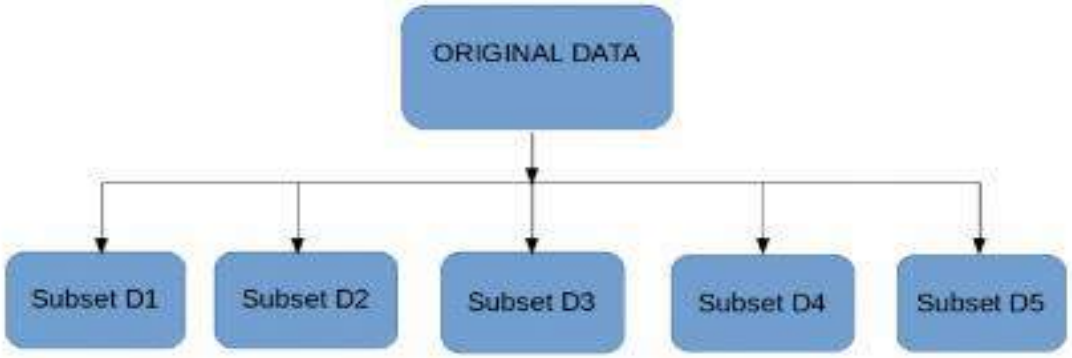
Manipulate data distribution
- Example: bagging, boosting

Manipulate input features
- Example: random forests

Bagging (Bootstrap Aggregating)

- Technique uses these subsets (bags) to get a fair idea of the distribution (complete set).
- The size of subsets created for bagging may be less than the original set.
- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**.
- When you sample with replacement, items are independent. One item does not affect the outcome of the other. You have 1/7 chance of choosing the first item and a 1/7 chance of choosing the second item.
- If the two items are **dependent**, or linked to each other. When you choose the first item, you have a 1/7 probability of picking a item. Assuming you don't replace the item, you only have six items to pick from. That gives you a 1/6 chance of choosing a second item.

Bagging



- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.

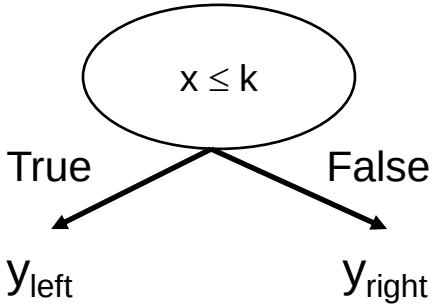
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Bagging Algorithm

Algorithm 5.6 Bagging Algorithm

1: Let k be the number of bootstrap samples.
2: for $i = 1$ to k do
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: end for
6: $C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

Boosting

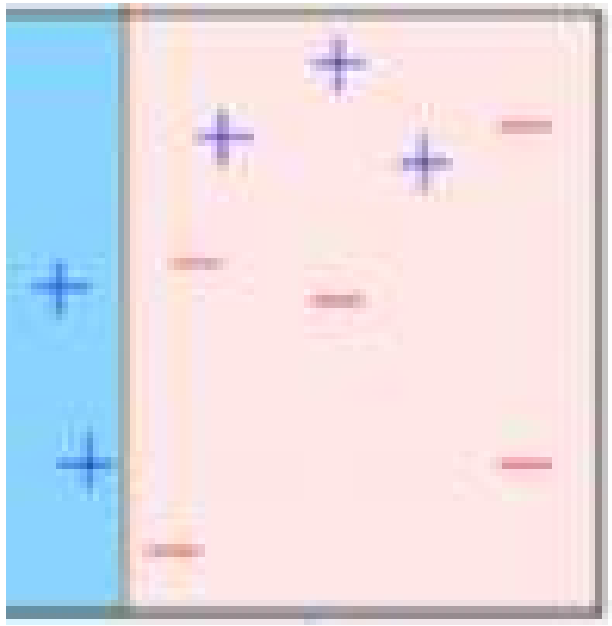
- What if a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.
- Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model.
- The succeeding models are dependent on the previous model.

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of each boosting round

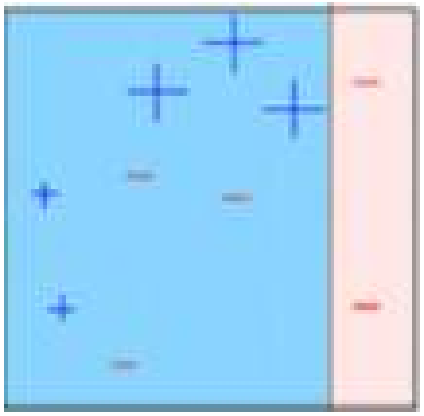
Boosting

- A subset is created from the original dataset.
- Initially, all data points are given equal weights.
- A base model is created on this subset.
- This model is used to make predictions on the whole dataset.



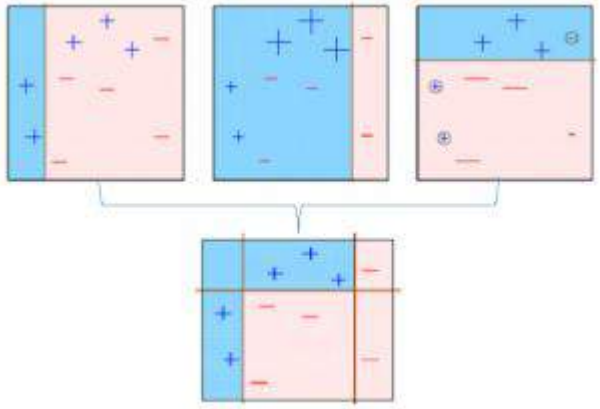
Boosting

- Errors are calculated using the actual values and predicted values.
- The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)
- Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)



Boosting

- Similarly, multiple models are created, each correcting the errors of the previous model.
- The final model (strong learner) is the weighted mean of all the models (weak learners).



- Individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

Algorithms based on Bagging and Boosting

Bagging algorithms:

- Random forest

Boosting algorithms:

- AdaBoost

Random Forest

- Random Forest is ensemble machine learning algorithm that follows the bagging technique.
- The base estimators in random forest are decision trees.
- Random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

Random Forest

- Random subsets are created from the original dataset (bootstrapping).
- At each node in the decision tree, only a random set of features are considered to decide the best split.
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.


All Data
random subset
random subset
random subset
random subset
tree
tree
tree
tree
tree
At each node:
choose some ballsubset of variables at random
find a variable (and a value for that variable) which optimizes the split

Advantages of Random Forest

- Algorithm can solve both type of problems i.e. classification and regression
- Power to handle large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.
- Model outputs **Importance of variable,** which can be a very handy feature (on some random data set).

Disadvantages of Random Forest

- May over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!

AdaBoost

- Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model.
- AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

AdaBoost Algorithm

- Initially, all observations (n) in the dataset are given equal weights (1/n).
- A model is built on a subset of data.
- Using this model, predictions are made on the whole dataset.
- Errors are calculated by comparing the predictions and actual values.
- While creating the next model, higher weights are given to the data points which were predicted incorrectly.

Adaboost Algorithm

- Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
- This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

AdaBoost

- Base classifiers C_i: C_1, C_2, …, C_T
- Error rate:
  - N input samples

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

https://en.wikipedia.org/wiki/AdaBoost#Choosing_αt

AdaBoost Algorithm

Algorithm 5.7 AdaBoost Algorithm

1: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Initialize the weights for all n instances.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all instances in the original training set, D.
7: $\epsilon_i = \frac{1}{n}\left[\sum_j w_j \, \delta\left(C_i(x_j) \neq y_j\right)\right]$ {Calculate the weighted error}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Reset the weights for all n instances.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each instance according to equation (5.88).
14: **end for**
15: $C^*(\mathbf{x}) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y))$.

AdaBoost: Weight Update

Weight Update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$ <- Eqn:5.88

where Z_j is the normalization factor

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

- Reduce weight if correctly classified else increase
- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated

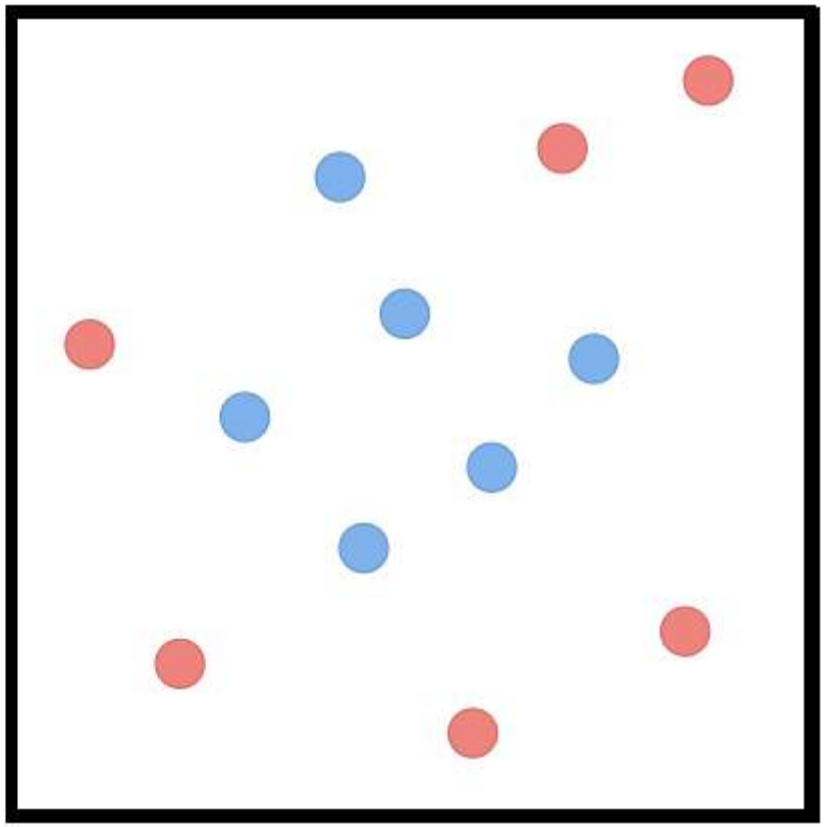
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



- Size of point represents the instance's weight

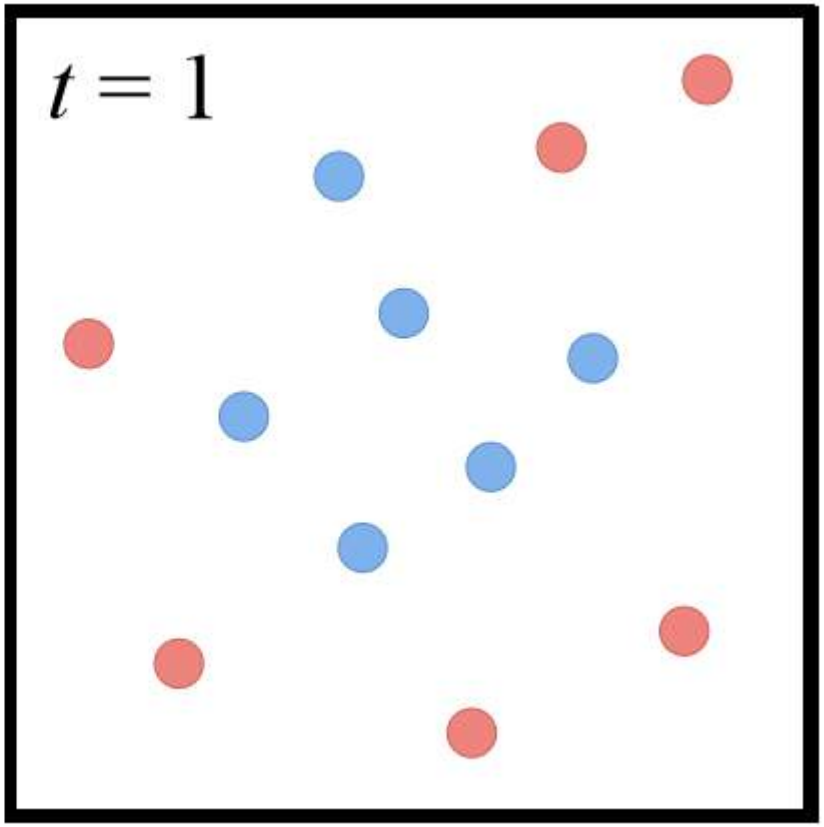
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



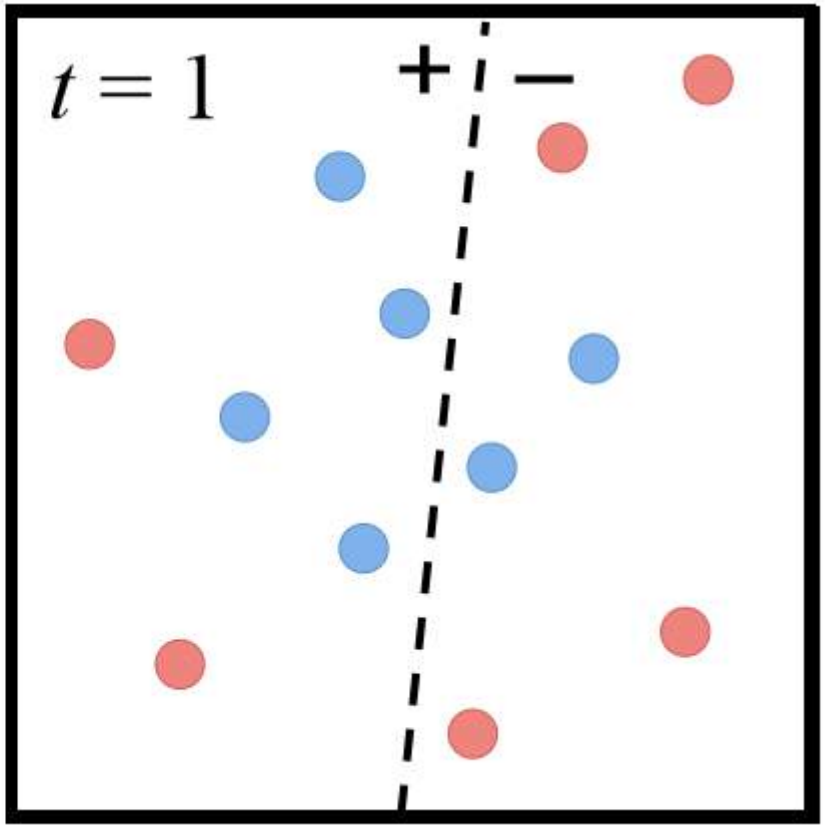
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



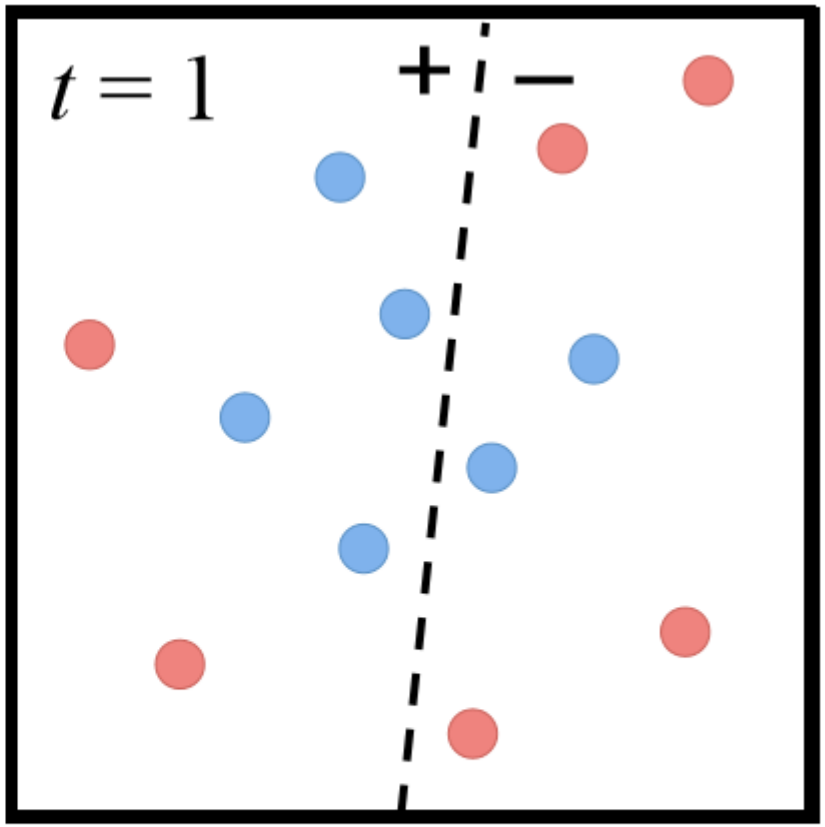
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

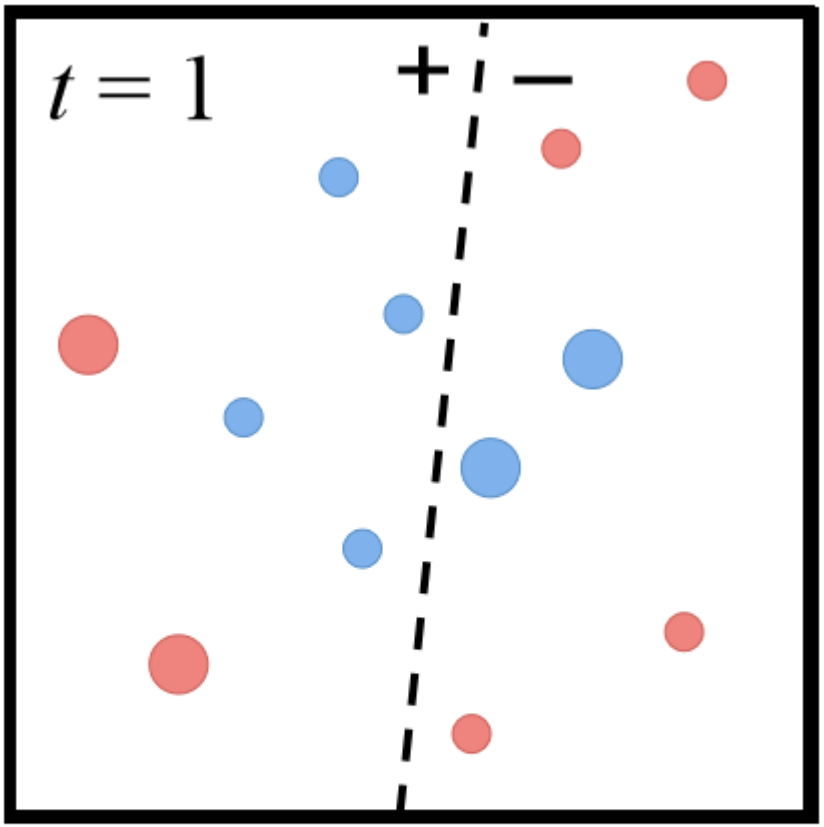
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

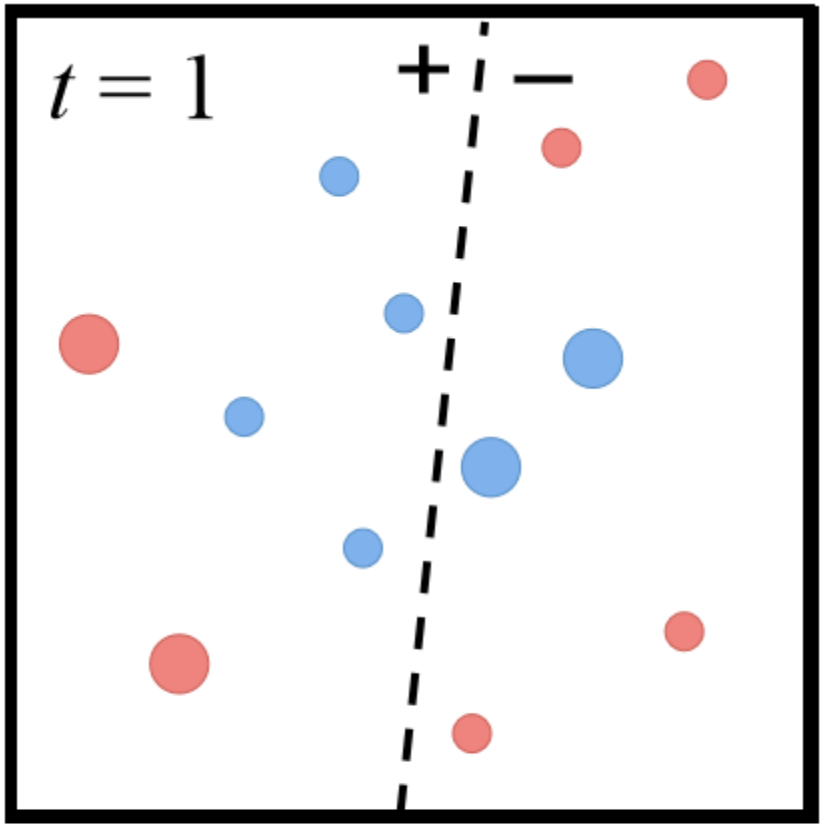
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



Disclaimer: Note that resized points in the illustration above are not necessarily to scale with β_t

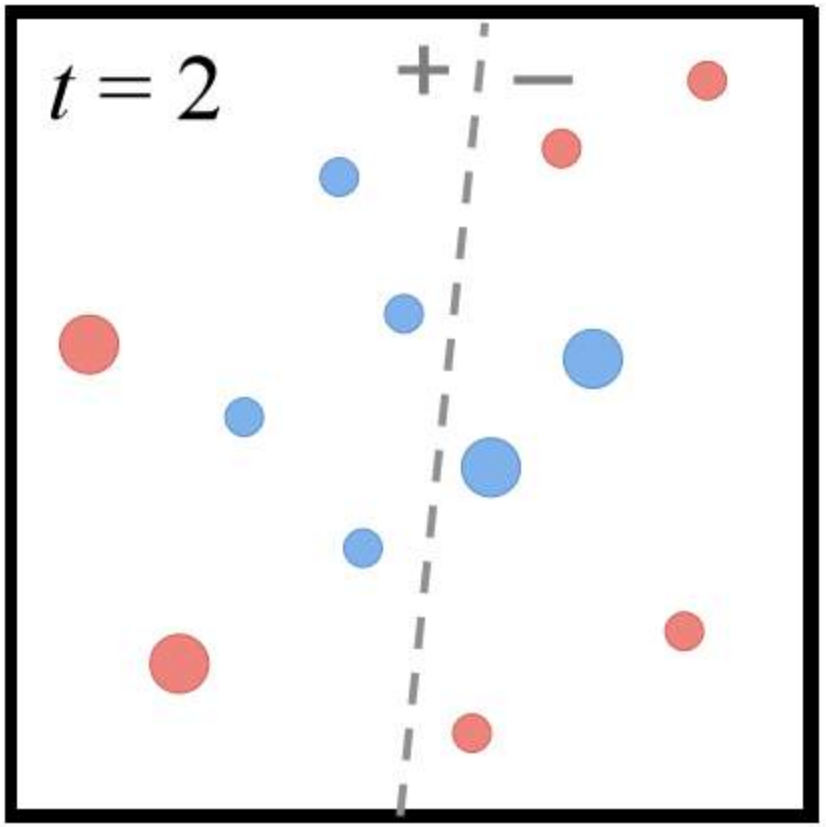
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



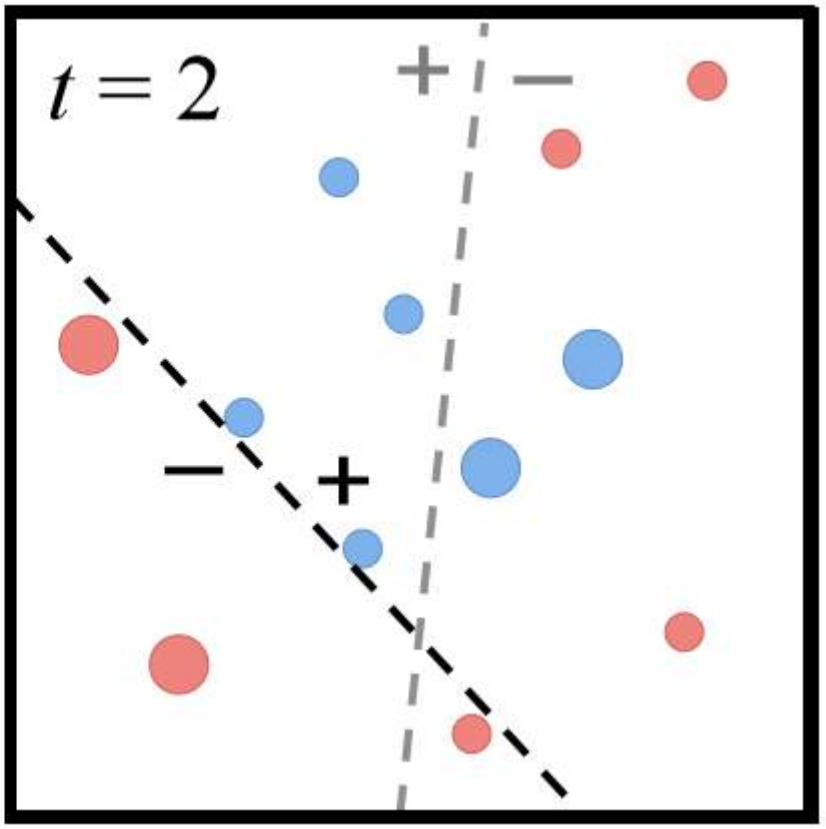
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



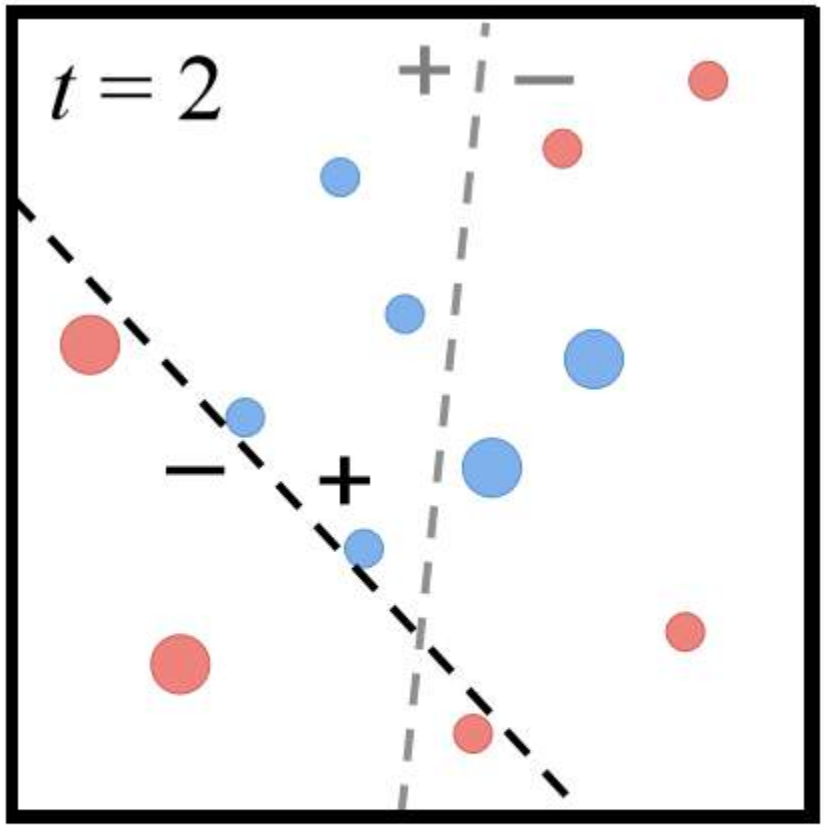
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

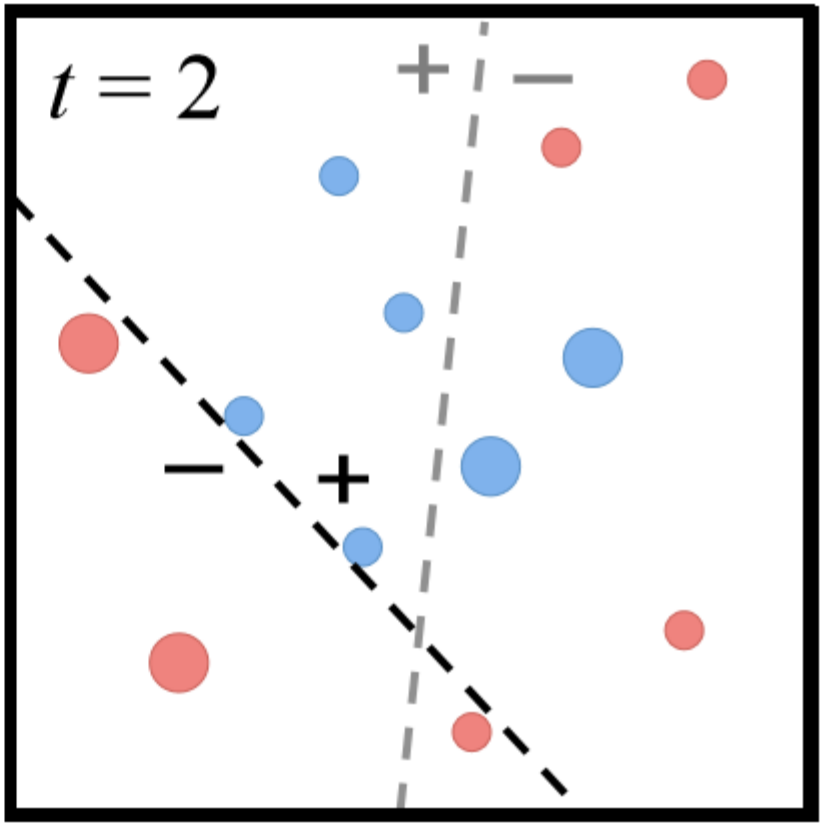
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

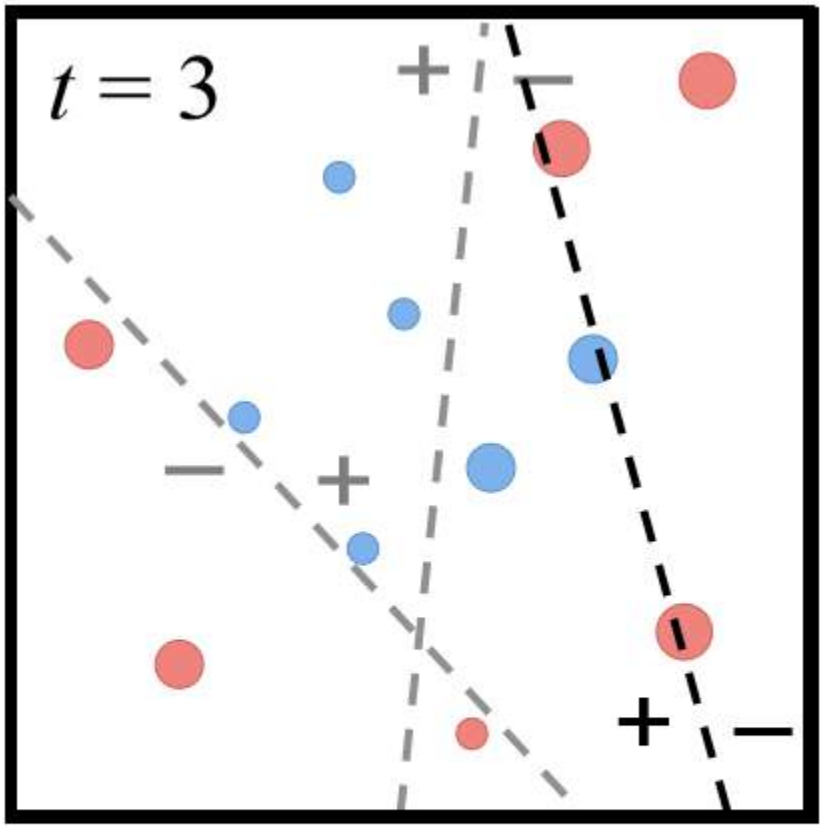
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

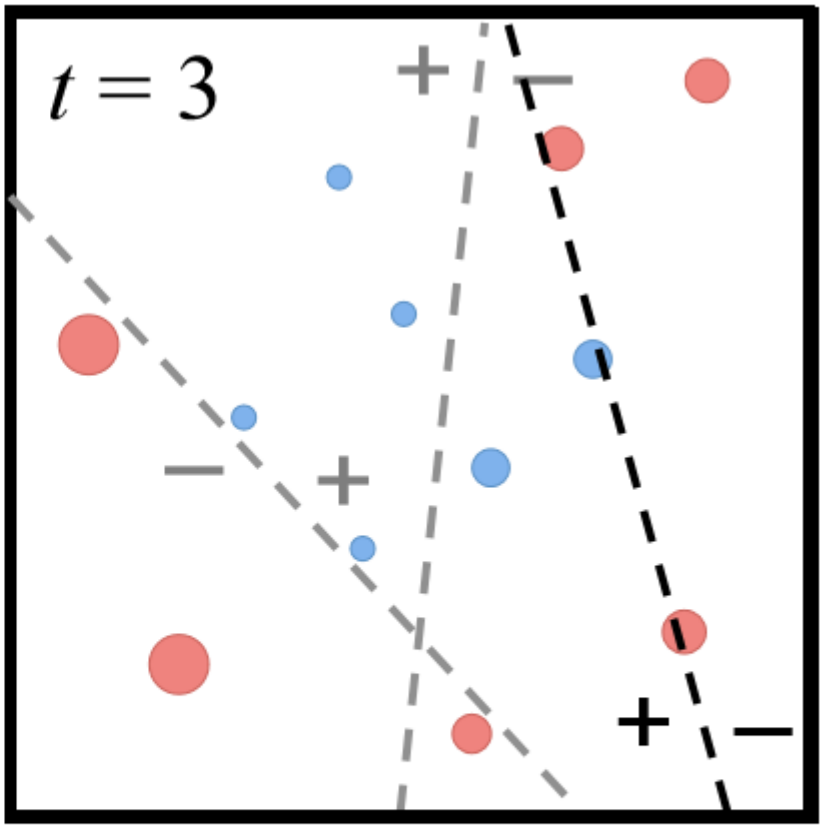
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

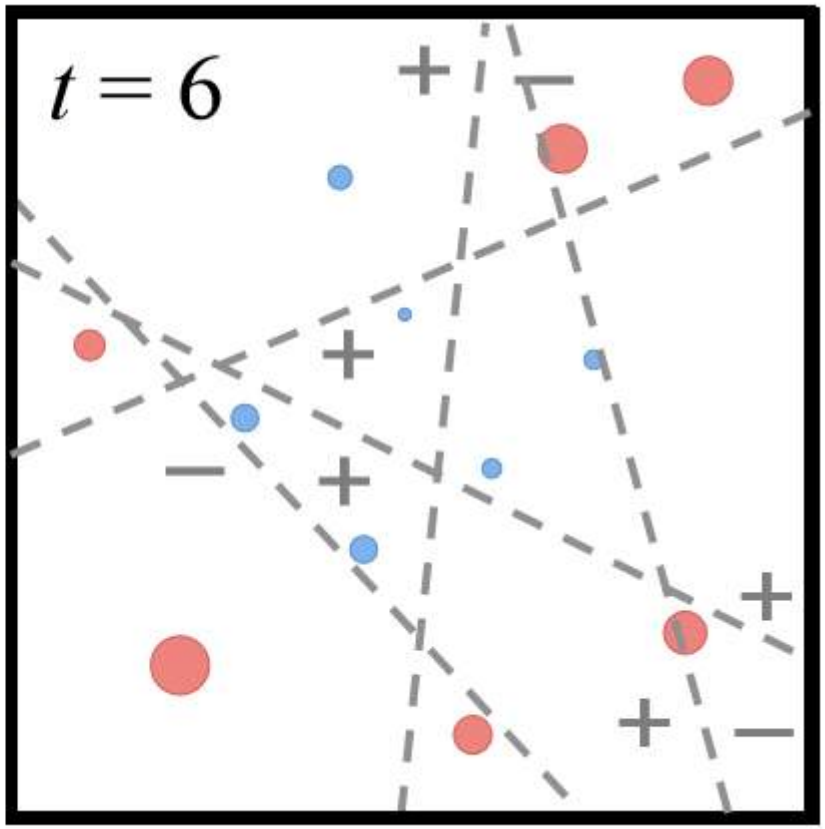
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$



AdaBoost Algorithm

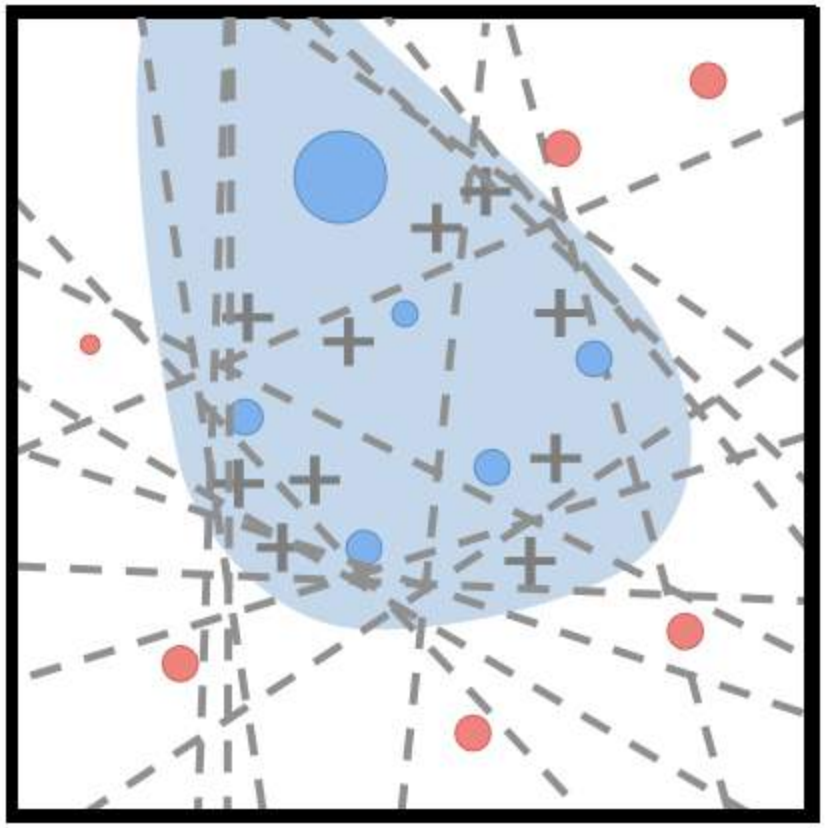
1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$

$t = \mathrm{T}$



- Final model is a weighted combination of members
  - Each member weighted by its importance

AdaBoost Algorithm

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$

2: **for** $t = 1, \ldots, T$

3: Train model h_t on X, y with instance weights $\mathbf{w}_t$

4: Compute the weighted training error rate of h_t:

$$\epsilon_t = \sum_{i: y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$

5: Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution:

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \ldots, n$$

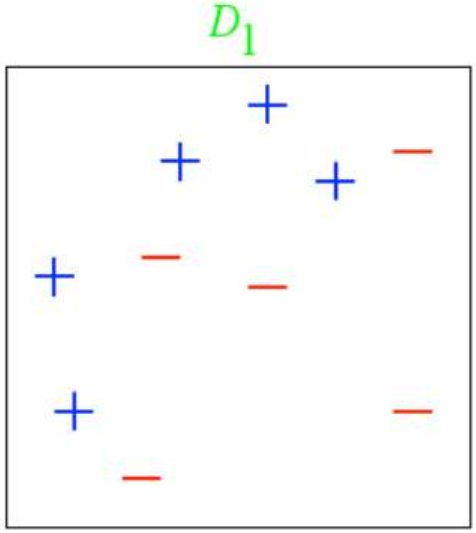
8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$

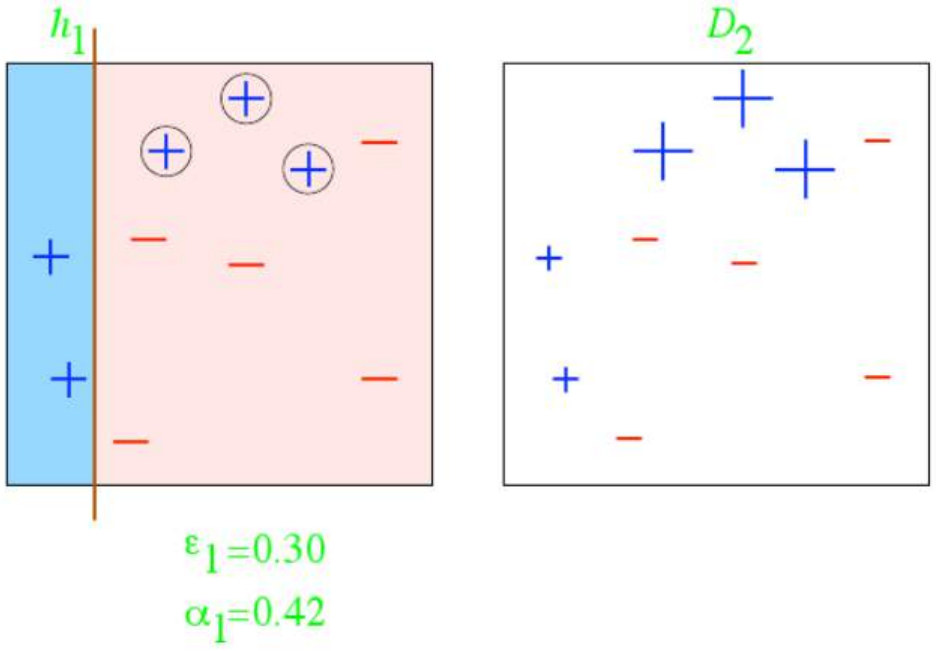
Member classifier with less error are given more weight in final ensemble hypothesis. Final prediction is a weighted combination of each members prediction

Example



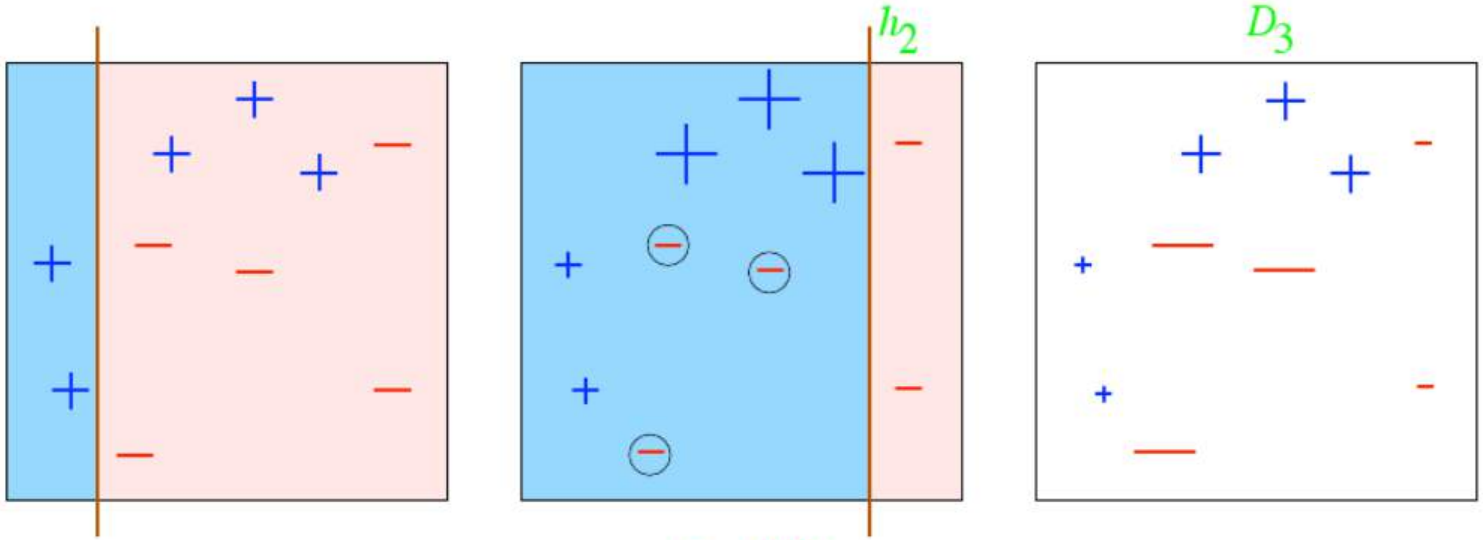
From, L´eon Bottou

Example



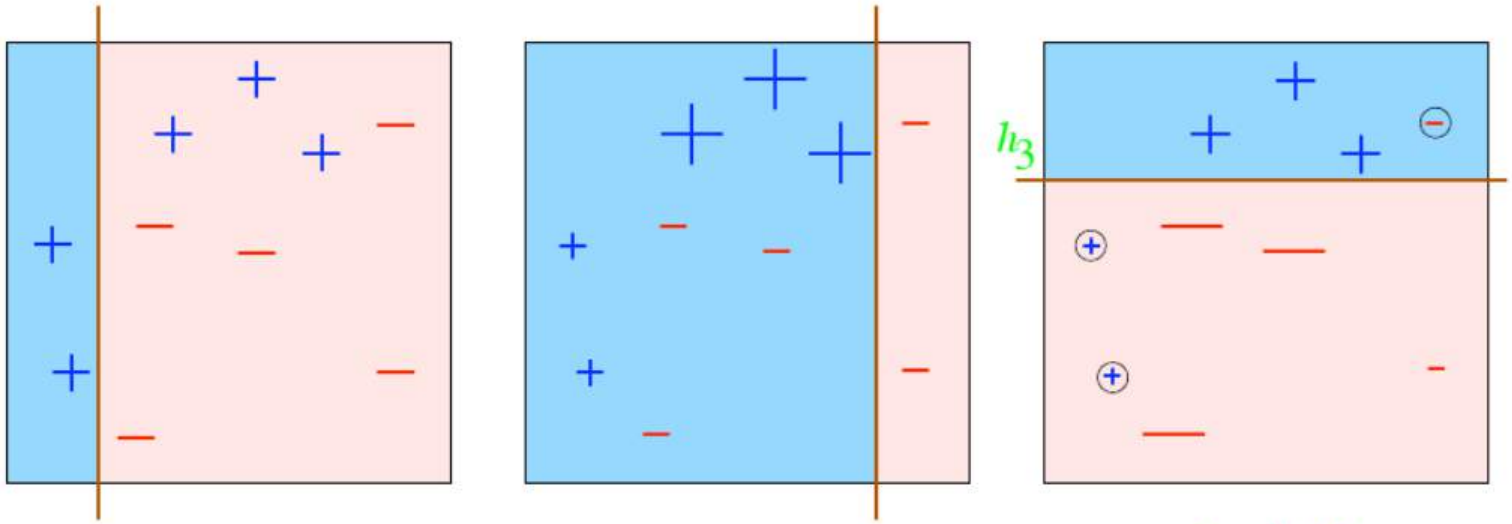
From, L´eon Bottou

Example



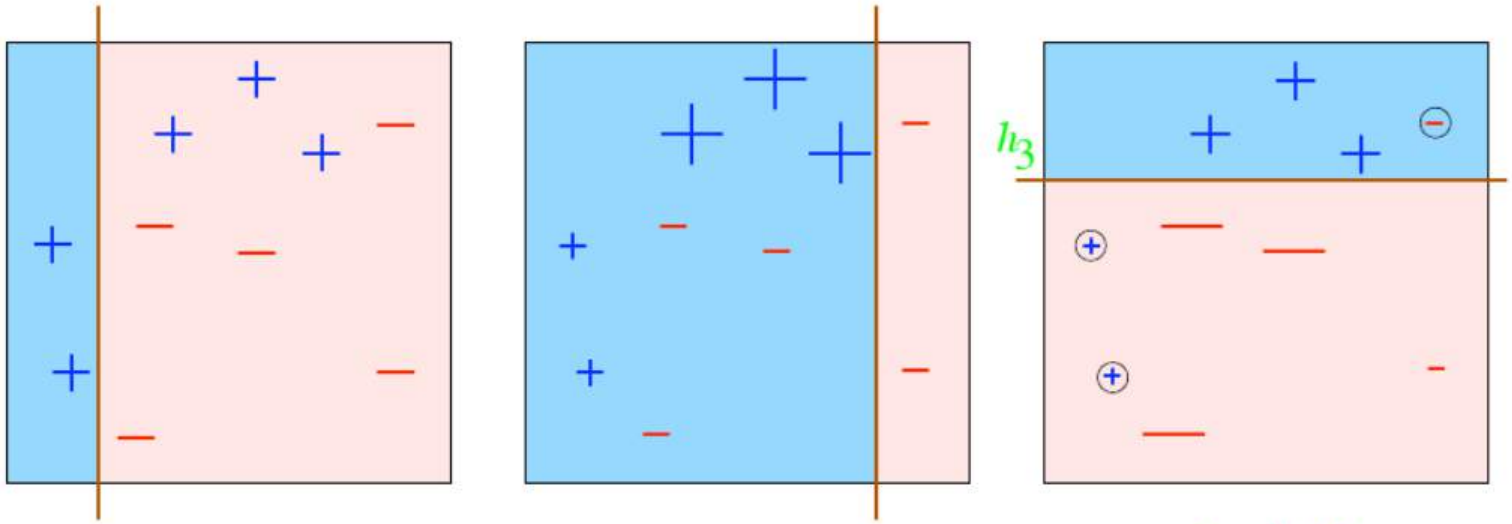
From, L´eon Bottou

Example



From, L´eon Bottou

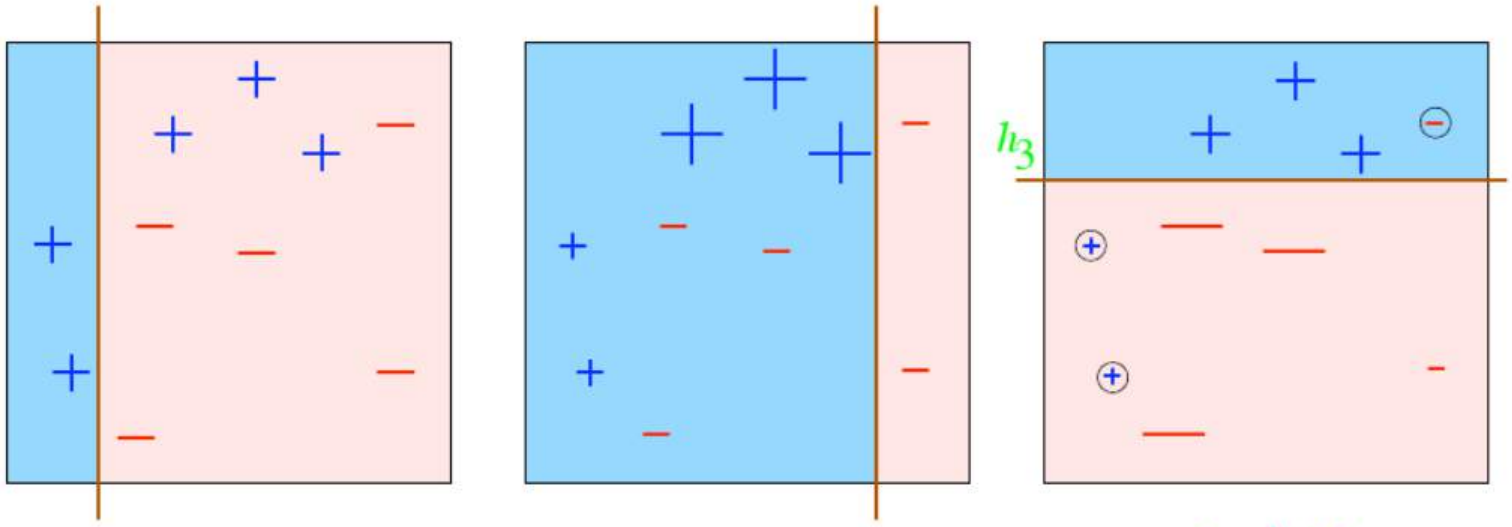
Example



From, L´eon Bottou

Example

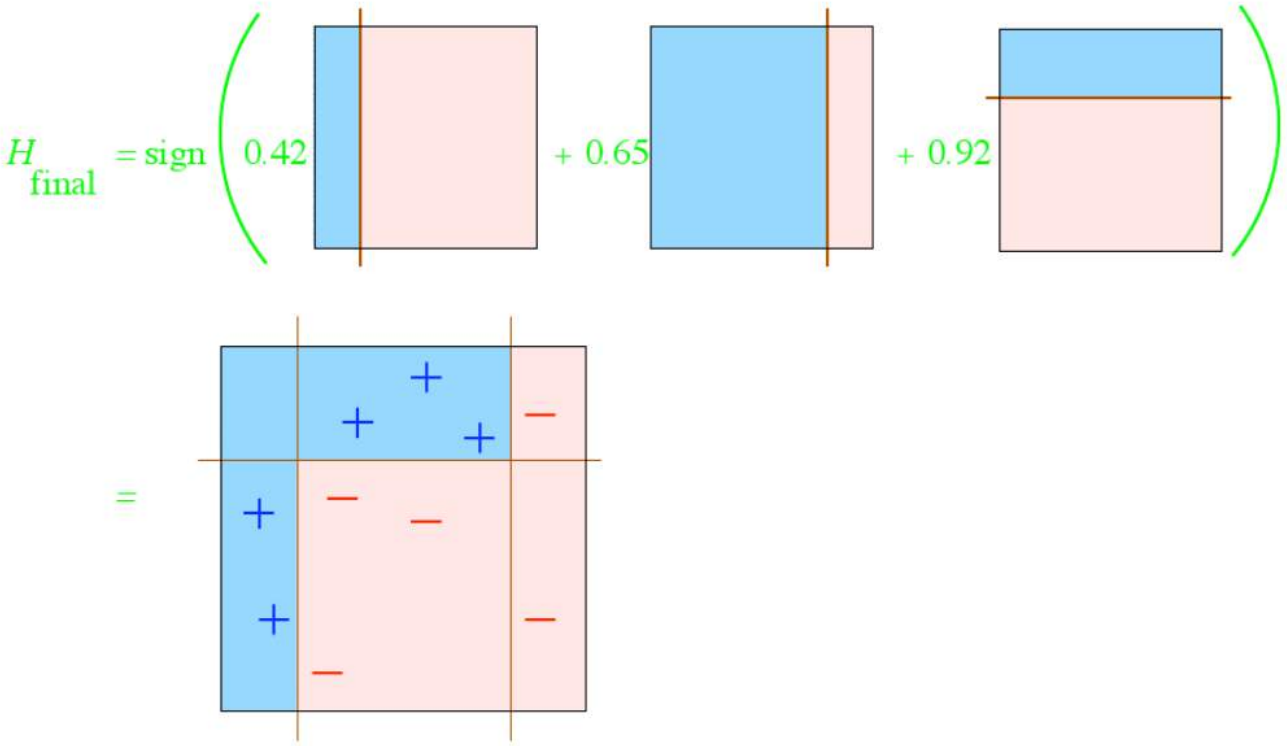
How do we combine the results now?



From, L´eon Bottou

Example

How do we combine the results now?



From, L´eon Bottou

AdaBoost base learners

- AdaBoost works best with "weak" learners
  - Should not be complex
  - Typically high bias classifiers
  - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
    - Can prove training error goes to 0 in O(log n) iterations

- Examples:
  - Decision stumps (1 level decision trees)
  - Depth-limited decision trees
  - Linear classifiers

AdaBoost in practice

Strengths:

- Fast and simple to program
- No parameters to tune (besides T)
- No assumptions on weak learner

When boosting can fail:

- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

Fine Tuning Ensembles

- Model combination does not always guaranteed to decrease error, unless
  - base-learners are diverse and accurate
- Ignore poor base learners
  - Use accuracy as a cut-off
  - Introduce some pruning with which at each iteration remove poor learners / learners whose absence lead to improvement (if any)
    - Modify iterations to allow both additions / deletions of learners
  - Discarding appropriately leads to better performance

References

The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011

Bishop - Pattern Recognition And Machine Learning - Springer 2006

A Gentle Introduction to Gradient Boosting
Cheng Li chengli@ccs.neu.edu College of Computer and Information Science Northeastern University

https://www.youtube.com/watch?time_continue=647&v=LsK-xG1cLYA&feature=emb_logo

Thank You!

Support Vector Machines

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani
Pilani Campus

Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

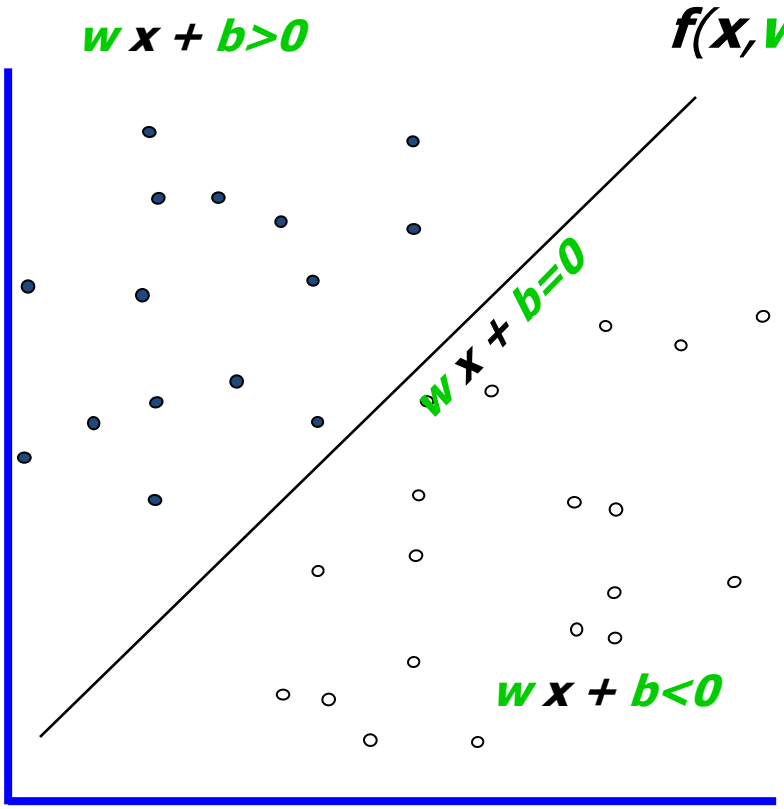
These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Topics to be covered

- Linear Classifiers
- Maximum Margin Classification
- Linear SVM
- SVM optimization problem
- Soft Margin SVM

Linear Classifiers

- denotes +1
- denotes -1

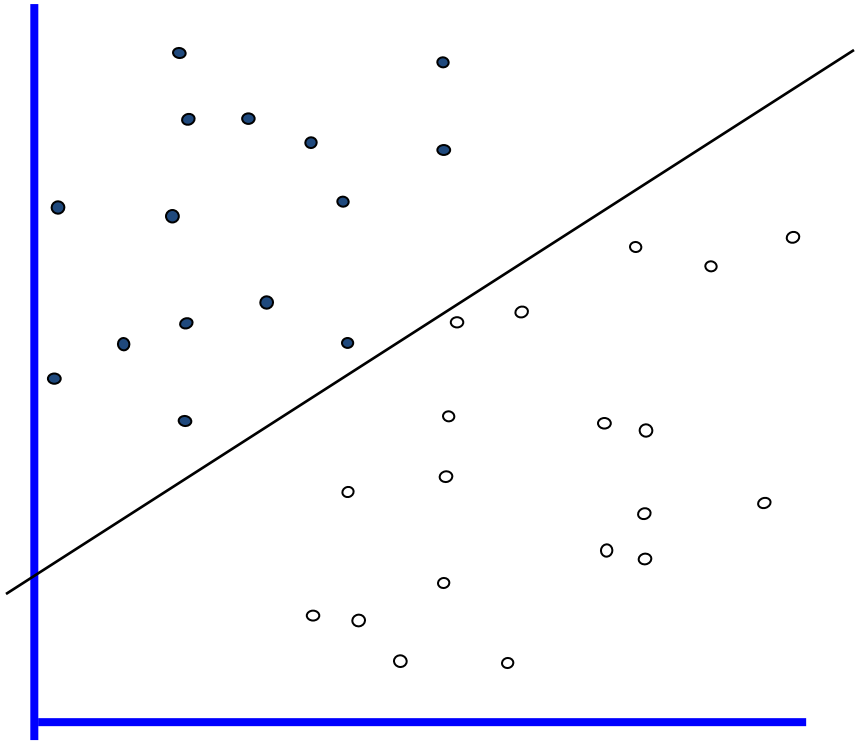


How would you classify this data?

Linear Classifiers

$$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}\,\mathbf{x} + b)$$

- denotes +1
- denotes -1

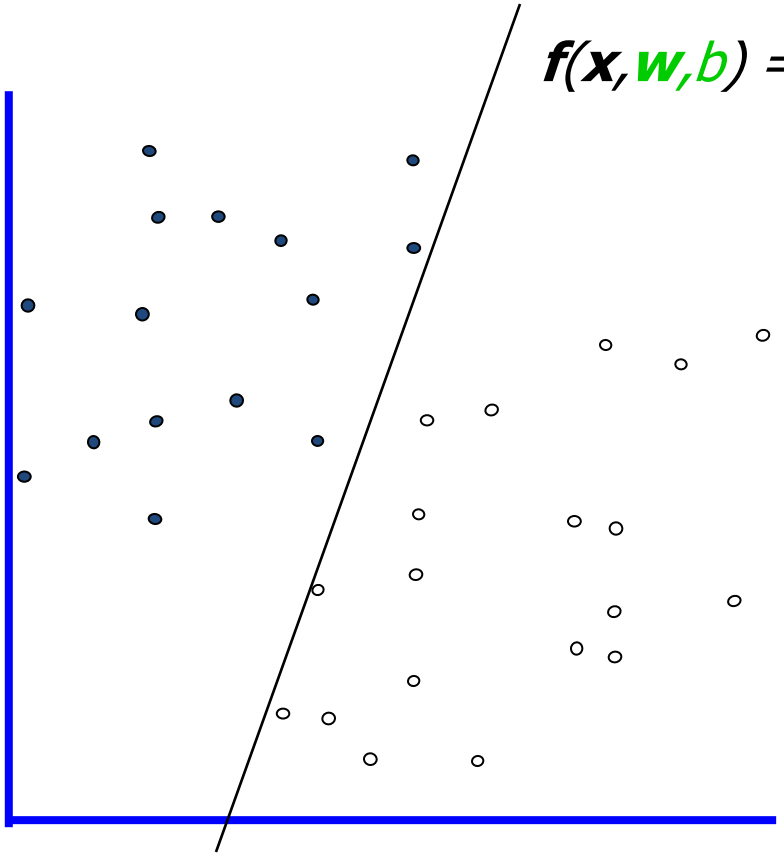


How would you classify this data?

Linear Classifiers

$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}\,\mathbf{x} + b)$

- denotes +1
- denotes -1

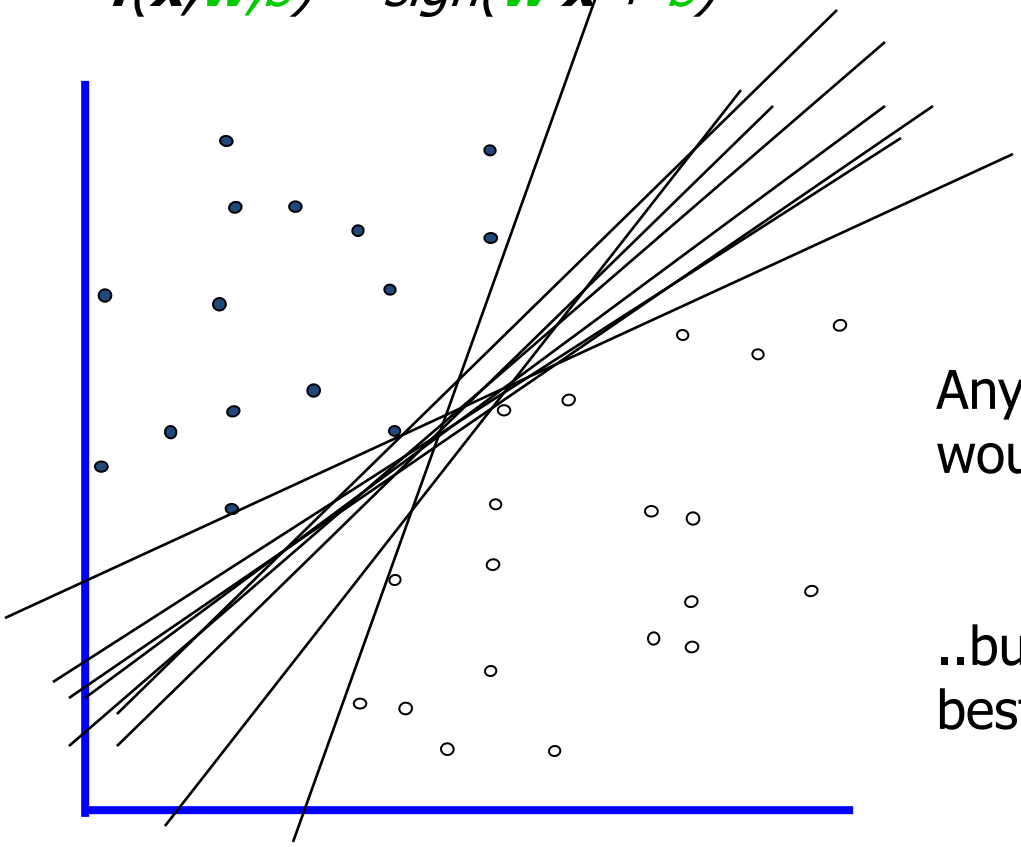


How would you classify this data?

Linear Classifiers

$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}\,\mathbf{x} + b)$

- denotes +1
- denotes -1



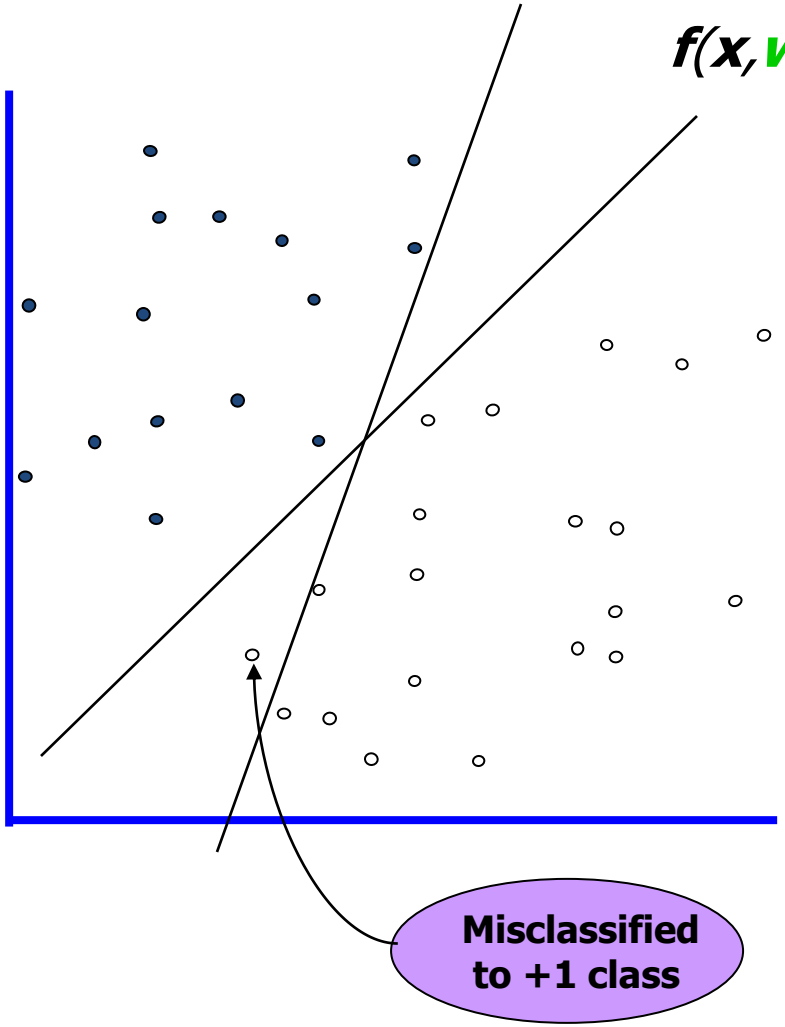
Any of these would be fine..

..but which is best?

Linear Classifiers

- denotes +1
- denotes -1

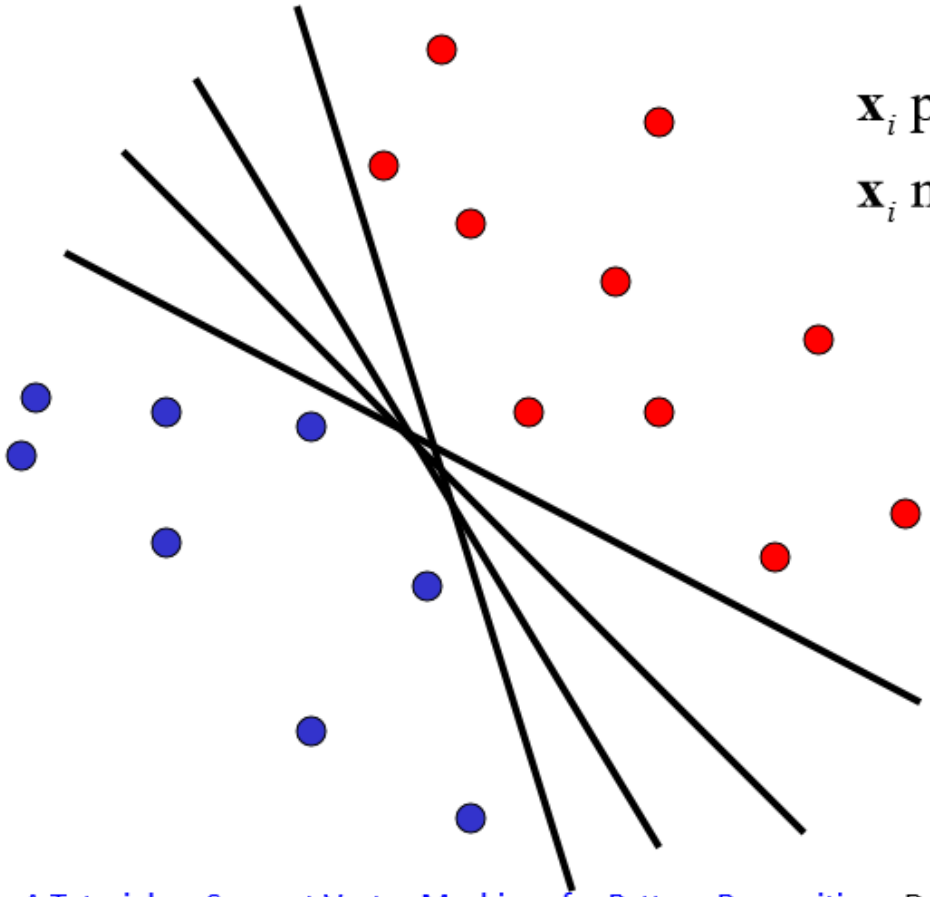
$$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}\,\mathbf{x} + b)$$



How would you classify this data?

Linear Classifier

- Find linear function to separate positive and negative examples



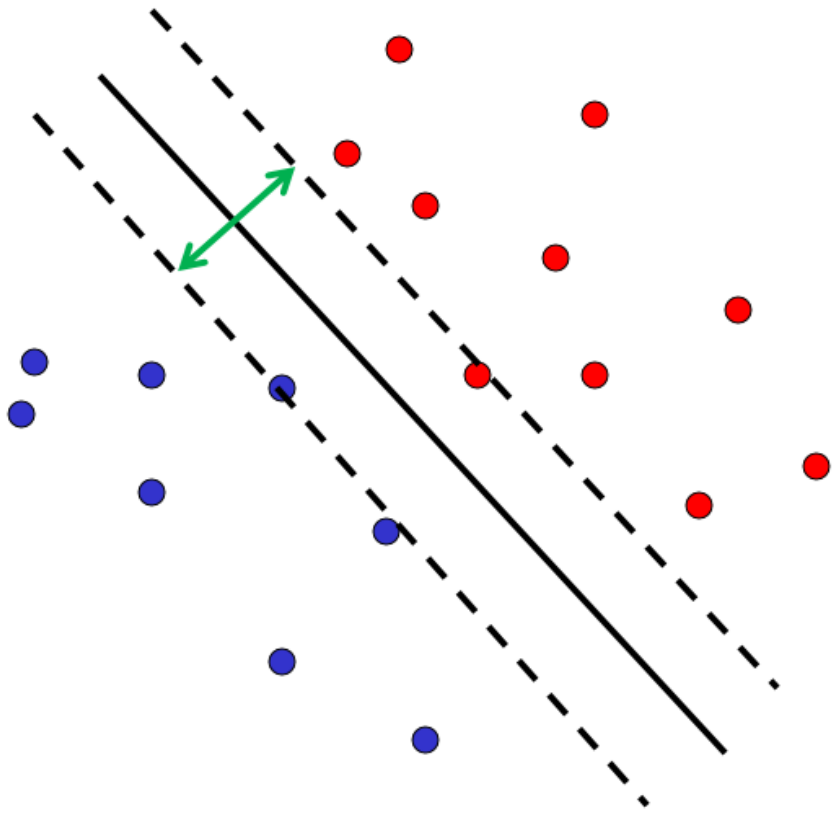
$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$

Which line is best?

. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

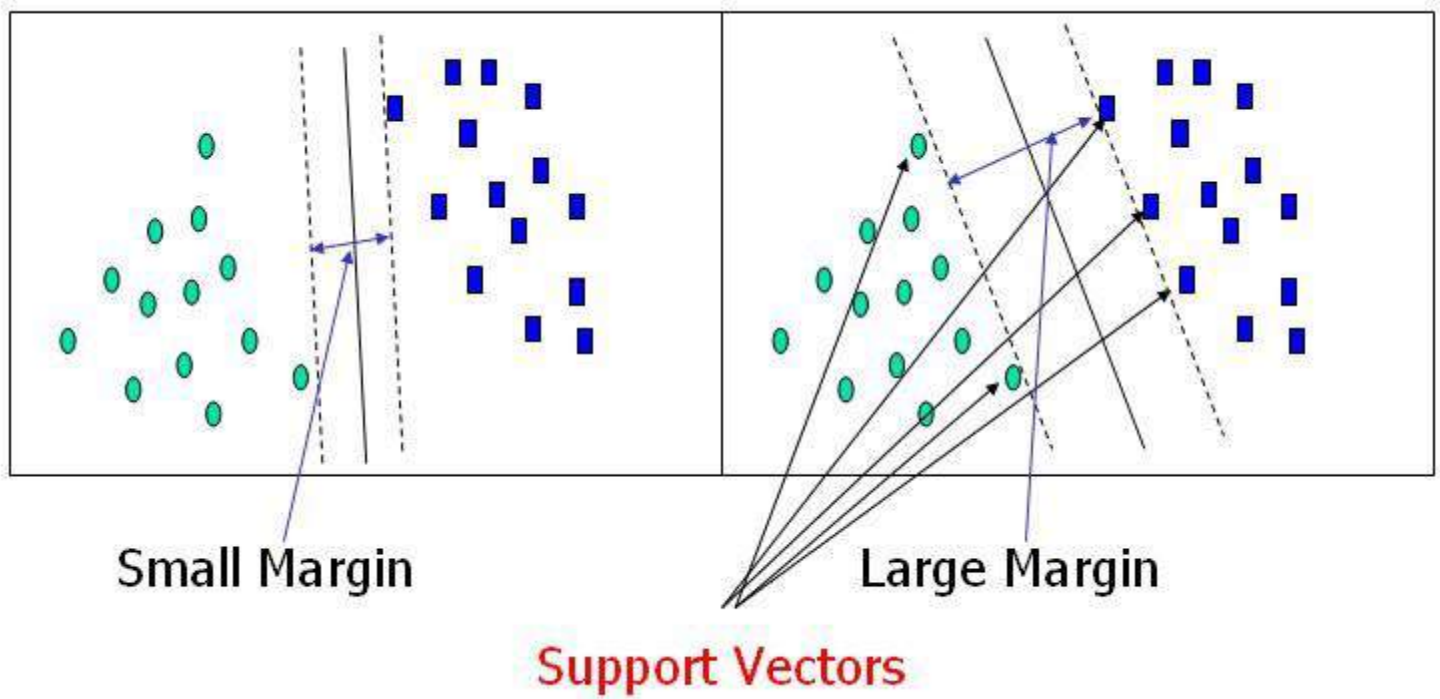
Linear Classifier



- Discriminative classifier based on *optimal separating line (for 2d case)*
- Maximize the *margin* between the positive and negative training examples

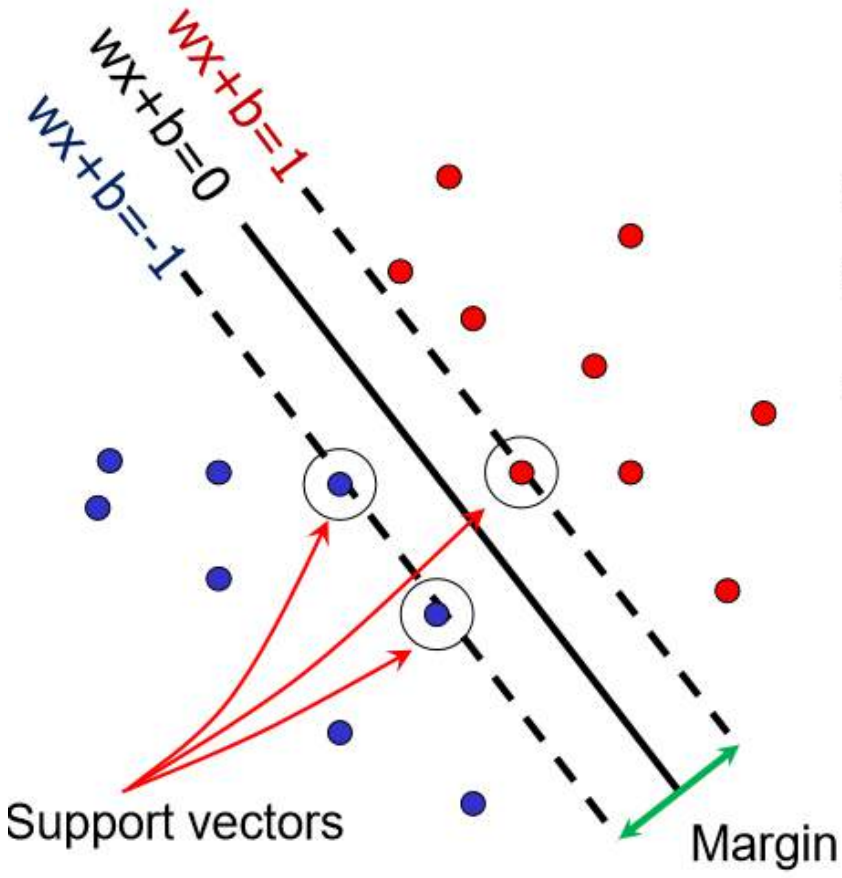
C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

Large margin and support vectors



Support Vector Machines

- Want line that maximizes the margin.

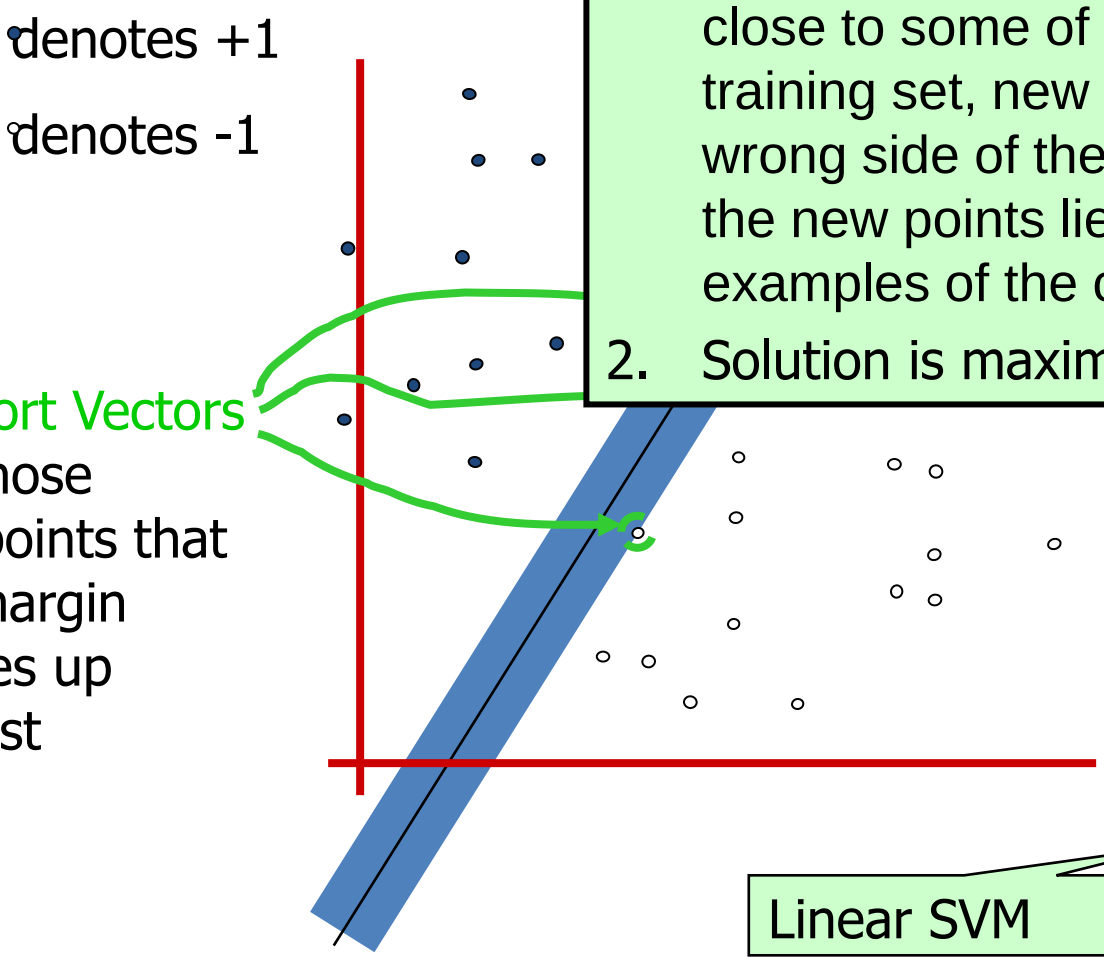


$\mathbf{x}_i$ positive $(y_i = 1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Maximum Margin



1. If hyperplane is oriented such that it is close to some of the points in your training set, new data may lie on the wrong side of the hyperplane, even if the new points lie close to training examples of the correct class.
2. Solution is maximizing the margin with the, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Support Vectors

- Geometric description of SVM is that the max-margin hyperplane is completely determined by those points that lie nearest to it.
- Points that lie on this margin are the support vectors.
- The points of our data set which if removed, would alter the position of the dividing hyperplane

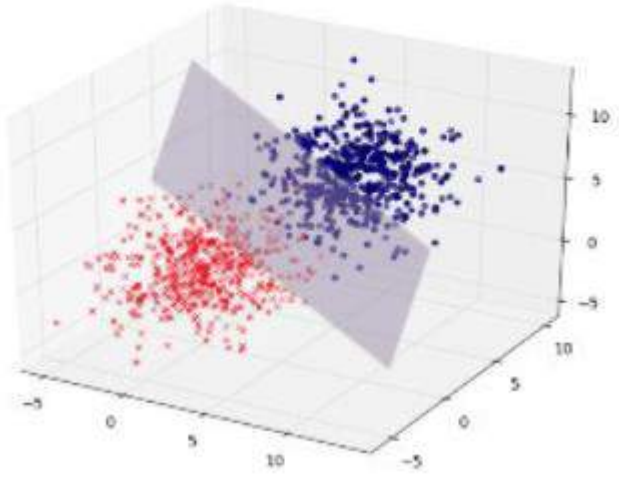
Example

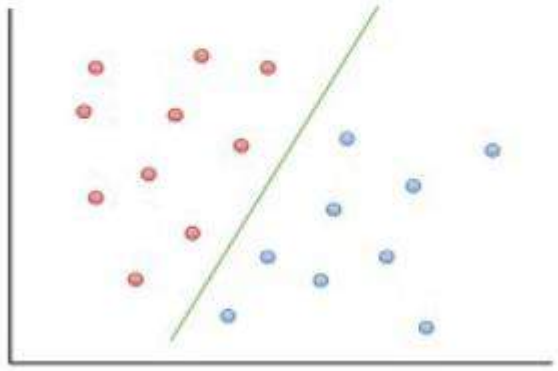
$$\mathbf{w}^T\mathbf{x} = 0$$

Hyperplane

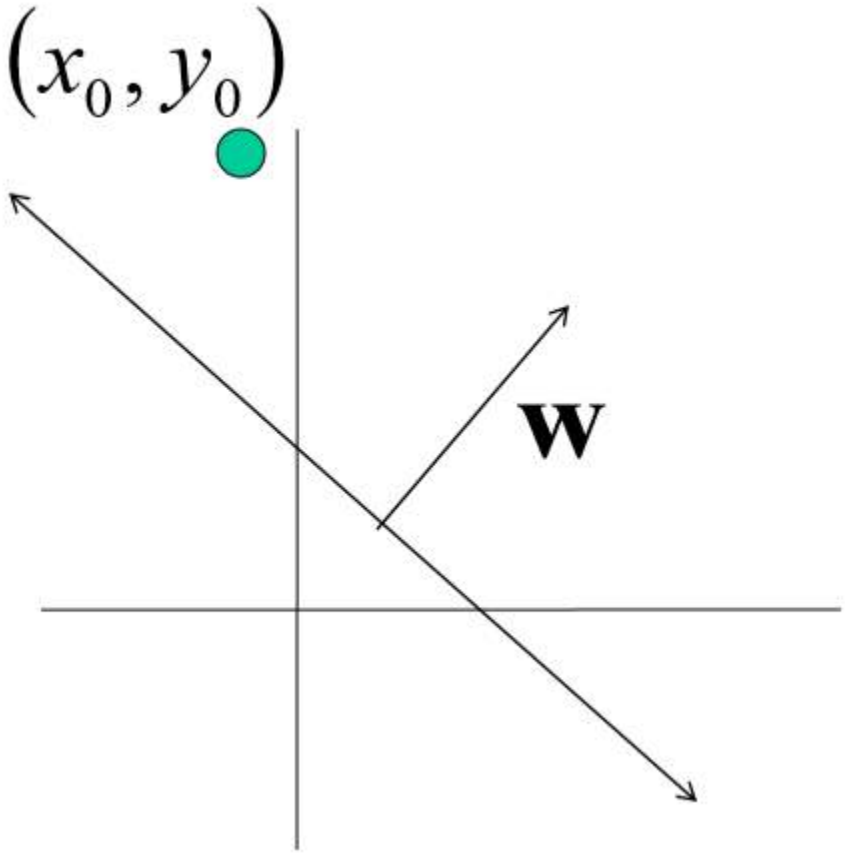
$$y = ax + b$$

Line





Line with 2 features: R2



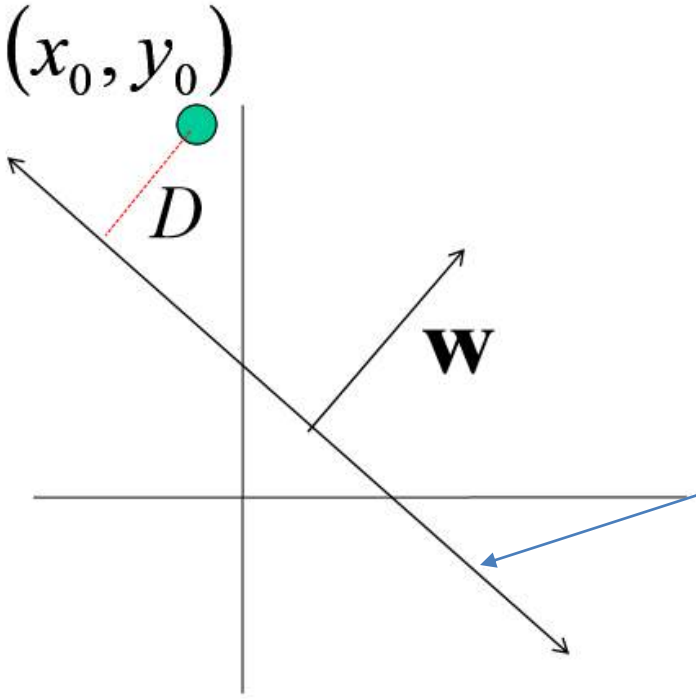
Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

$$\updownarrow$$

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

Line with 2 features: R2



Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

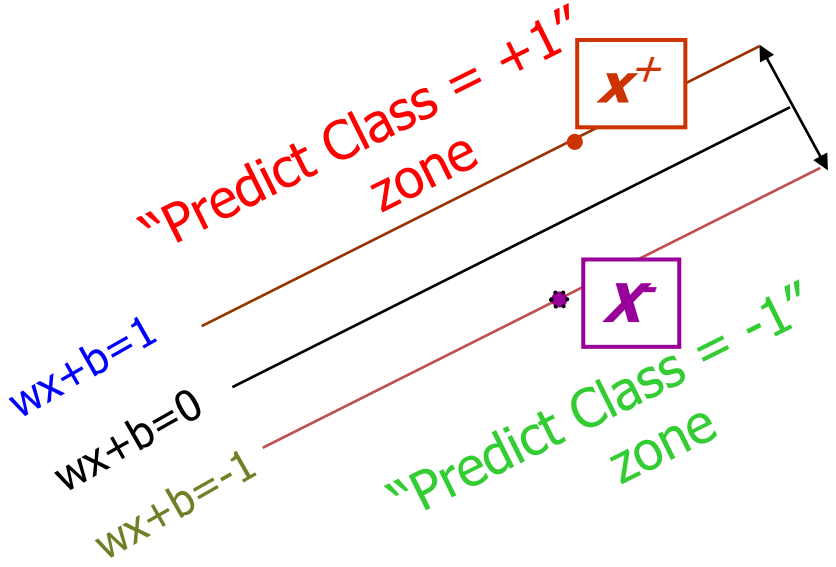
$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|}$$

distance from point to line

Kristen Grauman

Linear SVM Mathematically



M=Margin Width

What we know:

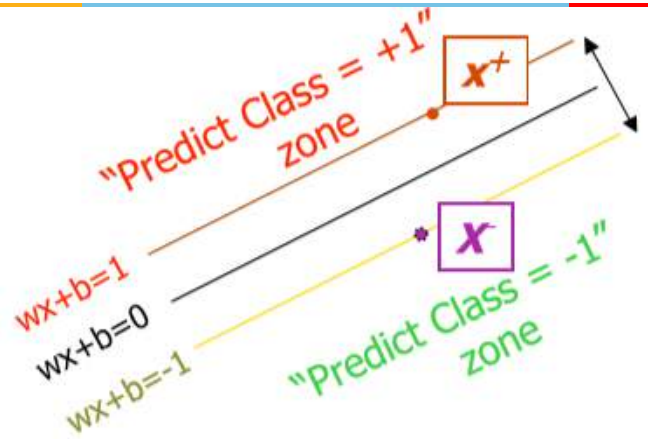
- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$

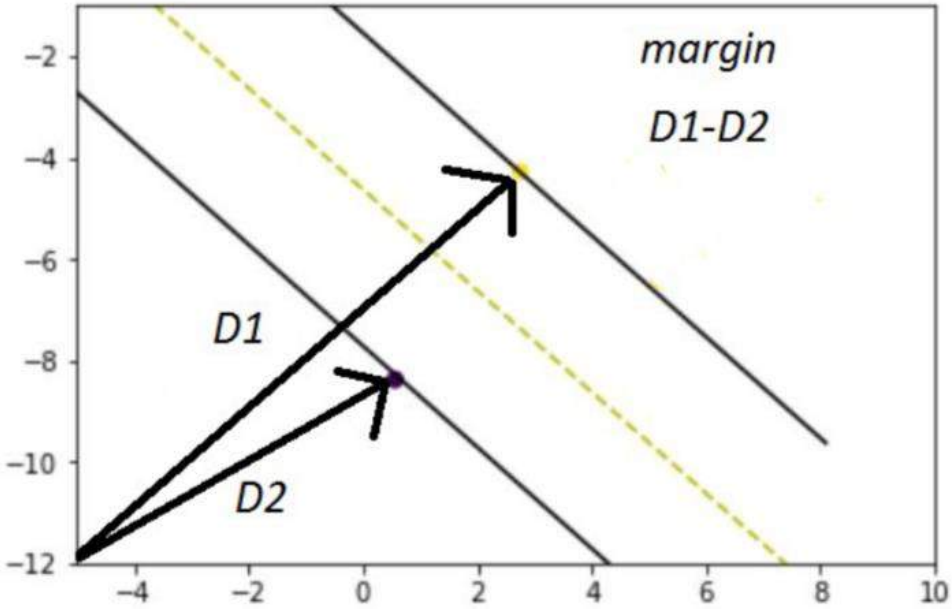
Distance between lines given by solving linear equation:

Maximize margin: $M = \frac{2}{||w||}$

Equivalent to minimize: $\frac{1}{2}||w||^2$

Linear SVM Mathematically





$D1 = w^T x+b=1$ $\quad w^T x+b-1=0$

$D2= w^T x+b=-1$ $\quad w^T x+b+1=0$

$w^T x+b-1 \; - \; w^T x+b+1$

⇩ *Solve algebraically*

$$\frac{2}{|w|}$$

Solving the Optimization Problem

1. Maximize margin $2/\|\mathbf{w}\|$
2. Correctly classify all training data points:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Quadratic optimization problem:

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

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

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

Solving the Optimization Problem

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

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

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

- **Need to optimize a *quadratic* function subject to *linear inequality* constraints.**
- **All constraints in SVM are linear**
- **Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.**

Optimization Problem

- Optimization problem is typically written:

$$\text{Minimize } f(x)$$

$$\text{subject to}$$

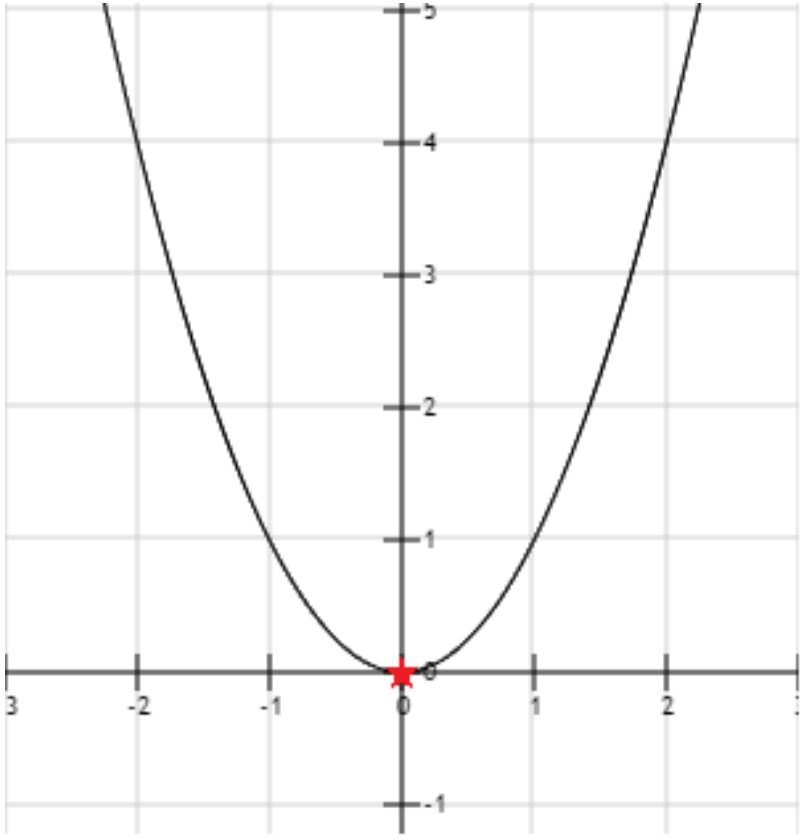
$$g_i(x) = 0, \quad i=1,\ldots,p$$

$$h_i(x) <= 0, \quad i=1,\ldots,m$$

- f(x) is called the objective function
- By changing x (the optimization variable) we wish to find a value $x*$ for which f(x) is at its minimum.
- p functions of g_i define equality constraints and
- m functions h_i define inequality constraints.
- The value we find MUST respect these constraints!

Unconstrained Optimization

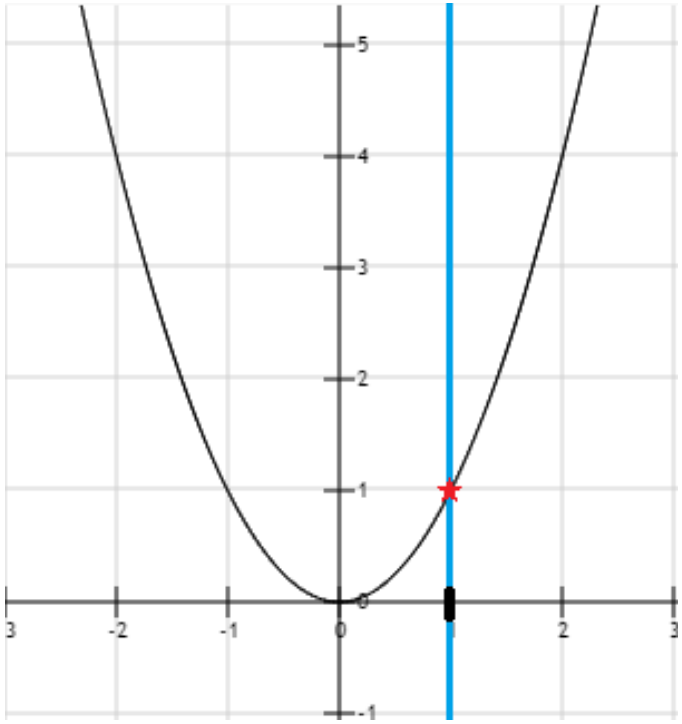
- Minimize x^2



Constrained Optimization -Equality Constraint

Minimize x^2

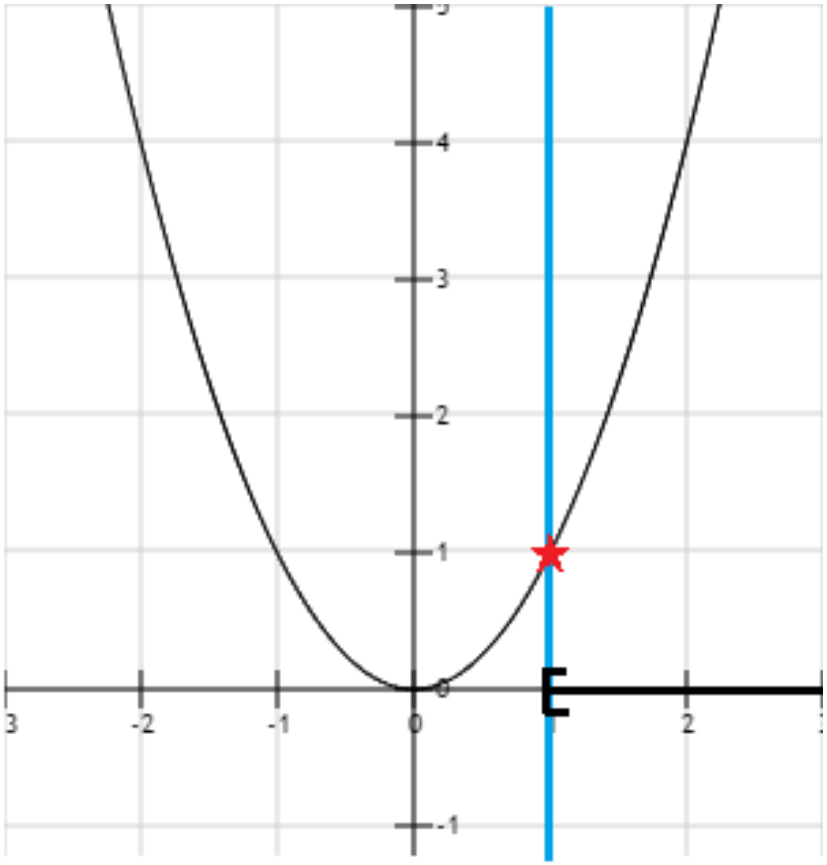
Subject to x = 1



Constrained Optimization -Inequality Constraint

Minimize x^2

Subject to x >= 1



Constrained optimization

- We can also have mix equality and inequality constraints together.
- Only restriction is that if we use contradictory constraints, we can end up with a problem which does not have a feasible set

Minimize x^2

Subject to

$x = 1$

$x < 0$

Impossible for x to be equal 1 and less than zero at the same

Constrained optimization

- A solution is an assignment of values to variables.
- A feasible solution is an assignment of values to variables such that all the constraints are satisfied.
- The objective function value of a solution is obtained by evaluating the objective function at the given solution.
- An optimal solution (assuming minimization) is one whose objective function value is less than or equal to that of all other feasible solutions.

Lagrange Multipliers

- **How do we find the solution to an optimization problem with constraints?**
- Constrained maximization (minimization) problem is rewritten as a Lagrange function whose optimal point is a saddle point, i.e. a global maximum (minimum)
- *Lagrange function use Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to constraints*

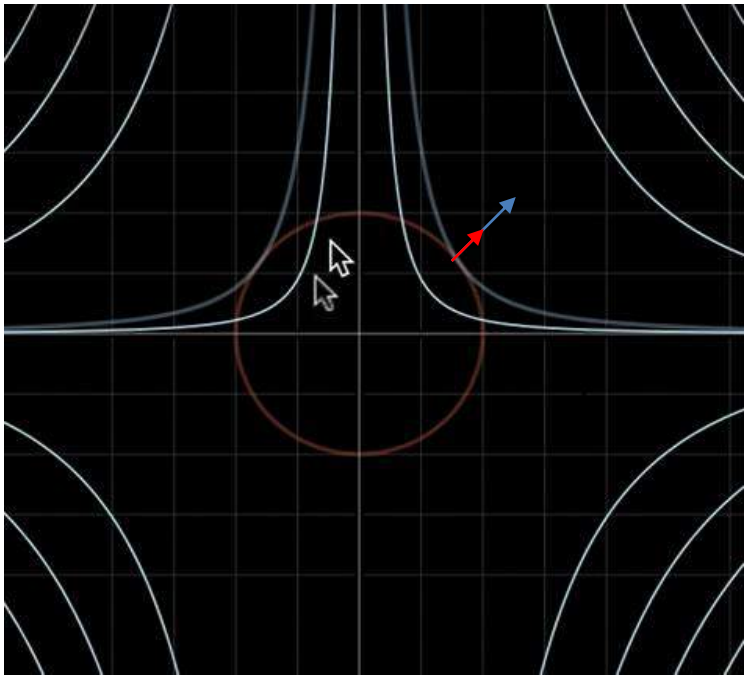
Constrained to Unconstrained Optimization: Lagrange Multiplier

Maximize

$$f(x,y) = x^2 y$$

Subject to

$$g(x, y) : x^2 + y^2 = 1$$



- Maximum of f(x,y) under constraint g(x, y) is obtained when their gradients point to same direction (when they are tangent to each other).
- Introduce a Lagrange multiplier λ for the equality constraint
- Mathematically,

$$\nabla f(x,y) = \lambda \nabla g(x,y)$$

Example:

$$\max_{x,y} xy \text{ subject to } x + y = 6$$

- Introduce a Lagrange multiplier λ for constraint
- Construct the Lagrangian

$$L(x, y) = xy - \lambda(x + y - 6)$$

- Stationary points

$$\frac{\partial L(x,y)}{\partial \lambda} = x + y - 6 = 0$$

$$\left.\begin{aligned} \frac{\partial L(x,y)}{\partial x} &= y - \lambda = 0 \\ \frac{\partial L(x,y)}{\partial y} &= x - \lambda = 0 \end{aligned}\right\} \Rightarrow x = y = \lambda$$

$$\Rightarrow x = y = 3$$

x and y values remain same even if you take $+\lambda$ or $-\lambda$ for equality constraint

2 x = 6
x = y = 3
λ = 3

Karush–Kuhn–Tucker (KKT) theorem

- KKT approach to nonlinear programming (quadratic) generalizes the method of Lagrange multipliers, which allows only equality constraints.
- KKT allows inequality constraints

Karush-Kuhn-Tucker (KKT) conditions

- Start with

  max f(x) subject to

$$g_i(x) = 0 \text{ and } h_j(x) \geq 0 \text{ for all } i, j$$

- Make the Lagrangian function

$$\mathcal{L} = f(x) - \sum_i \lambda_i g_i(x) - \sum_j \mu_j h_j(x)$$

- Take gradient and set to 0 – but other conditions also.

KKT conditions

- Make the Lagrangian function

$$\mathcal{L} = f(x) - \sum_i \lambda_i g_i(x) - \sum_j \mu_j h_j(x)$$

- Necessary conditions to have a minimum are

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0$$
$$g_i(x^*) = 0 \text{ for all } i$$
$$h_j(x^*) \geq 0 \text{ for all } j$$
$$\mu_j \geq 0 \text{ for all } j$$
$$\mu_j^* h_j(x^*) = 0 \text{ for all } j$$

KKT conditions

- Used to solve Inequality Constraints of the form

$h_j(\mathbf{x}) <= 0$ for j = 1,2,…,m

- Using Lagrangian

$$L = f(\mathbf{x}) + \sum_{i=1}^{q} \lambda_i h_i(\mathbf{x})$$

- With the constraints, called KKT conditions given as:

$$\frac{\partial L}{\partial x_i} = 0, \ \forall i = 1, 2, \ldots, d \qquad \lambda_i \geq 0, \ \forall i = 1, 2, \ldots, q$$

$$h_i(\mathbf{x}) \leq 0, \ \forall i = 1, 2, \ldots, q \qquad \lambda_i h_i(\mathbf{x}) = 0, \ \forall i = 1, 2, \ldots, q$$

Solving the Optimization Problem

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

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

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

- **Need to optimize a *quadratic* function subject to *linear inequality* constraints.**
- **The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primal problem**

Solving the Optimization Problem

- **The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:**

$$L(w, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \Sigma\, \alpha_i\, [y_i\, (\mathbf{w}^T \mathbf{x_i} + b) - 1]$$

- **Taking partial derivative with respect to w ,** $\frac{\partial L}{\partial w} = 0$
  - $w - \Sigma\, \alpha_i y_i\, \mathbf{x_i} = 0$
  - $w = \Sigma\, \alpha_i y_i\, \mathbf{x_i}$
- **Taking partial derivative with respect to b,** $\frac{\partial L}{\partial b} = 0$
  - $- \Sigma\, \alpha_i\, y_i = 0$
  - $\Sigma\, \alpha_i\, y_i = 0$

Solving the Optimization Problem

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

- Expanding above equation:

$$L(w, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \Sigma\, \alpha_i y_i\, \mathbf{w} \cdot \mathbf{x_i} + \Sigma\, \alpha_i y_i\, b + \Sigma\, \alpha_i$$

- Substituting $w = \Sigma\, \alpha_i y_i\, \mathbf{x_i}$ and $\Sigma\, \alpha_i\, y_i = 0$ in above equation

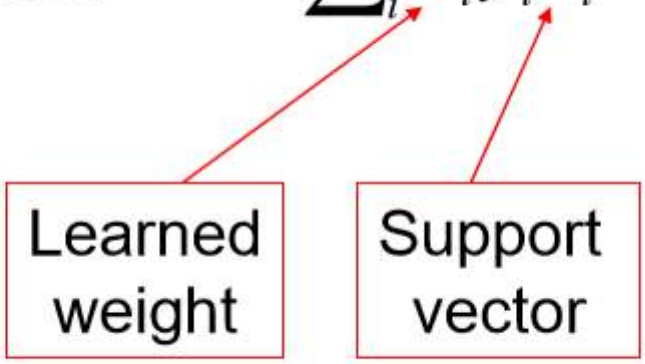
$$L(w, b, \alpha_i) = \frac{1}{2}(\sum_i \alpha_i y_i\, \mathbf{x_i})(\sum_j \alpha_j y_j\, \mathbf{x_j}) - (\sum_i \alpha_i y_i\, \mathbf{x_i})(\sum_j \alpha_j y_j\, \mathbf{x_j}) + \Sigma\, \alpha_i$$

$$L(w, b, \alpha_i) = \Sigma\, \alpha_i - \frac{1}{2}(\sum_i \alpha_i y_i\, \mathbf{x_i})(\sum_j \alpha_j y_j\, \mathbf{x_j})$$

$$L(w, b, \alpha_i) = \Sigma\, \alpha_i - \frac{1}{2}(\sum_i \sum_j \alpha_i\, \alpha_j y_i\, y_j\, \mathbf{x_i} \cdot \mathbf{x_j})$$

Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$



Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

  $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$ (for any support vector)

- Classification function:

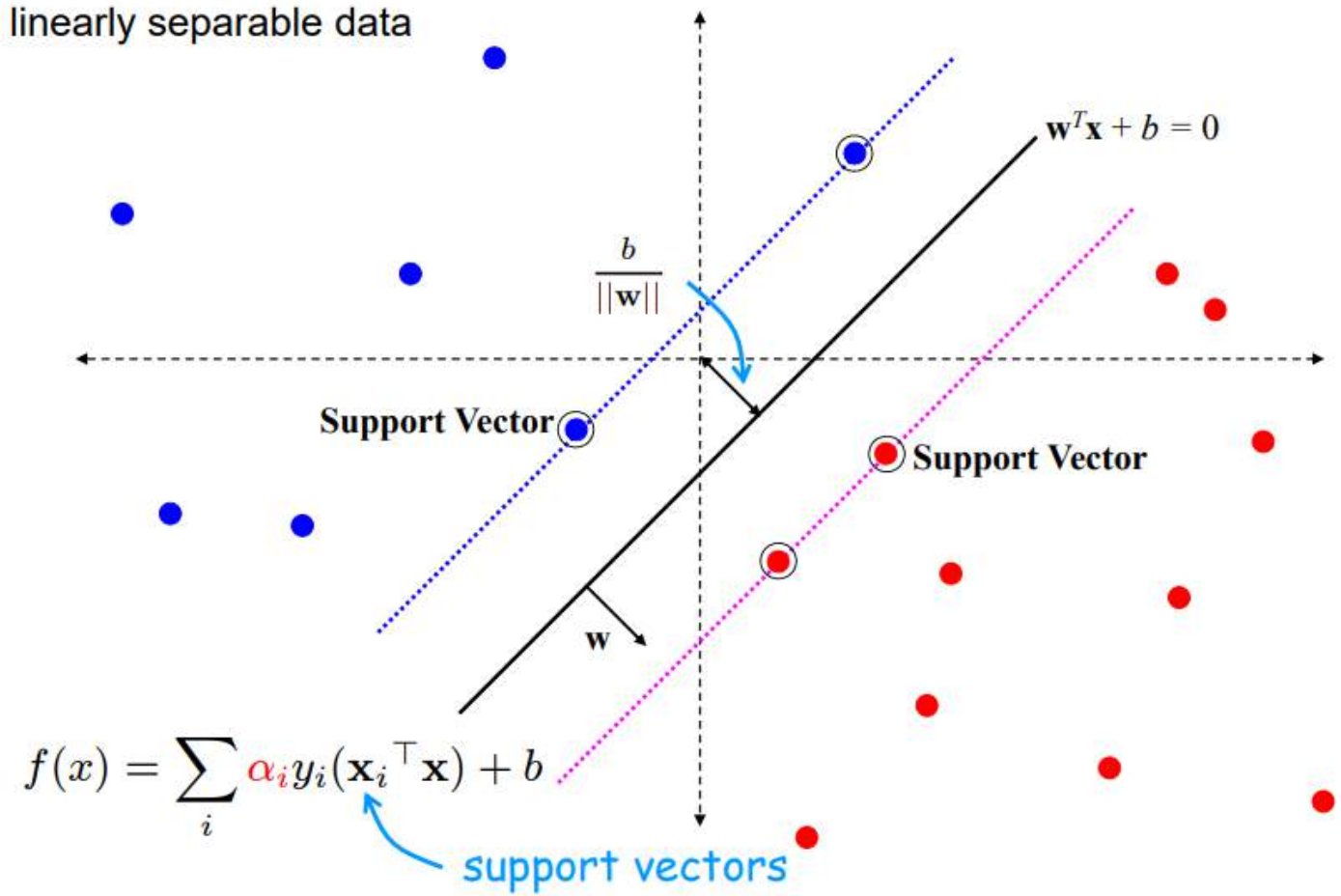
$$f(x) = \text{sign}\,(\mathbf{w} \cdot \mathbf{x} + \text{b})$$

$$= \text{sign}\left(\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b\right)$$

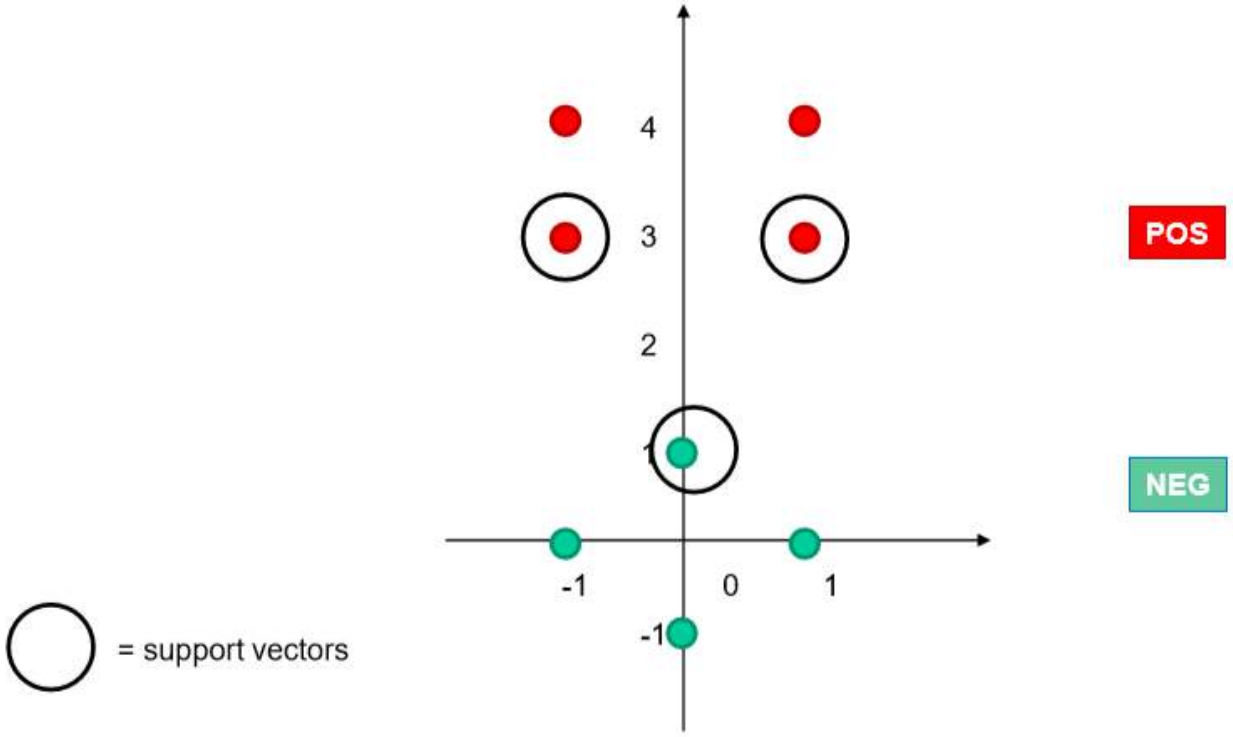
If f(x) < 0, classify as negative, otherwise classify as positive.

- Notice that it relies on an *inner product* between the test point ***x*** and the support vectors $\boldsymbol{x}_i$
- (Solving the optimization problem also involves computing the inner products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ between all pairs of training points)

Substituting w in support vectors function



Example



Example adapted from Dan Ventura

Solving for α

- We know that for the support vectors, f(x) = 1 or -1 exactly
- Add a 1 in the feature representation for the bias
- The support vectors have coordinates and labels:
  - x1 = [0 1 1], y1 = -1
  - x2 = [-1 3 1], y2 = +1
  - x3 = [1 3 1], y3 = +1
- Thus we can form the following system of linear equations:

Solving for α

- System of linear equations:

α1 y1 dot(x1, x1) + α2 y2 dot(x1, x2) + α3 y3 dot(x1, x3) = y1

α1 y1 dot(x2, x1) + α2 y2 dot(x2, x2) + α3 y3 dot(x2, x3) = y2

α1 y1 dot(x3, x1) + α2 y2 dot(x3, x2) + α3 y3 dot(x3, x3) = y3

-2 * α1 + 4 * α2 + 4 * α3 = -1

-4 * α1 + 11 * α2 + 9 * α3 = +1

-4 * α1 + 9 * α2 + 11 * α3 = +1

- Solution: α1 = 3.5, α2 = 0.75, α3 = 0.75

Solving for w and b

We know $w = \alpha_1 y_1 x_1 + \ldots + \alpha_N y_N x_N$ where N = # SVs

Thus w = -3.5 * [0 1 1] + 0.75 [-1 3 1] + 0.75 [1 3 1] = [0 1 -2]

Separating out weights and bias, we have: w = [0 1] and b = -2

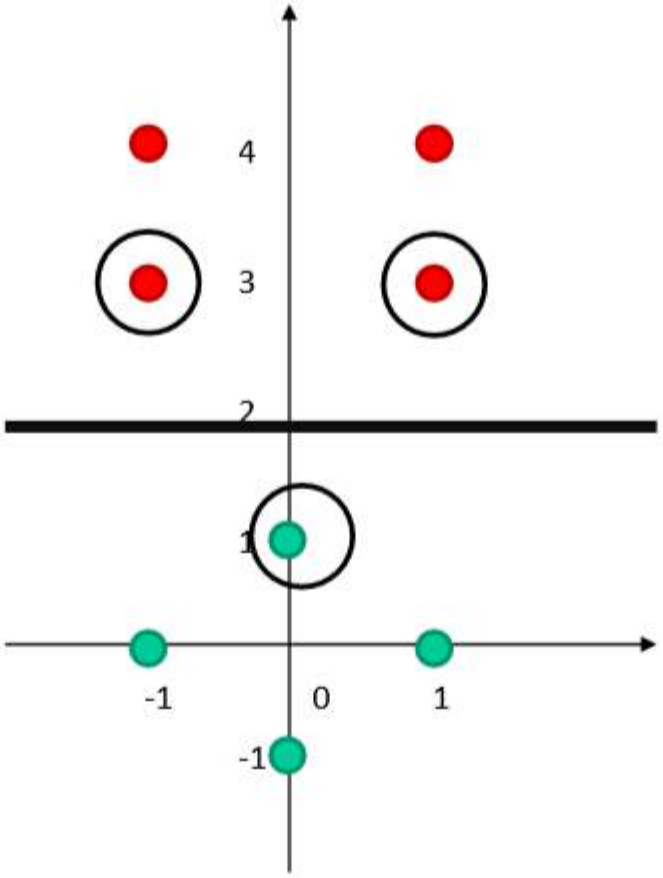
For SVMs, we used this eq for a line: ax + cy + b = 0 where w = [a c]

Thus ax + b = -cy ➔ y = (-a/c) x + (-b/c)

Thus y-intercept is -(-2)/1 = 2

The decision boundary is perpendicular to w and it has slope -0/1 = 0

Decision boundary



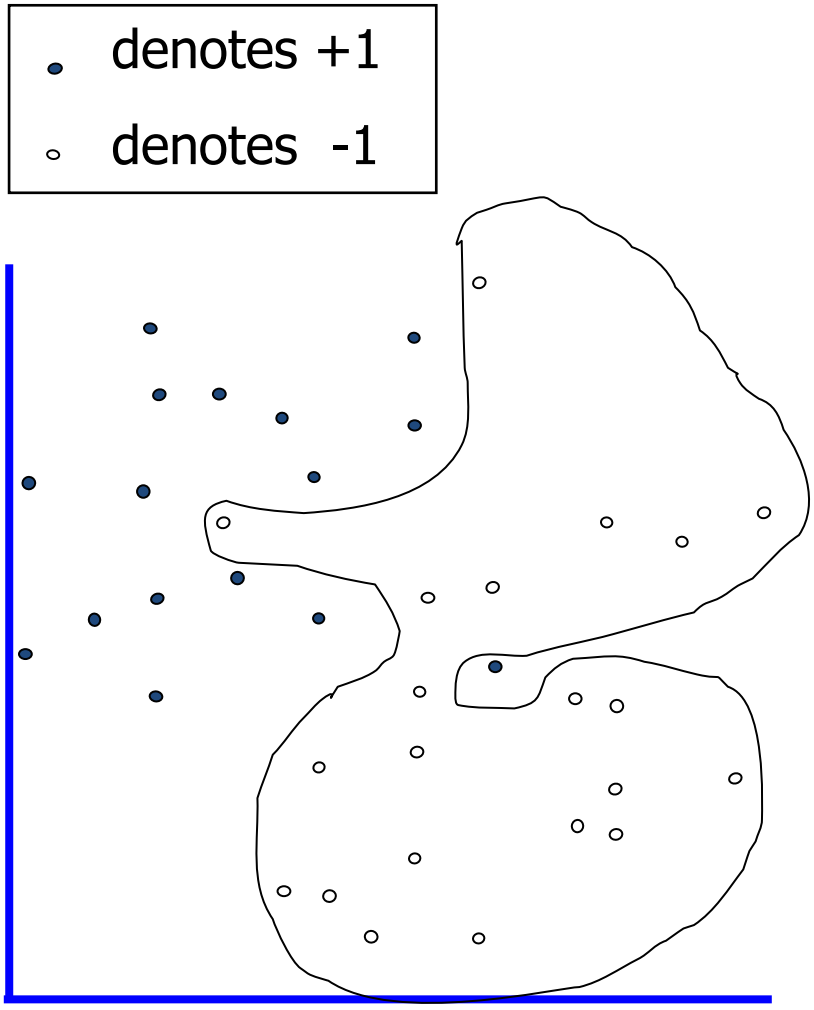
POS

DECISION BOUNDARY

NEG

= support vectors

Dataset with noise



- **Hard Margin: So far we require all data points be classified correctly**

  - No training error

- **What if the training set is noisy?**

Soft Margin Classification

Slack variables ξ_i **can be added to allow misclassification of difficult or noisy examples.**

What should our quadratic optimization criterion be?

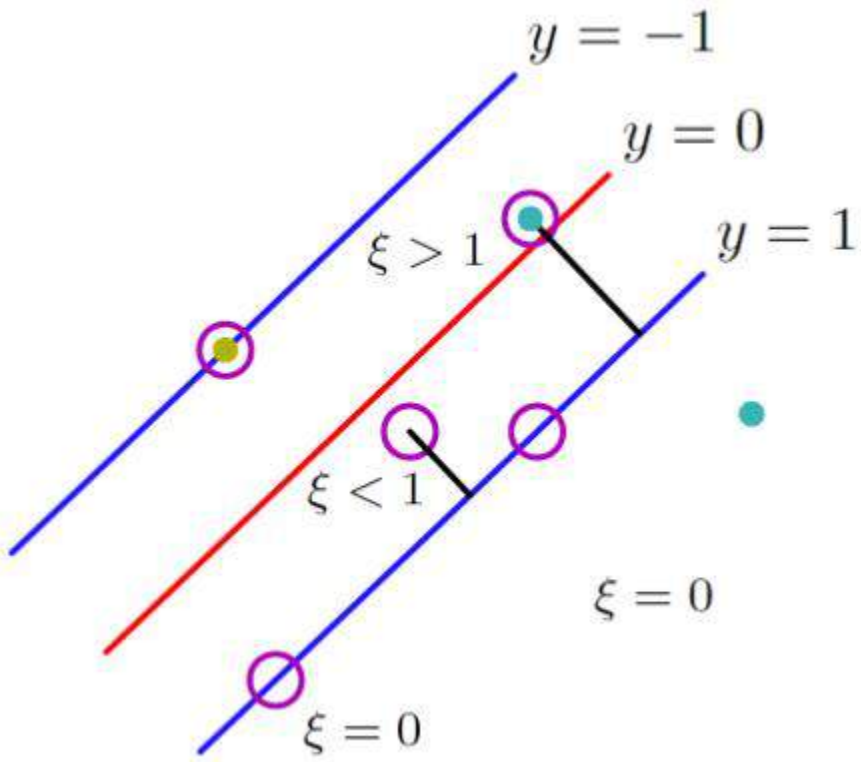
Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

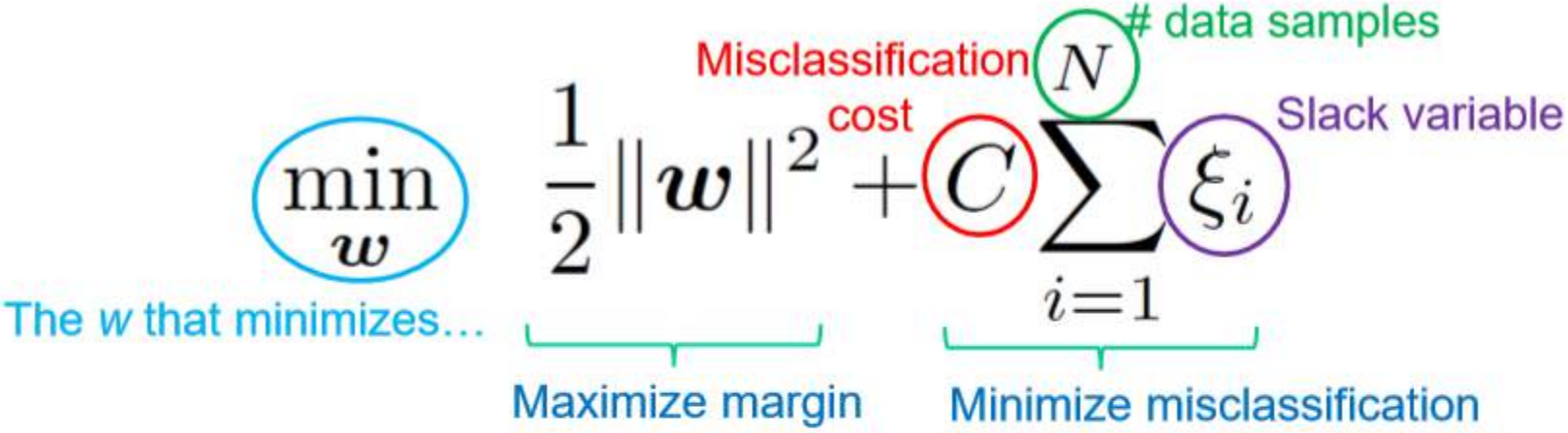
Slack Variable

- **Slack variable** as giving the classifier some leniency when it comes to moving around points near the **margin**.
- When C is large, larger slacks penalize the objective function of SVM's more than when C is small.

Soft margin example



Soft Margin



$$\text{subject to} \quad y_i \boldsymbol{w}^T \boldsymbol{x}_i \geq 1 - \xi_i,$$

$$\xi_i \geq 0, \quad \forall i = 1, \ldots, N$$

Hard Margin versus Soft Margin

- **Hard Margin:**

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

$\Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

$y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

- **Soft Margin incorporating slack variables:**

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

$\Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum \xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

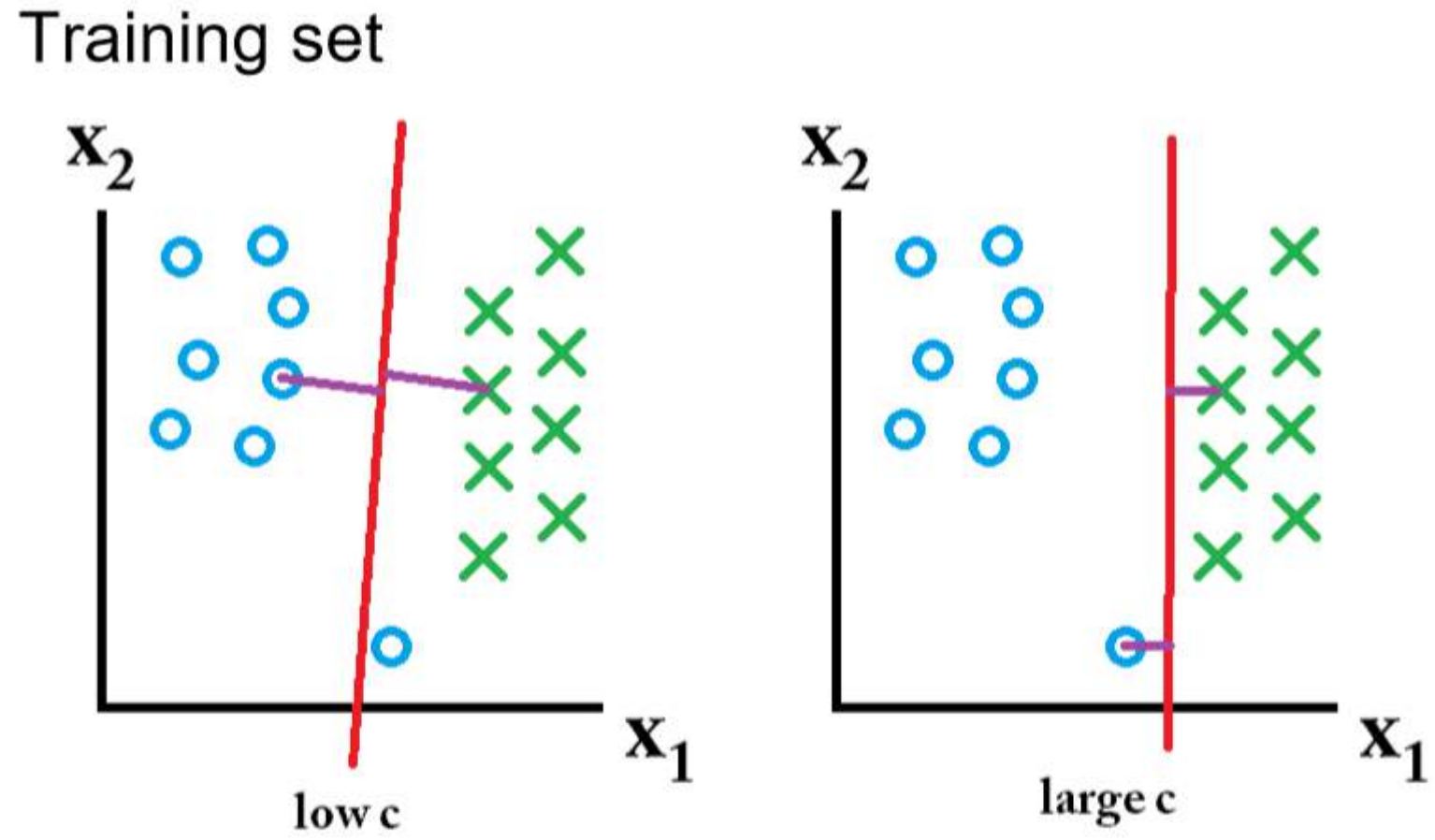
$y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- **Parameter *C* can be viewed as a way to control overfitting.**

Value of C parameter

- C parameter tells the SVM optimization how much you want to avoid misclassifying each training example.
- For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
- Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

Effect of Margin size v/s misclassification cost

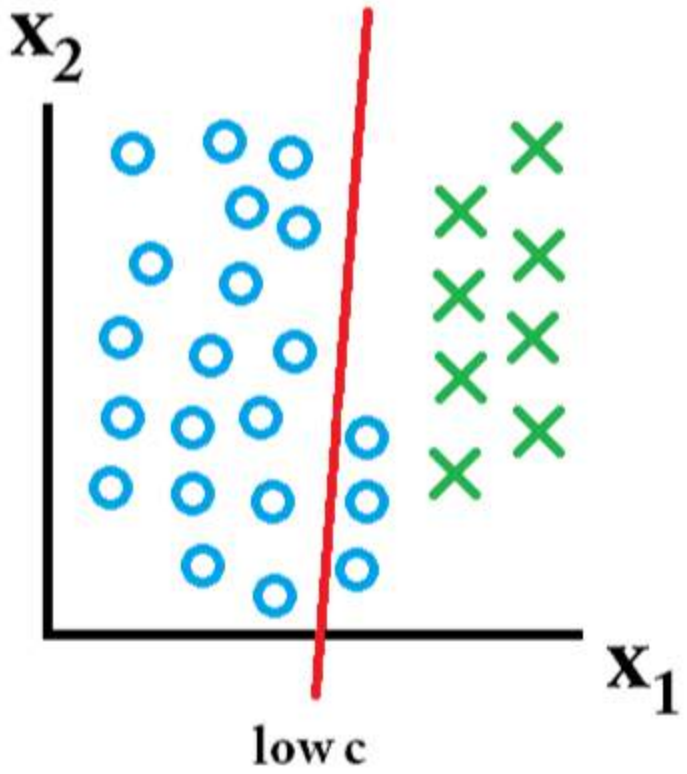


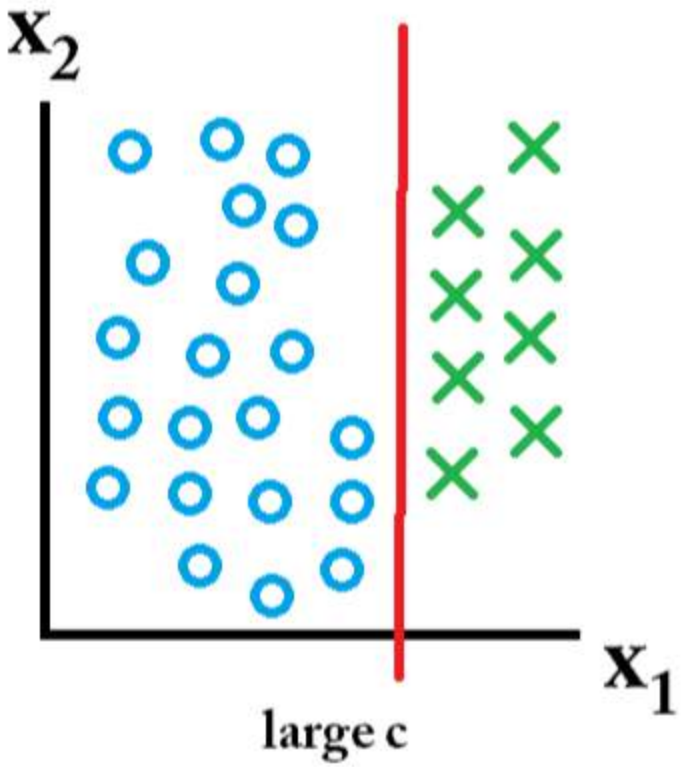
Misclassification ok, want large margin

Misclassification not ok

Effect of Margin size v/s misclassification cost

Including test set A



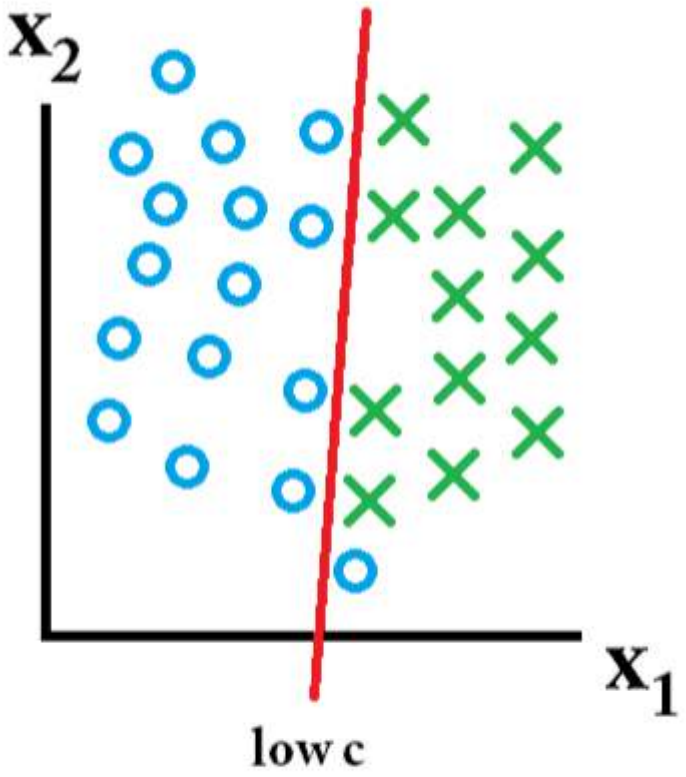


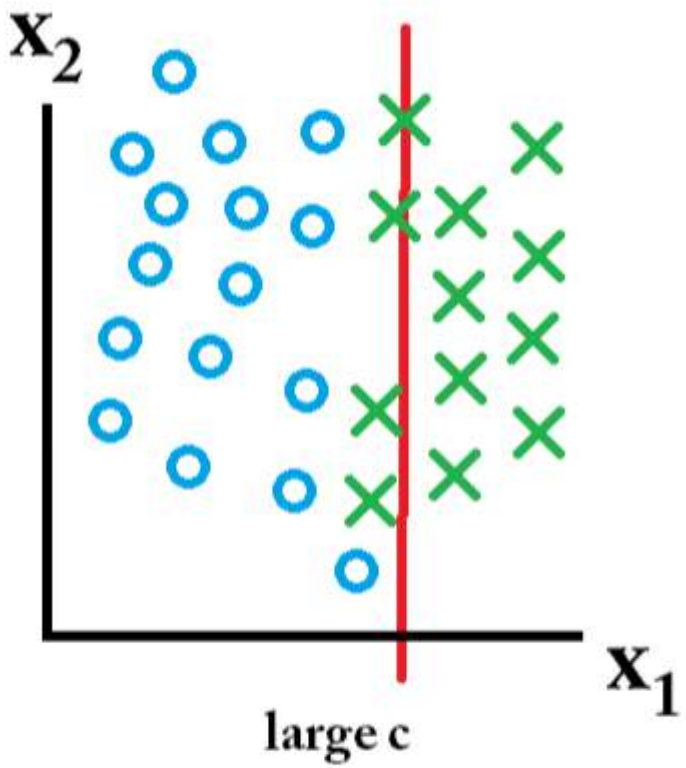
Misclassification ok, want large margin

Misclassification not ok

Effect of Margin size v/s misclassification cost

Including test set B





Misclassification ok, want large margin

Misclassification not ok

Linear SVMs: Overview

- The classifier is a *separating hyperplane.*
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points x_i are support vectors with non-zero Lagrangian multipliers α_i.

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

Good Web References for SVM

SVM

- **Text categorization with Support Vector Machines: learning with many relevant features** - T. Joachims, ECML
- **A Tutorial on Support Vector Machines for Pattern Recognition**, Kluwer Academic Publishers - Christopher J.C. Burges
- http://www.cs.utexas.edu/users/mooney/cs391L/
- https://www.coursera.org/learn/machine-learning/home/week/7
- https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
- https://data-flair.training/blogs/svm-kernel-functions/
- MIT 6.034 Artificial Intelligence, Fall 2010
- https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior
- https://www.sciencedirect.com/science/article/abs/pii/S089360800600279 6
- https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be
- Radial basis kernel

Thank You

Support Vector Machines

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

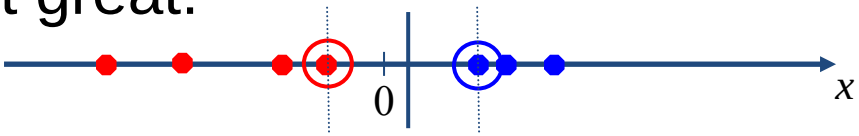
These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Topics to be covered

- Nonlinear SVM
- Kernel Trick
- SVM Kernels
- Multi-Class Problem
- SVM vs Logistic Regression
- SVM Applications

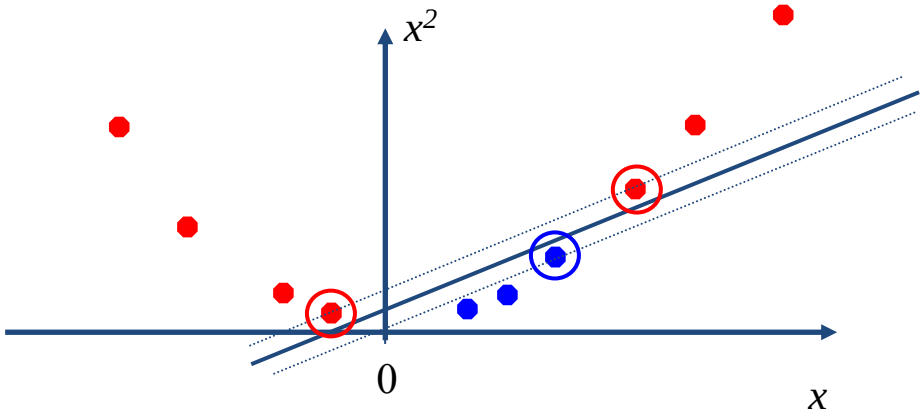
Non-linear SVMs

- Datasets that are linearly separable with some noise soft margin 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:



The "Kernel Trick"

- The linear classifier relies on dot product between vectors
  - $x_i^T \cdot x_j$
- If every data point is mapped into high-dimensional space via some transformation Φ: $x \rightarrow \varphi(x)$, the dot product becomes:

$$K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$$

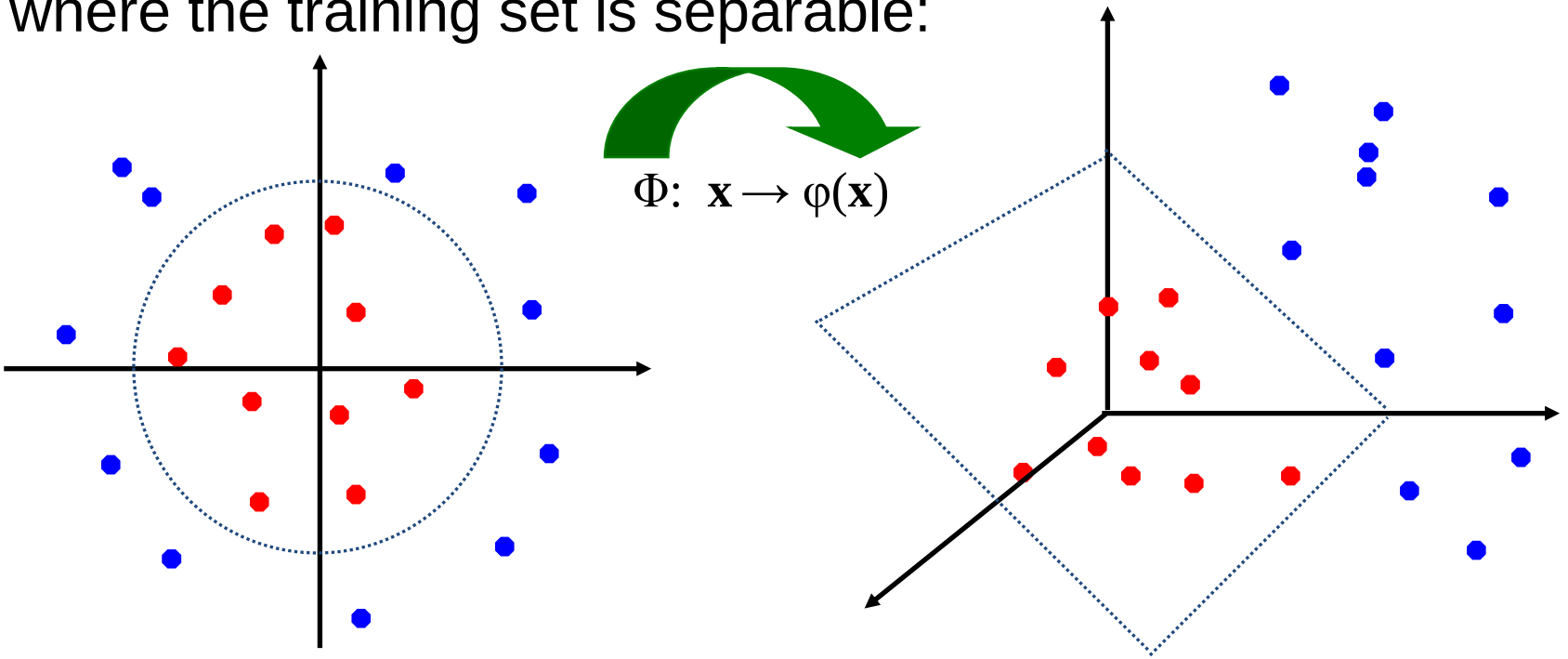
- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.

SVM Kernels

- SVM algorithms use a set of mathematical functions that are defined as the kernel.
- Function of kernel is to take data as input and transform it into the required form.
- Different SVM algorithms use different types of kernel functions. Example *linear, nonlinear, polynomial, and sigmoid etc.*

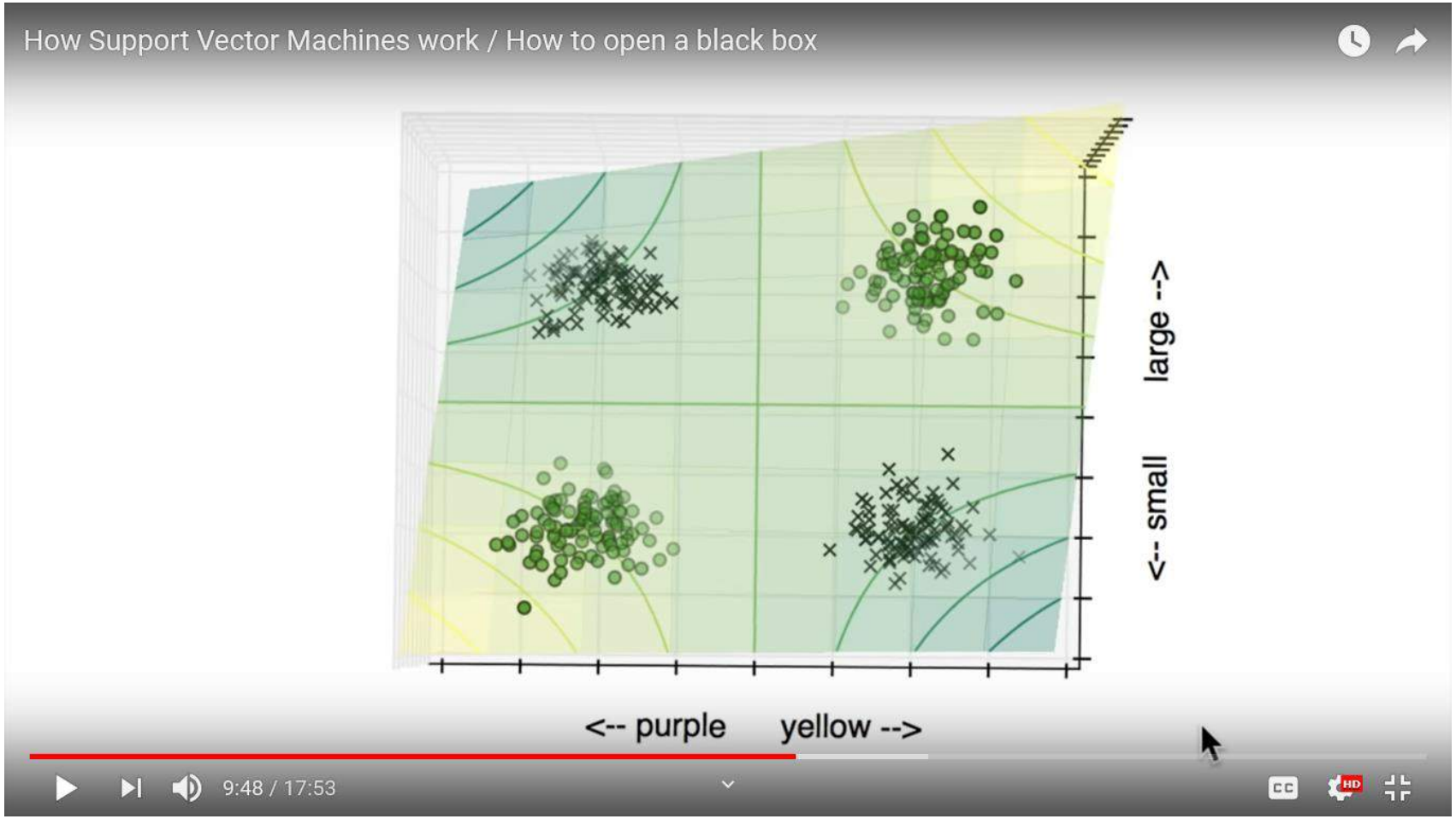
Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:

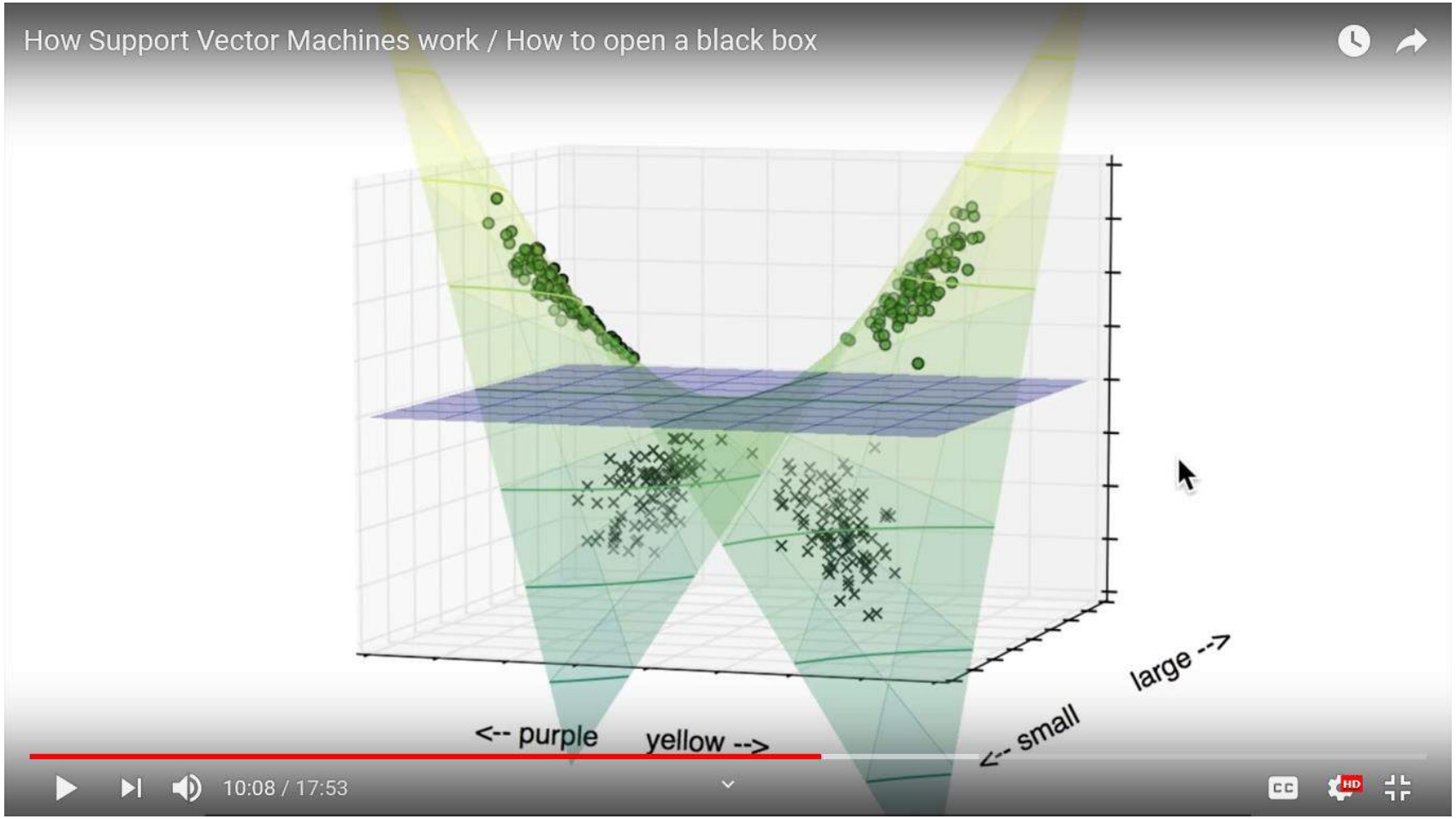


Non-linear SVMs

Non-linear SVMs: Feature spaces



Non-linear SVMs: Feature spaces



What Functions are Kernels?

- For some functions $K(x_i,x_j)$ checking that $K(x_i,x_j)= \varphi(x_i)^T \varphi(x_j)$ can be cumbersome.

- Mercer's theorem:

  Every positive-semidefinite symmetric function is a kernel

What Functions are Kernels?

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *distance function*, i.e.
  - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') = 0$ only for $x = x'$,
  - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

  then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

2) We can *construct kernels from other kernels*:

- if k is a kernel and $\alpha > 0$, then αk and $k + \alpha$ are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i,\mathbf{x}_j)= \mathbf{x}_i^T\mathbf{x}_j$
- Polynomial of power p: $K(\mathbf{x}_i,\mathbf{x}_j)= (1+ \mathbf{x}_i^T\mathbf{x}_j)^p$
- Gaussian (radial-basis function network):

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

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

Non-linear SVMs Mathematically

- **The solution is:**

$$f(\mathbf{x}) = \Sigma \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b$$

- **Optimization techniques for finding α_i's remain the same!**

Non-linear SVM using kernel

1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Multi-Class Problem

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

Multi-Class Problem

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

Multi-Class Problem

One-vs-all (a.k.a. one-vs-others)

- Train C classifiers
- In each, pos = data from class *i*, neg = data from classes other than *i*
- The class with the most confident prediction wins
- Example:
  - You have 4 classes, train 4 classifiers
  - 1 vs others: score 3.5
  - 2 vs others: score 6.2
  - 3 vs others: score 1.4
  - 4 vs other: score 5.5
  - Final prediction: class 2
- Issues?

Multi-Class Problem

One-vs-one (a.k.a. all-vs-all)

- Train C(C-1)/2 binary classifiers (all pairs of classes)
- They all vote for the label
- Example:
  - You have 4 classes, then train 6 classifiers
  - 1 vs 2, 1 vs 3, 1 vs 4, 2 vs 3, 2 vs 4, 3 vs 4
  - Votes: 1, 1, 4, 2, 4, 4
  - Final prediction is class 4

SVM versus Logistic Regression

- When viewed from the point of view of regularized empirical loss minimization, SVM and logistic regression appear quite similar:

$$\text{SVM:} \quad \sum_{i=1}^{n} \left(1 - y_i \left[w_0 + \mathbf{x}_i^T \mathbf{w}_1\right]\right)^+ + \|\mathbf{w}_1\|^2/2$$

$$\text{Logistic:} \quad \sum_{i=1}^{n} \overbrace{-\log \sigma\left(y_i \left[w_0 + \mathbf{x}_i^T \mathbf{w}_1\right]\right)}^{-\log P(y_i|\mathbf{x},\mathbf{w})} + \|\mathbf{w}_1\|^2/2$$

where $\sigma(z) = (1 + \exp(-z))^{-1}$ is the logistic function.

SVM versus Logistic Regression

- The difference comes from how we penalize "errors":

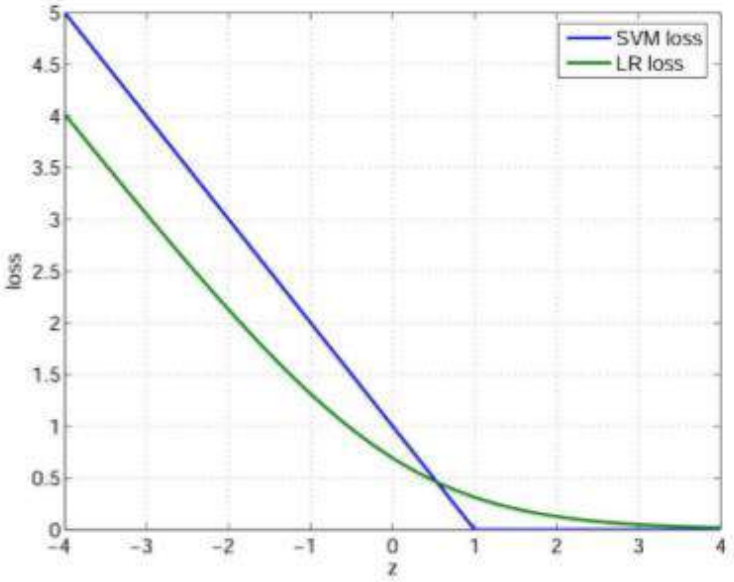
$$\text{Both:} \quad \sum_{i=1}^{n} \text{Loss}\Big(\overbrace{y_i\left[w_0 + \mathbf{x}_i^T \mathbf{w}_1\right]}^{z}\Big) + \quad \|\mathbf{w}_1\|^2/2$$

- SVM:

$$\text{Loss}(z) = (1-z)^+$$

- Regularized logistic reg:

$$\text{Loss}(z) = \log(1+\exp(-z))$$

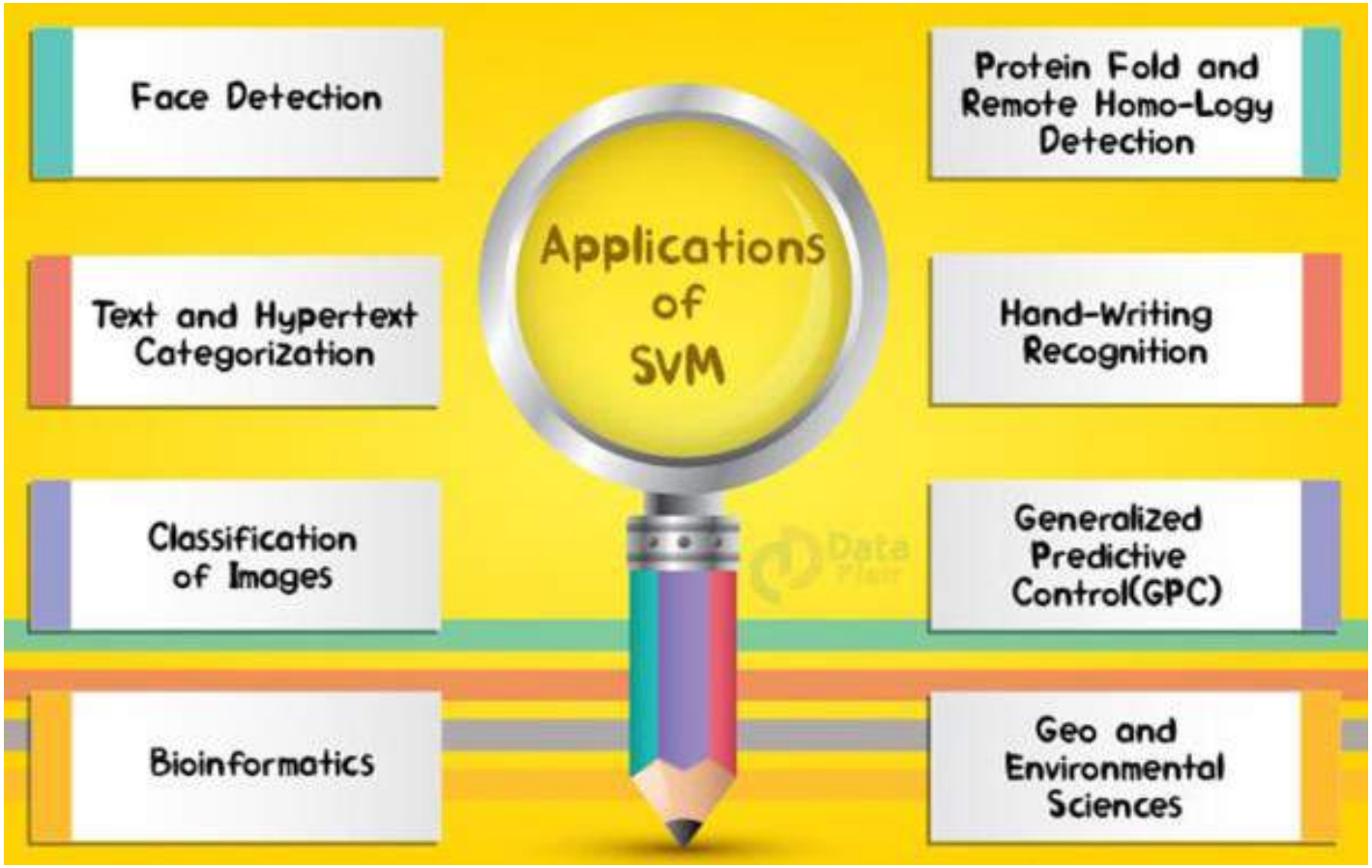


Properties of SVM

- Flexibility in choosing a similarity function
- Sparseness of solution when dealing with large data sets
  - Only support vectors are used to specify the separating hyperplane
  - Therefore SVM also called sparse kernel machine.
- Ability to handle large feature spaces
  - complexity does not depend on the dimensionality of the feature space
- Overfitting can be controlled by soft margin approach
- Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution
- Feature Selection

SVM Applications

SVM has been used successfully in many real-world problems



Application : Text Categorization

- Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content.

  A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category

Text Categorization using SVM

- The distance between two documents is $\phi(x)\cdot\phi(z)$
- $K(x,z) = \phi(x)\cdot\phi(z)$ is a valid kernel, SVM can be used with $K(x,z)$ for discrimination.
- Why SVM?

  -High dimensional input space

  -Few irrelevant features (dense concept)

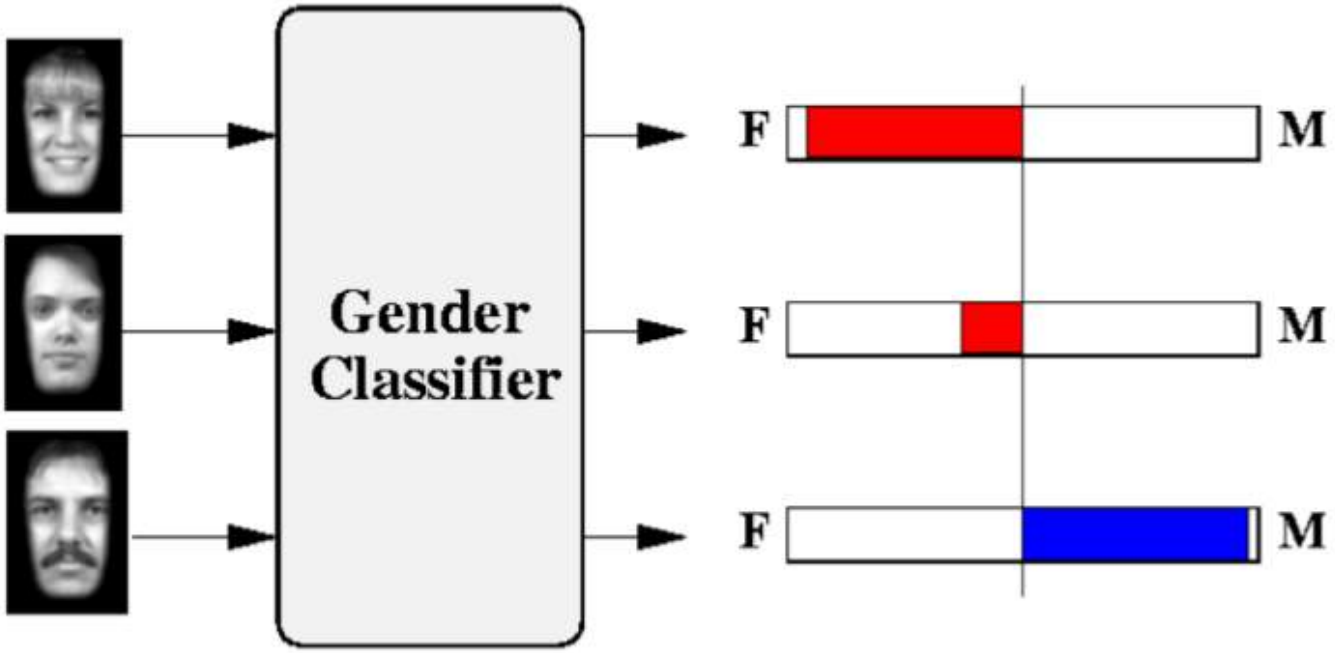
  -Sparse document vectors (sparse instances)

  -Text categorization problems are linearly separable

Using SVM

1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

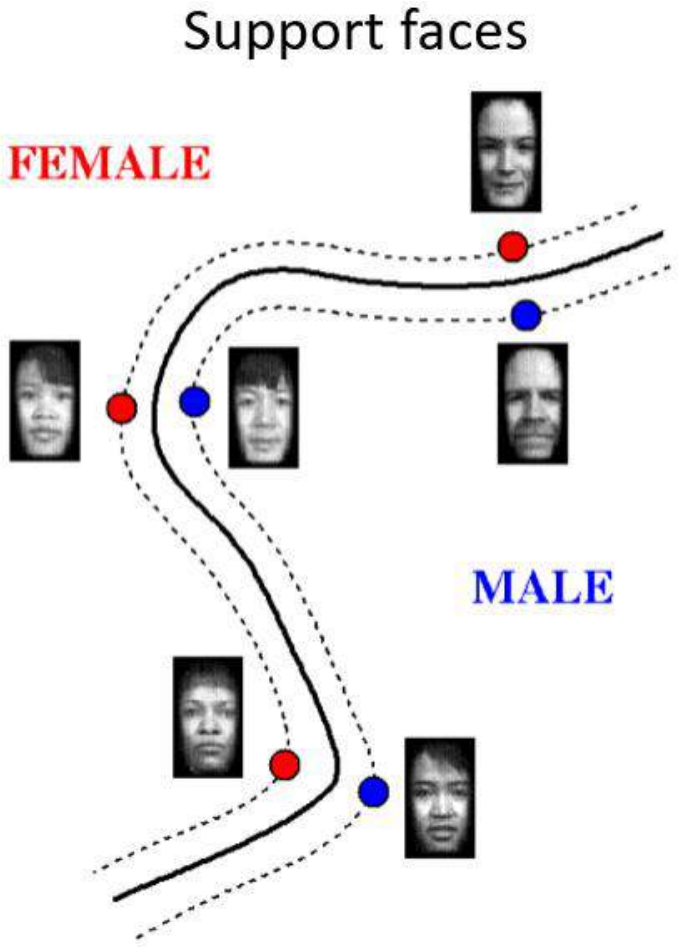
Learning Gender from image with SVM



Moghaddam and Yang, Learning Gender with Support Faces, TPAMI 2002

Moghaddam and Yang, Face & Gesture 2000

Support faces



Accuracy of SVM Classifier

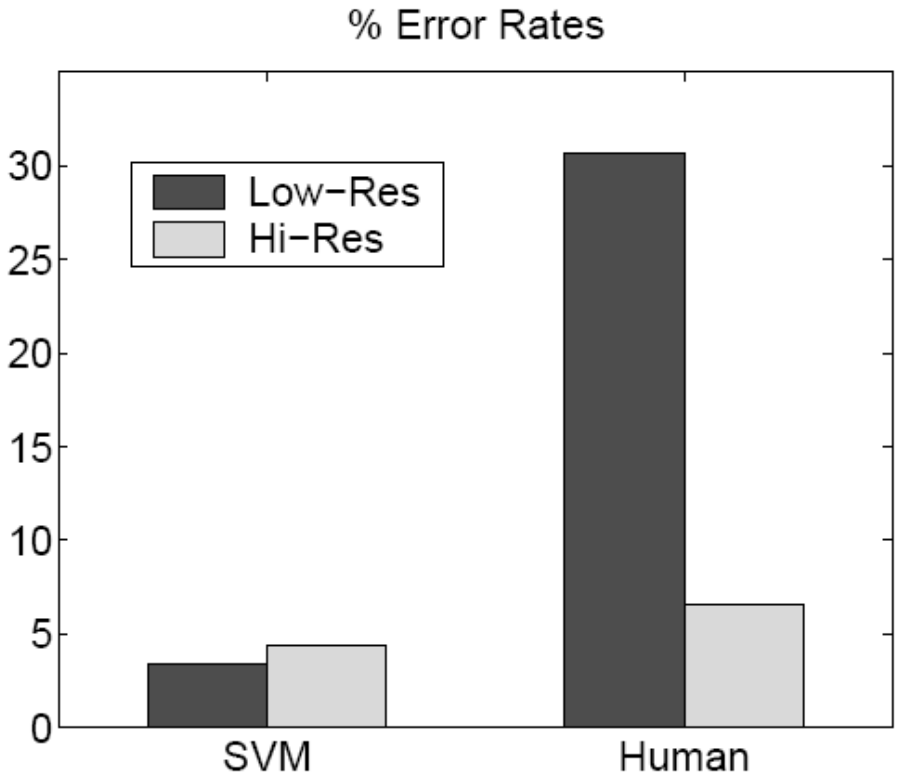


Figure 6. SVM vs. Human performance

- SVMs performed better than humans, at either resolution

Some Issues

- **Sensitive to noise**
  - A relatively small number of mislabeled examples can dramatically decrease the performance
- **Choice of kernel**
  - Gaussian or polynomial kernel is default
  - if ineffective, more elaborate kernels are needed
  - domain experts can give assistance in formulating appropriate similarity measures
- **Choice of kernel parameters**
  - e.g. σ in Gaussian kernel
  - σ is the distance between closest points with different classifications
  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.
- **Optimization criterion** – Hard margin v.s. Soft margin
  - a lengthy series of experiments in which various parameters are tested

Good Web References for SVM

SVM

- **Text categorization with Support Vector Machines: learning with many relevant features** - T. Joachims, ECML
- **A Tutorial on Support Vector Machines for Pattern Recognition**, Kluwer Academic Publishers - Christopher J.C. Burges
- http://www.cs.utexas.edu/users/mooney/cs391L/
- https://www.coursera.org/learn/machine-learning/home/week/7
- https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
- https://data-flair.training/blogs/svm-kernel-functions/
- MIT 6.034 Artificial Intelligence, Fall 2010
- https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior
- https://www.sciencedirect.com/science/article/abs/pii/S0893608006002796
- https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be
- Radial basis kernel
- http://www.engr.mun.ca/~baxter/Publications/LagrangeForSVMs.pdf

Thank You

Unsupervised Learning

BITS Pilani
Pilani Campus

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

innovate achieve lead

BITS Pilani

Pilani Campus

Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Bishop and many others who made their course materials freely available online.

Topics to be covered

Ref: Christopher Bishop: Chapter 9

- Unsupervised learning
- Clustering
- K-means Clustering
- Gaussian Mixture Models
- EM algorithm

Unsupervised Learning

- We only use the features X, not the labels Y
- This is useful because we may not have any labels but we can still detect patterns
- For example:
  - We can detect that news articles revolve around certain topics, and group them accordingly
  - Discover a distinct set of objects appear in a given environment, even if we don't know their names, then ask humans to label each group
  - Identify health factors that correlate with a disease

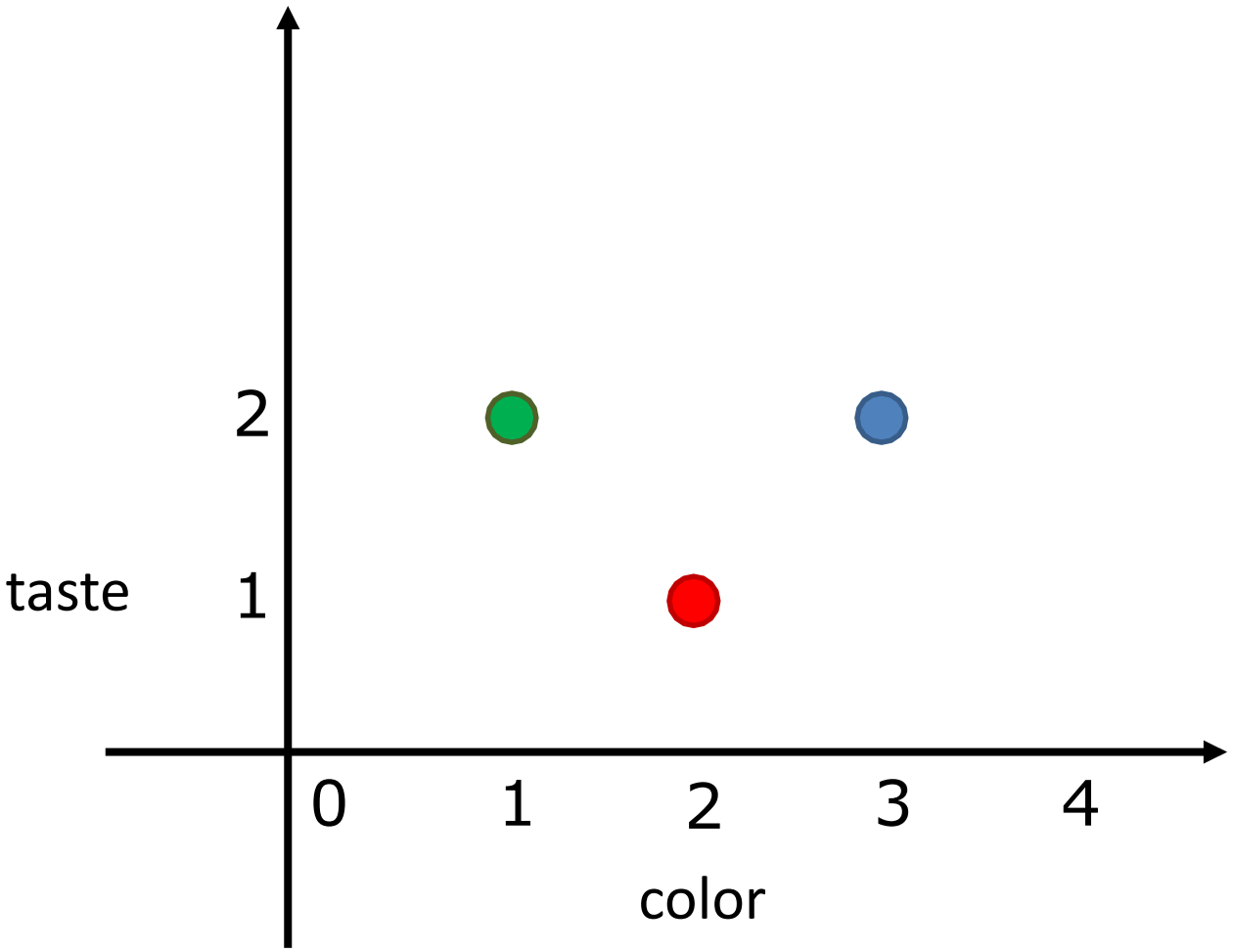
What is clustering?

- Grouping items that “belong together” (i.e. have similar features)

Feature representation (x)

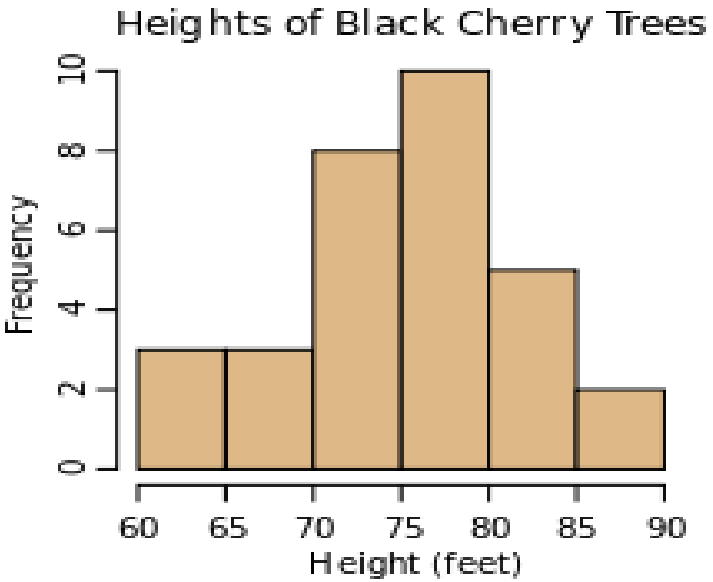
- A vector representing measurable characteristics of a data sample we have
- E.g. a glass of juice can be represented via its color = {yellow=1, red=2, green=3, purple=4} and taste = {sweet=1, sour=2}
- For a given glass i, this can be represented as a vector: xi = [3 2] represents sour green juice
- For D features, this defines a D-dimensional space where we can measure similarity between samples

Feature representation (x)



Why do we cluster?

- Counting
  - Feature histograms: by grouping similar features and counting how many of each a data sample has
- Summarizing data
  - Look at large amounts of data

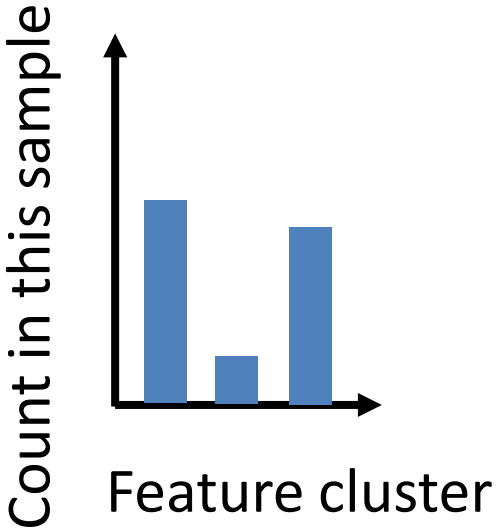


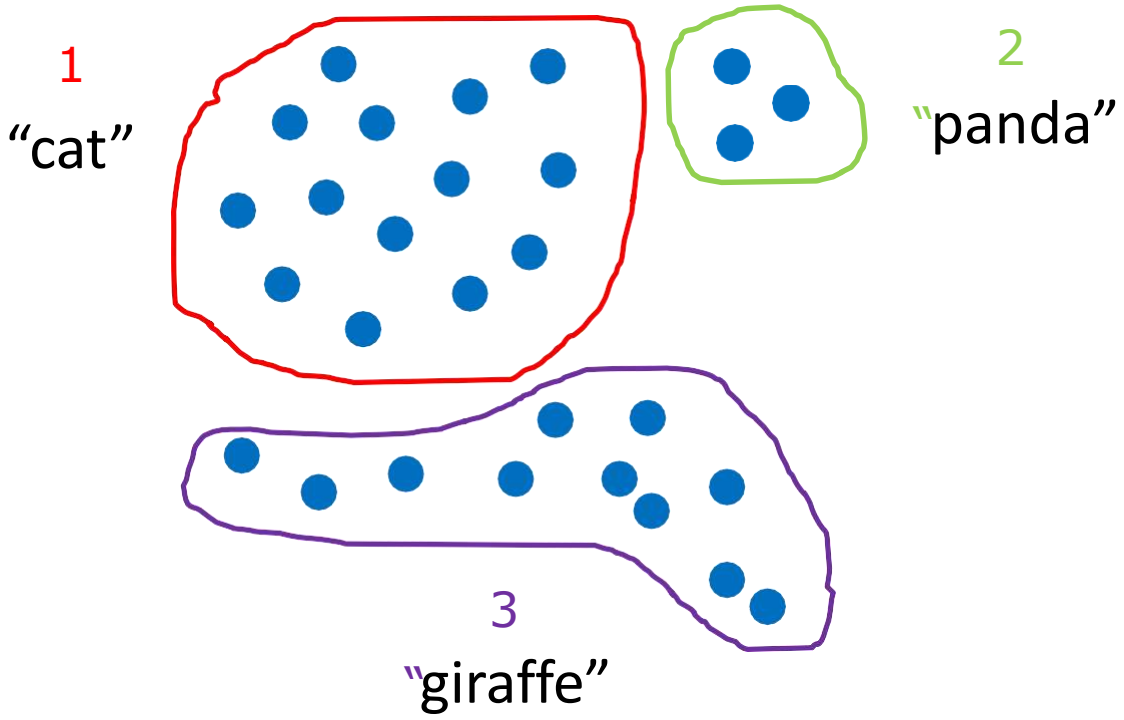
  - Represent a large continuous vector with the cluster number
- Prediction
  - Data points in the same cluster may have the same labels
  - Ask a human to label the clusters

Slide credit: J. Hays, D. Hoiem

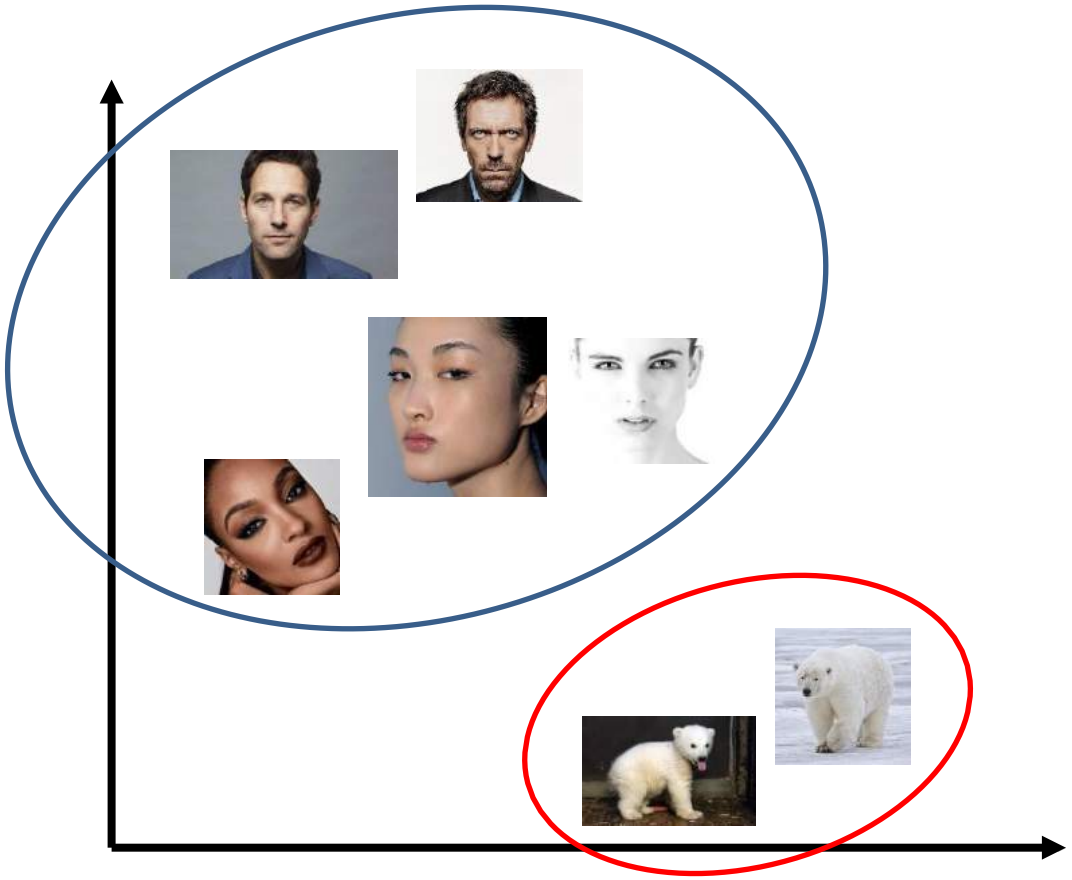
Why do we cluster?

- Two uses of clustering in one application
- Cluster, then ask human to label groups
- Compute a histogram to summarize the data





Unsupervised discovery

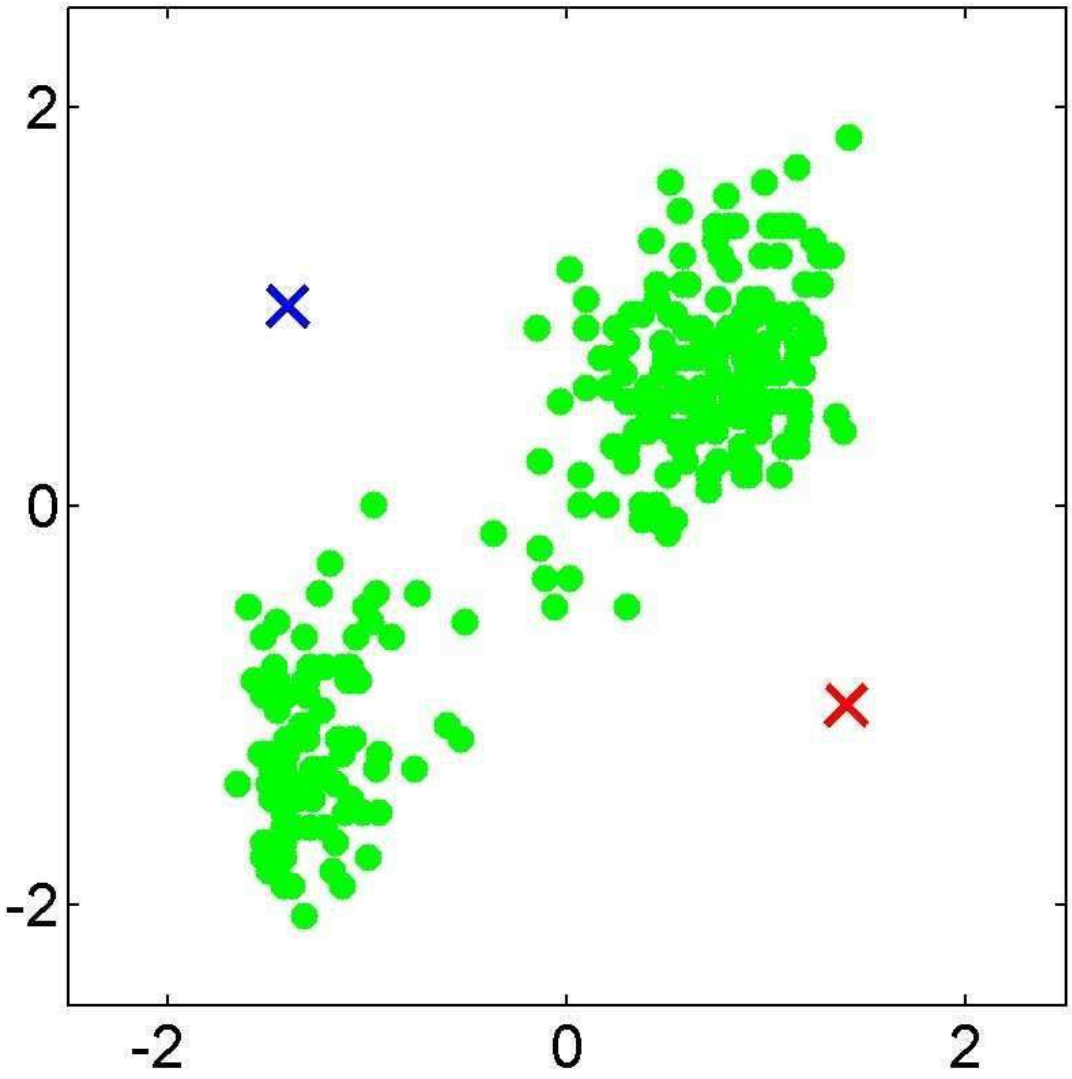


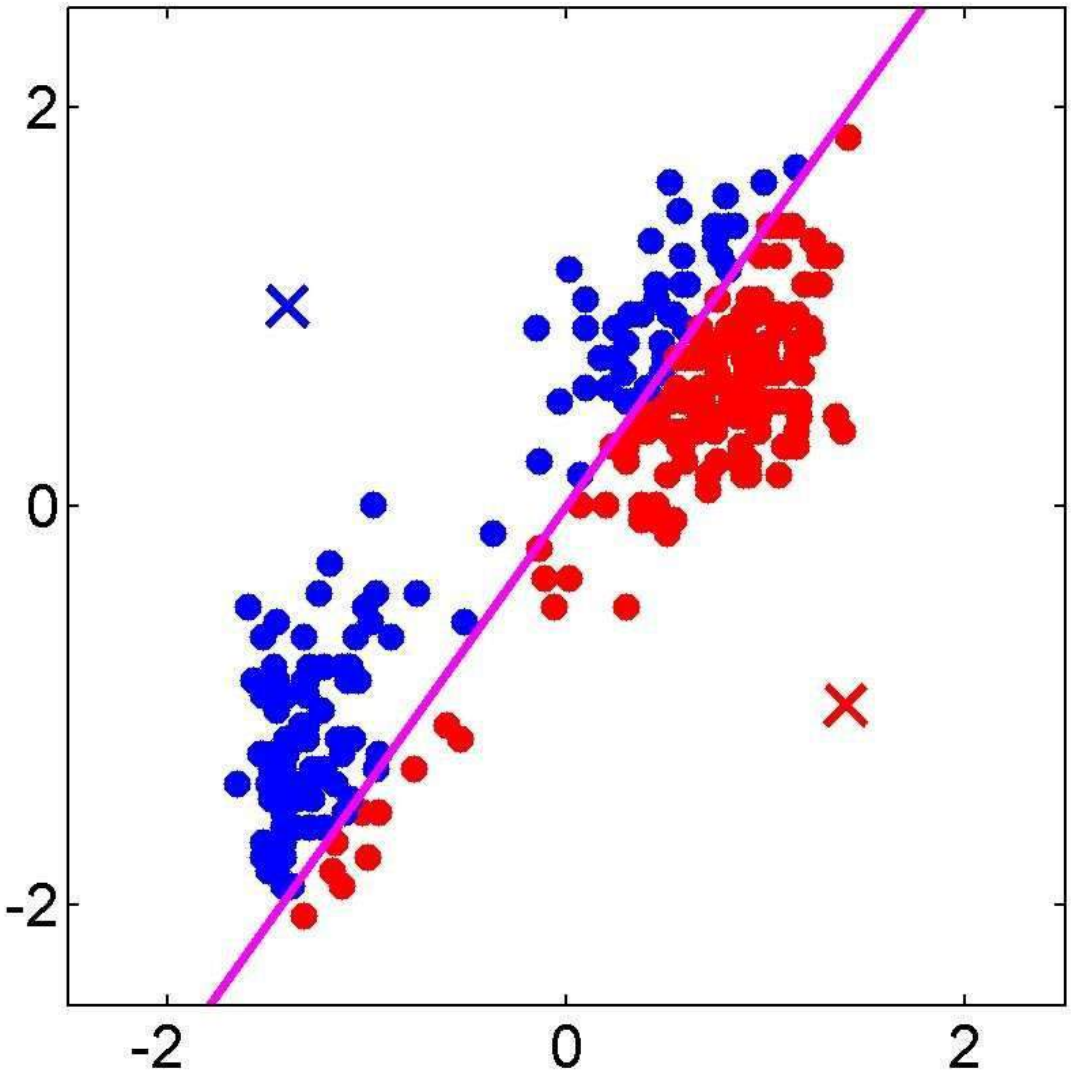
Clustering algorithms

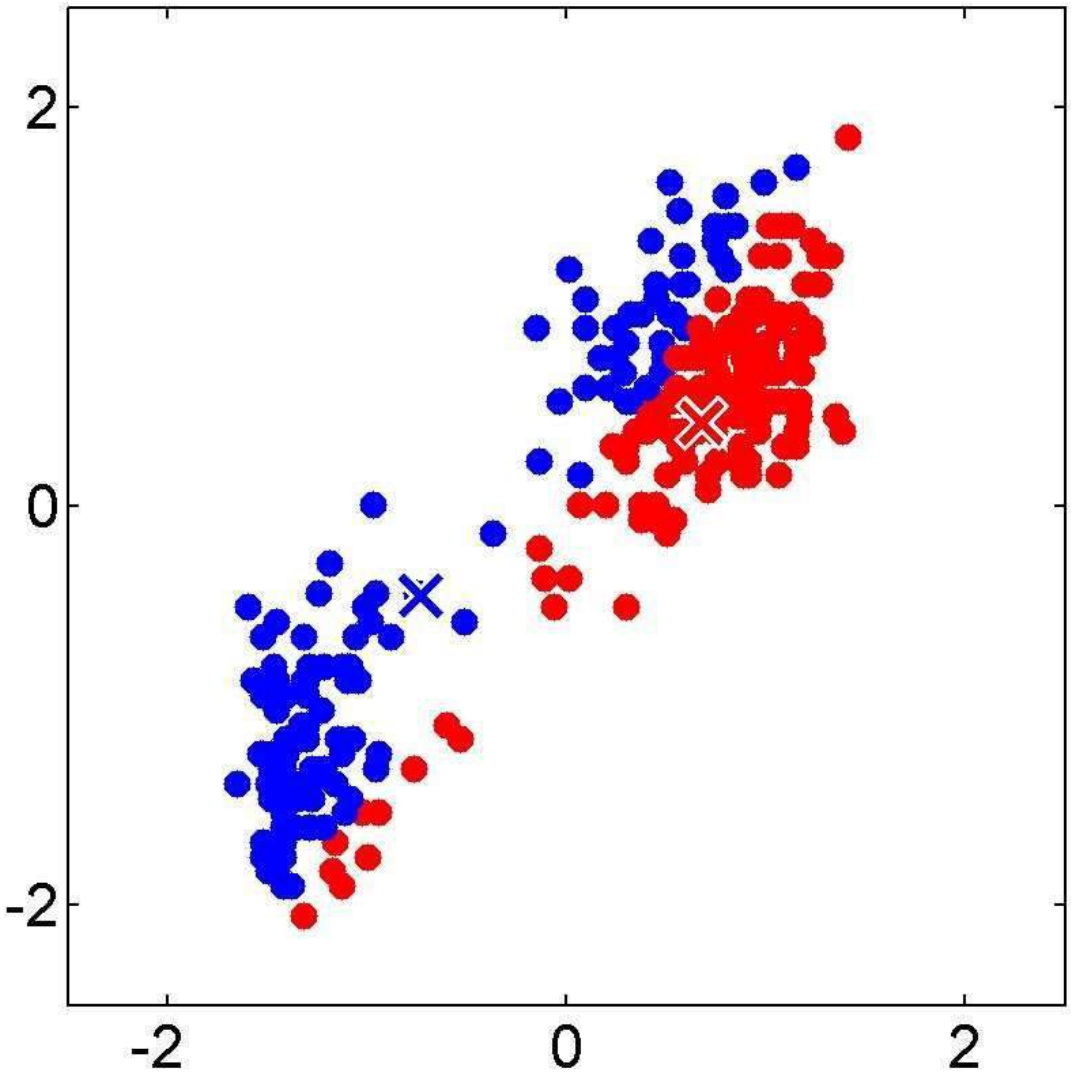
- In depth
  - K-means (iterate between finding centers and assigning points)
  - Gaussian Mixture Models (GMMs)

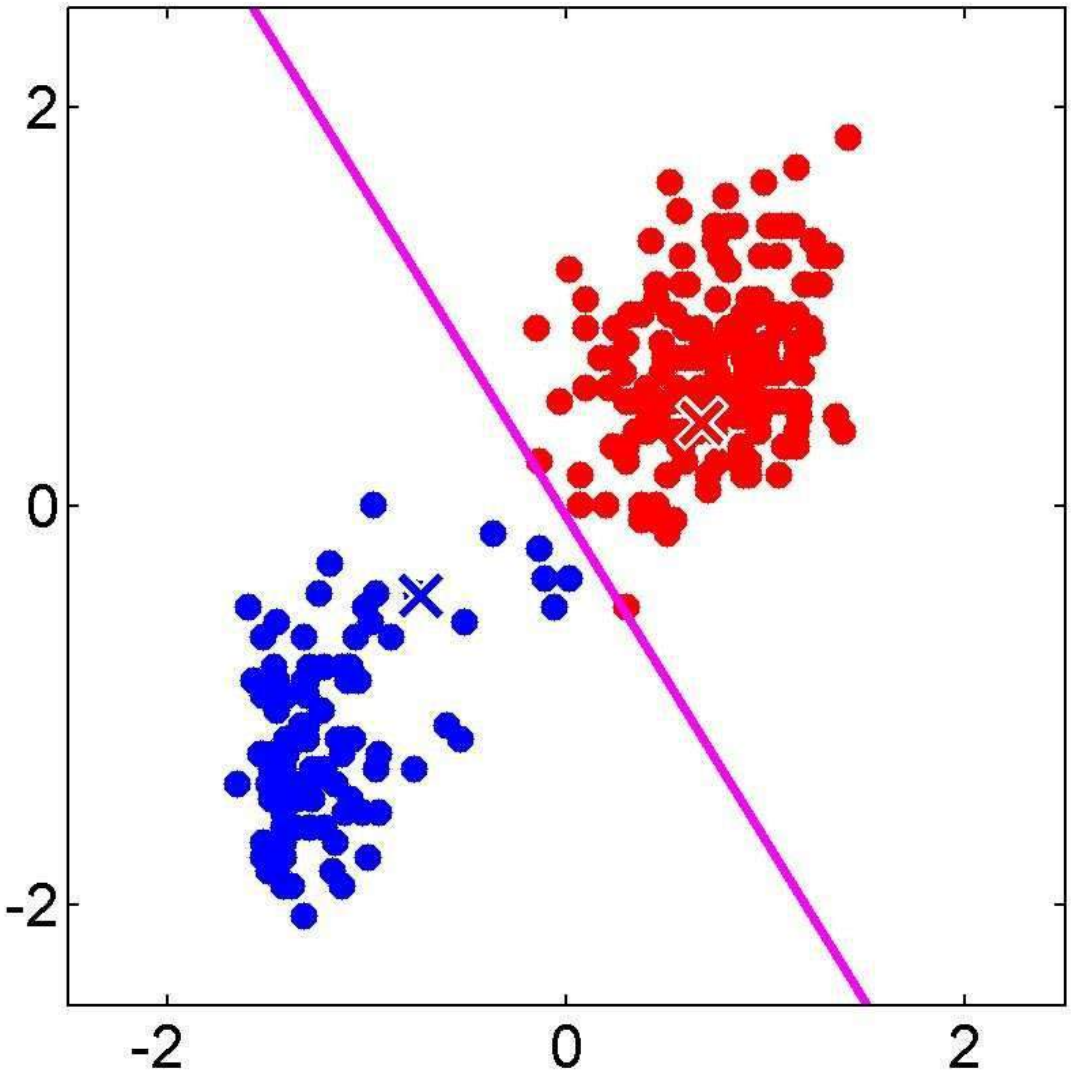
K-means Algorithm

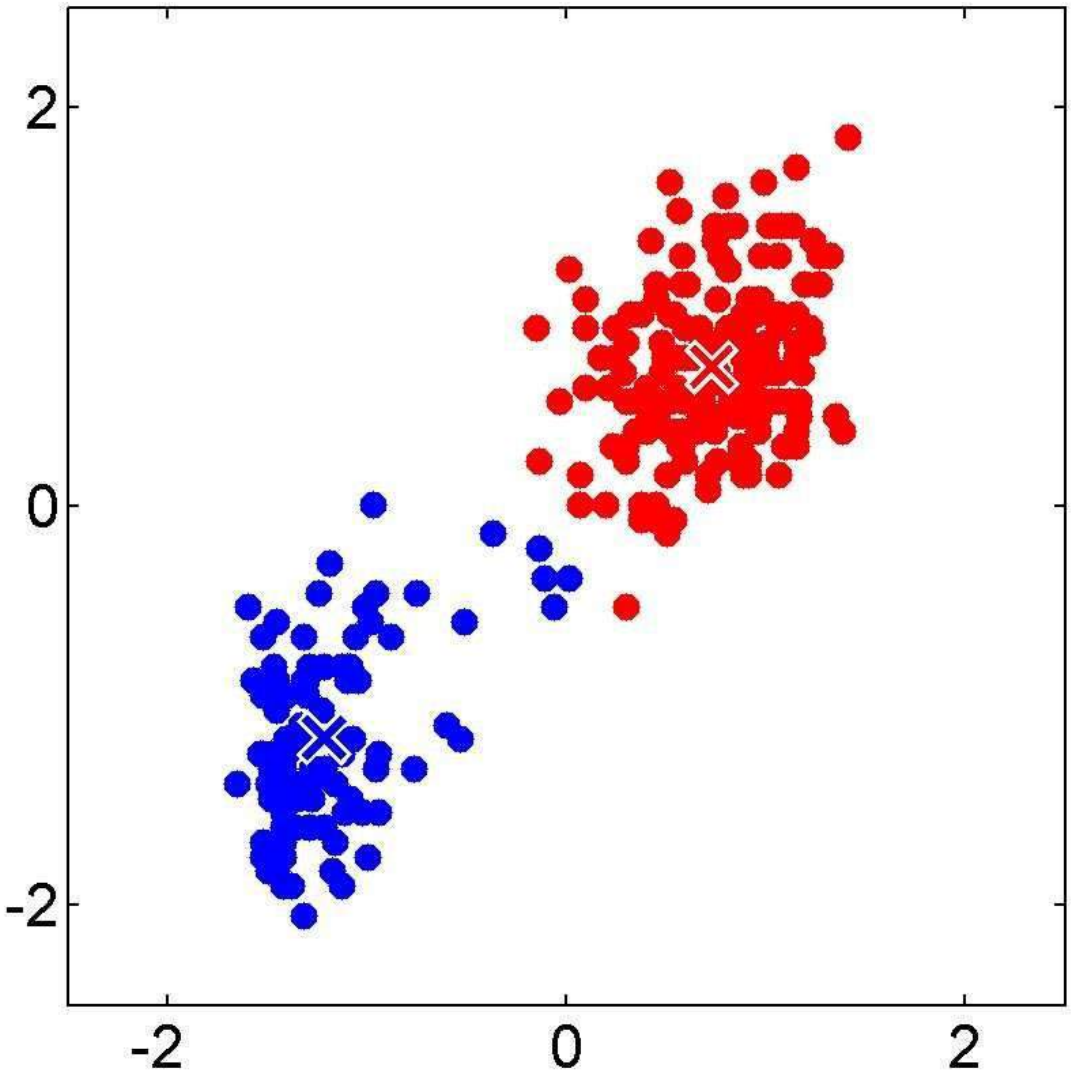
- Goal: represent a data set in terms of K clusters each of which is summarized by a prototype μ_k
- Initialize prototypes, then iterate between two phases:
  - E-step: assign each data point to nearest prototype
  - M-step: update prototypes to be the cluster means
- Simplest version is based on Euclidean distance
  - re-scale Old Faithful data

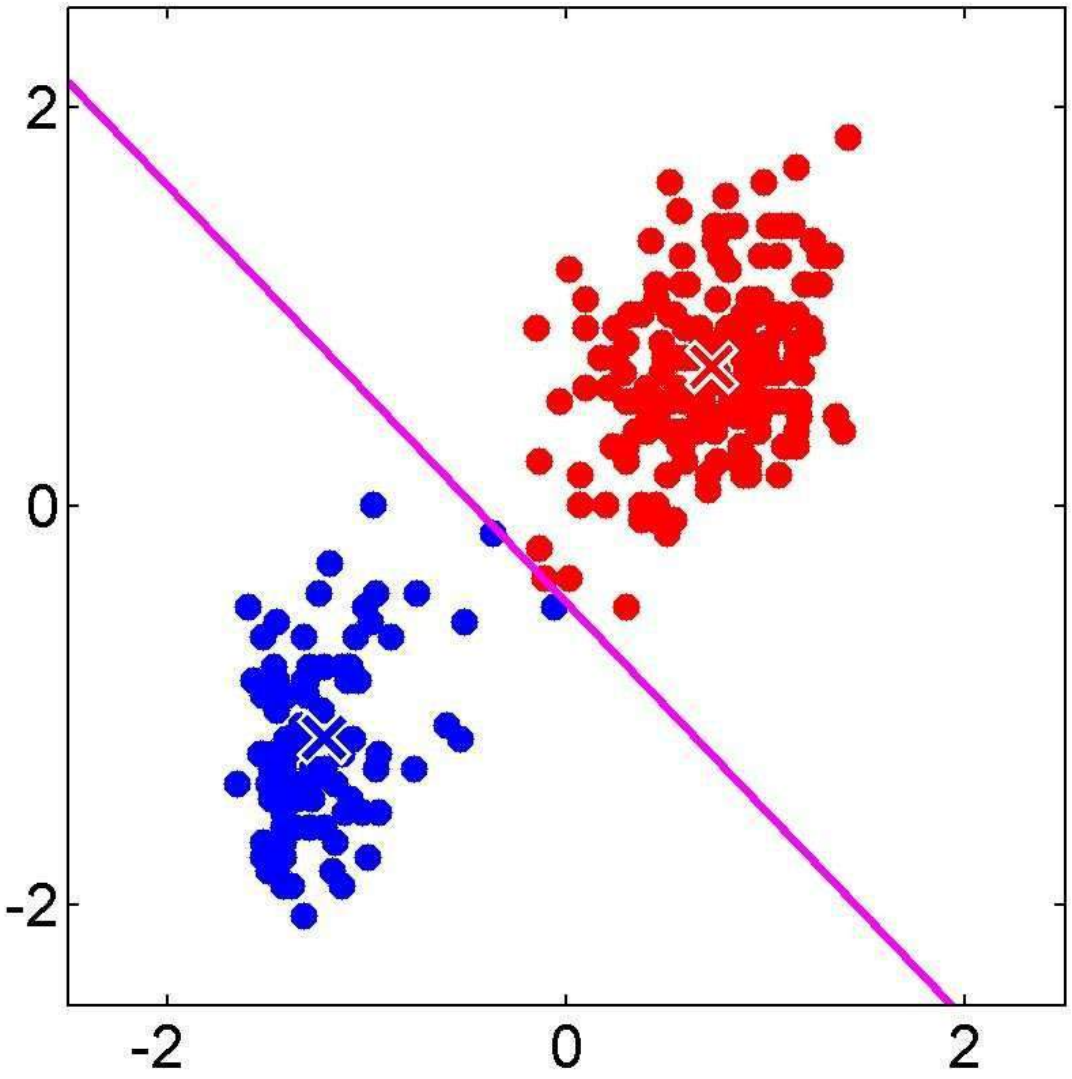

2
0
-2
-2
0
2

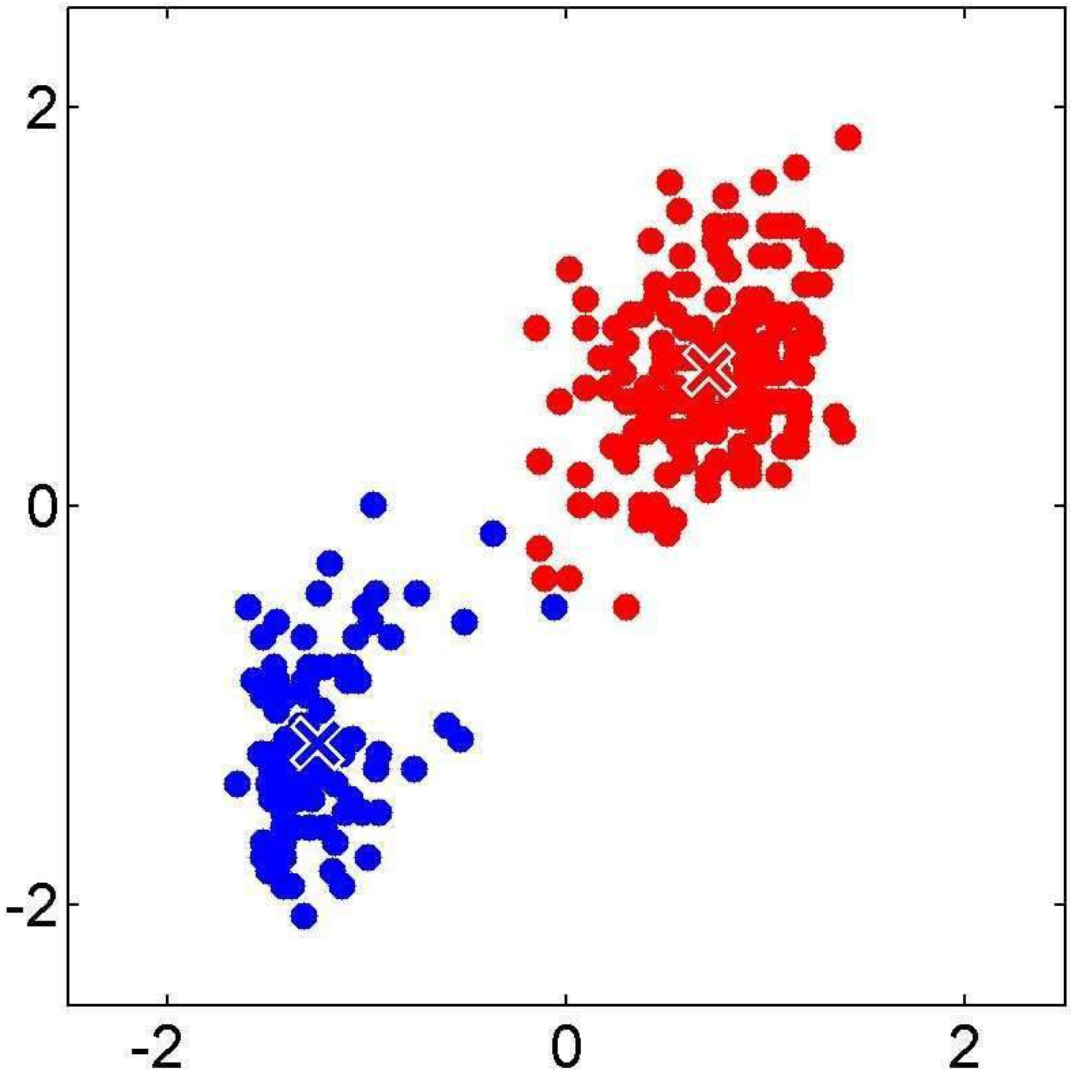

2
0
-2
-2
0
2

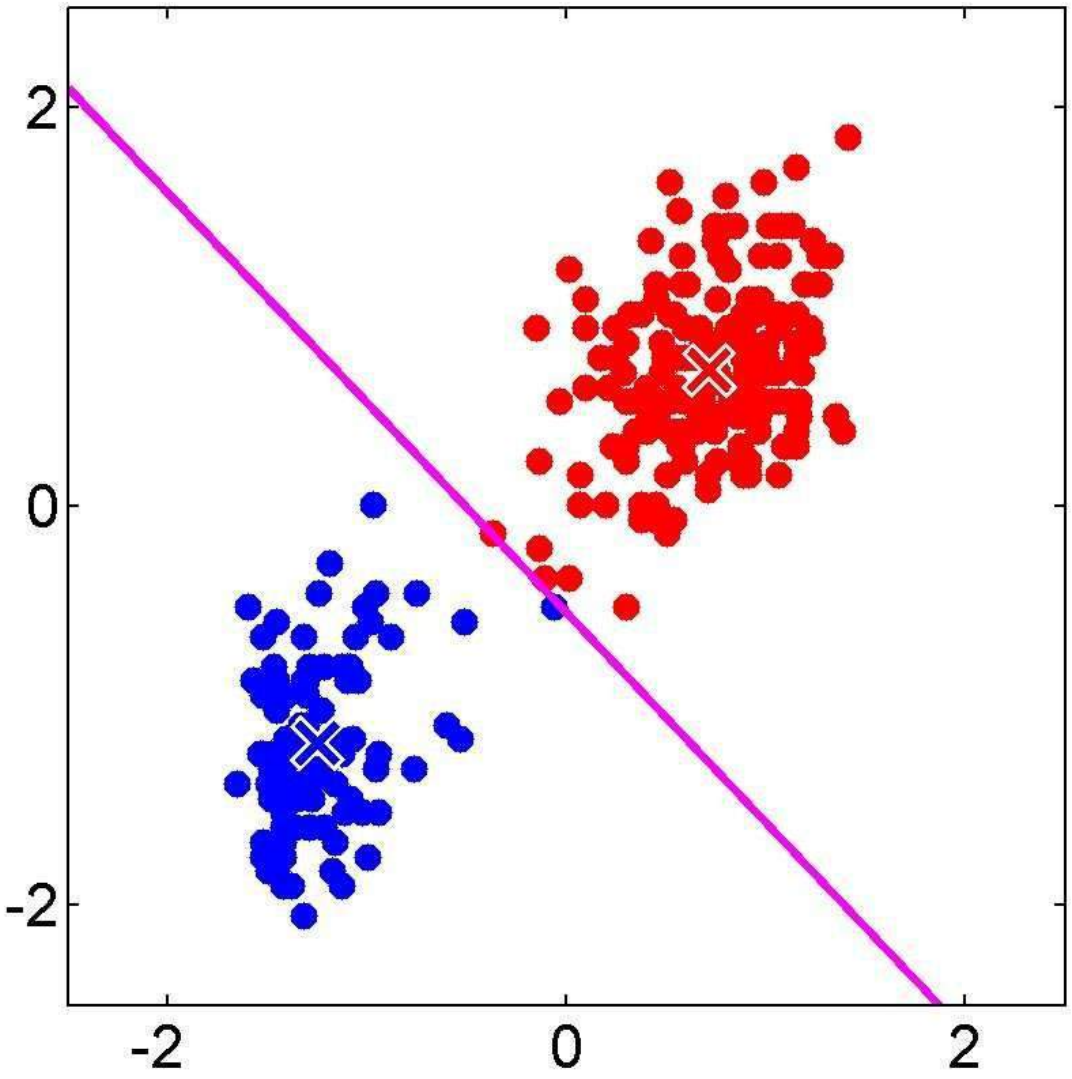

2
0
-2
-2
0
2

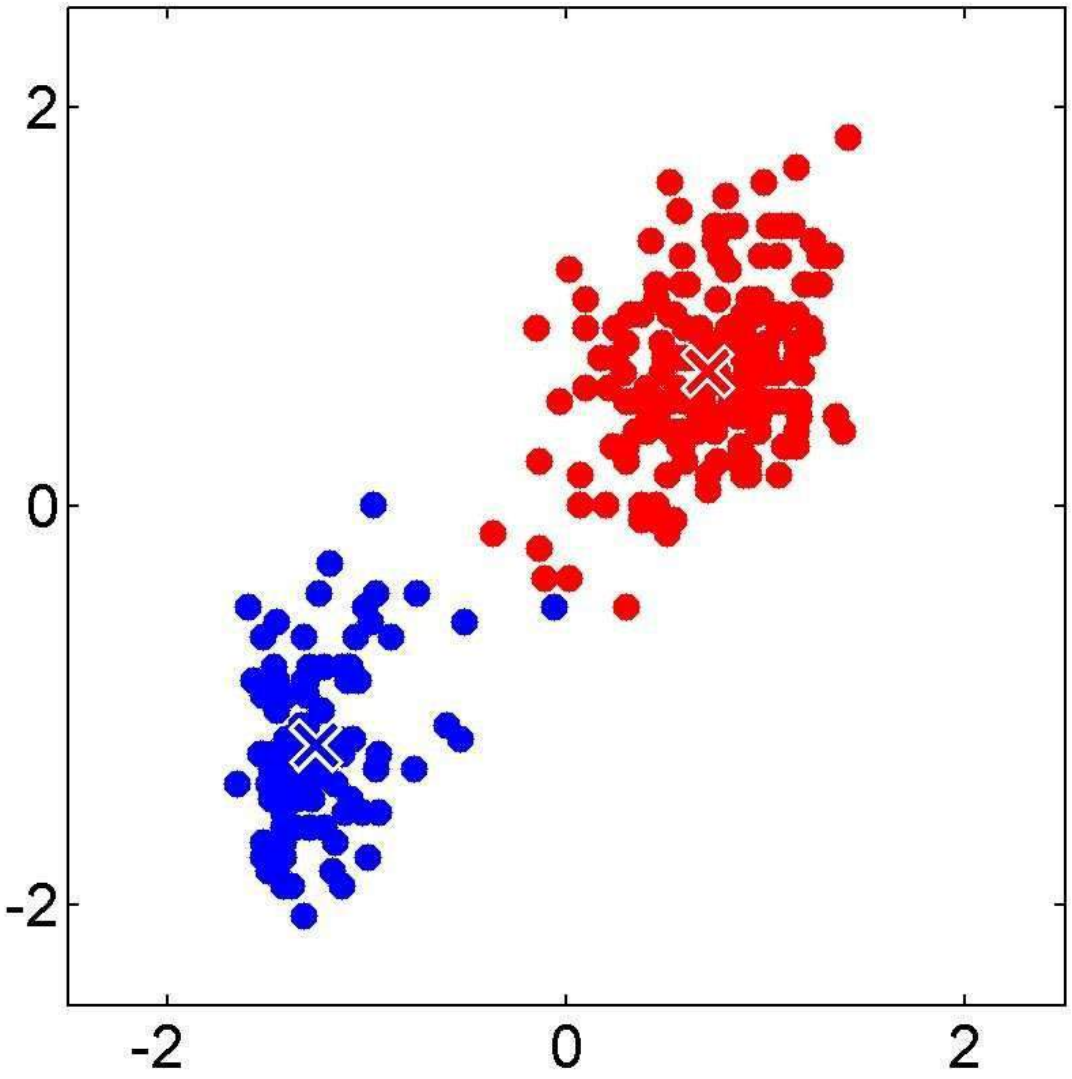

2
0
-2
-2
0
2


2
0
-2
-2
0
2


2
0
-2
-2
0
2


2
0
-2
-2
0
2


2
0
-2
-2
0
2


2
0
-2
-2
0
2

1 of K coding mechanism

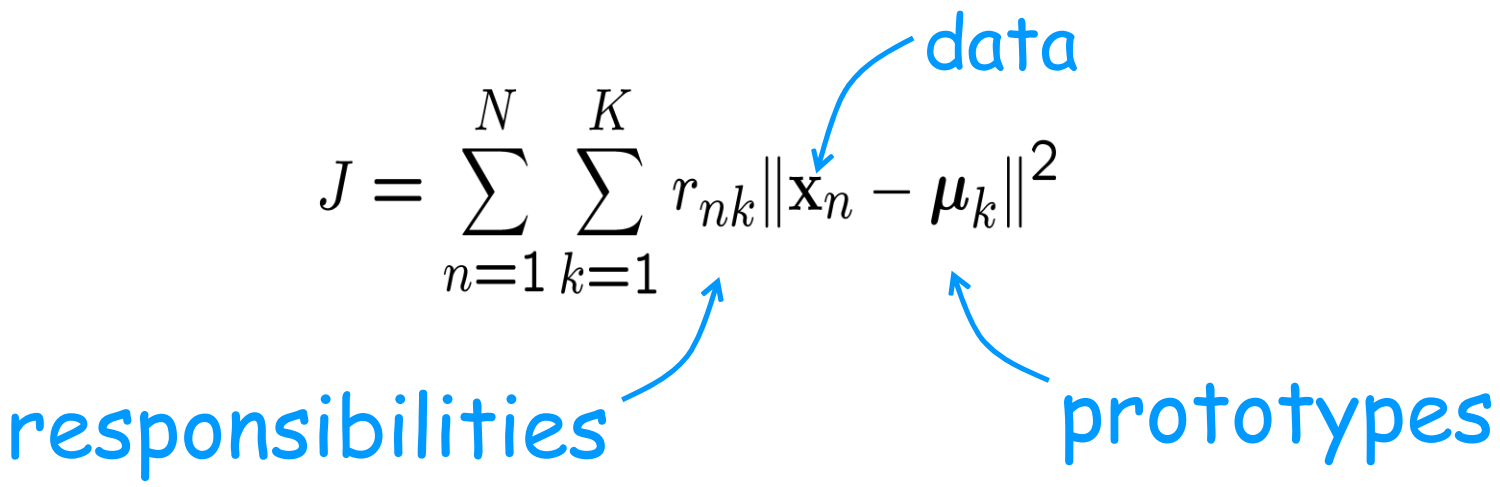
- *For each data point x_n, we introduce a set of binary indicator variables* $r_{nk} \in \{0, 1\}$
  such that $\sum_k r_{nk} = 1$

  where k =1,...,K describing which of the K clusters the data point x_n is assigned to, so that if data point x_n is assigned to cluster k then r_{nk} =1, and r_{nj} =0 for j not equal to k.

- Example: 5 data points and 3 clusters

$$(r_{nk}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

K-means Cost Function



Minimizing the Cost Function

- E-step: minimize J w.r.t. r_{nk}

$$J = \sum_n \sum_k r_{nk} \|\underline{x}_n - \underline{\mu}_k\|^2$$

- M-step: minimize J w.r.t $\boldsymbol{\mu}_k$

$$0 = 2 \sum_{n=1}^{N} r_{nj} (\underline{x}_n - \underline{\mu}_j)$$

$$\underline{\mu}_j = \frac{\sum_n r_{nj} \underline{x}_n}{\sum_n r_{nj}}$$

- Convergence guaranteed since there is a finite number of possible settings for the responsibilities

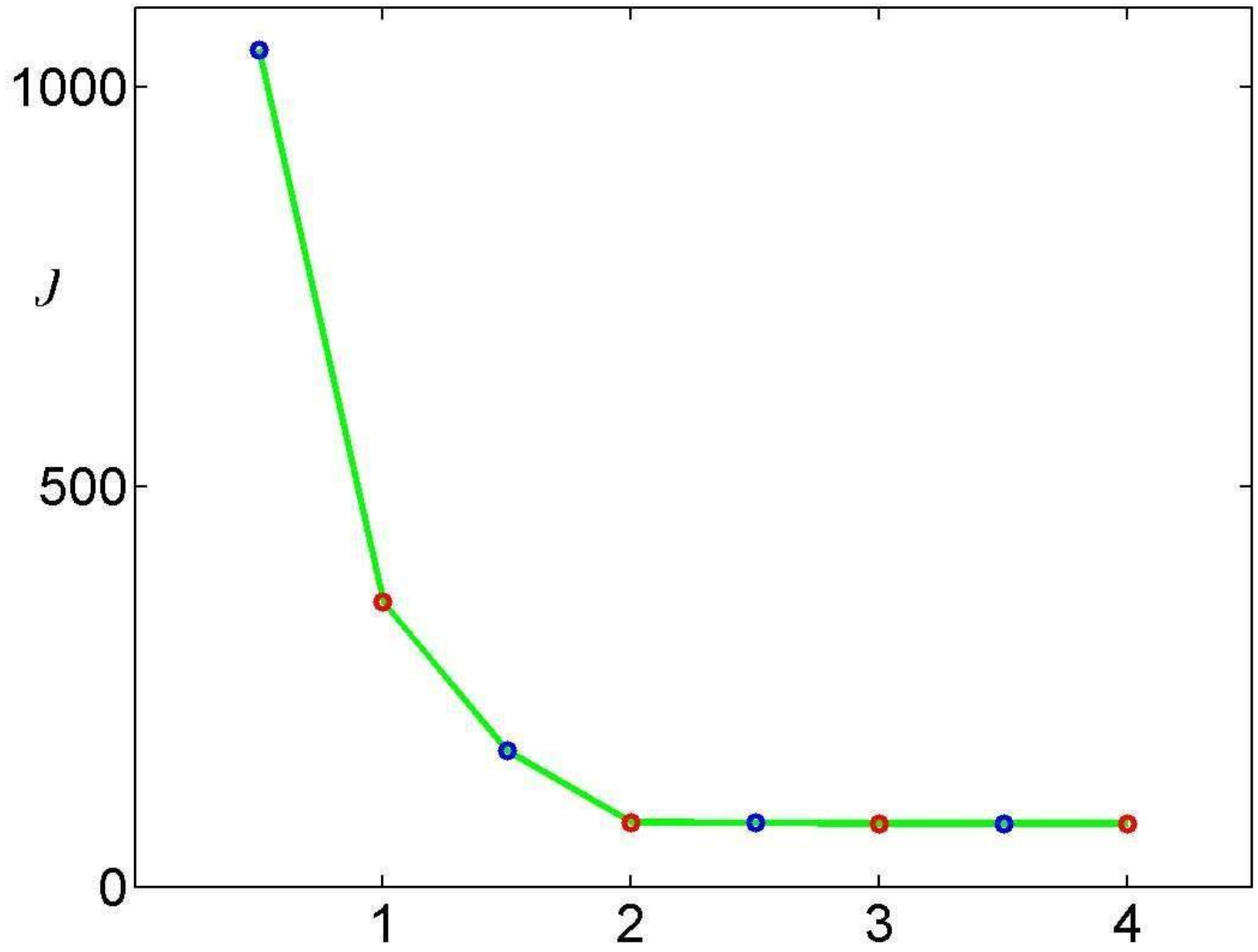
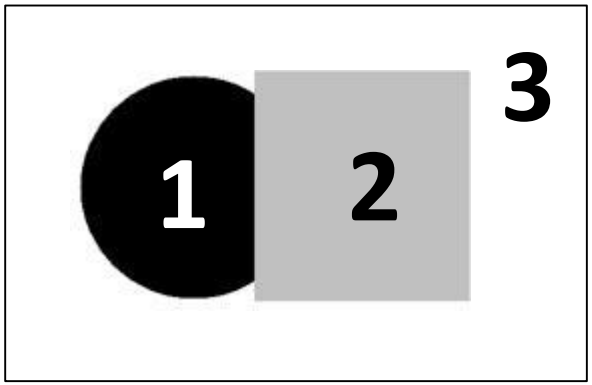
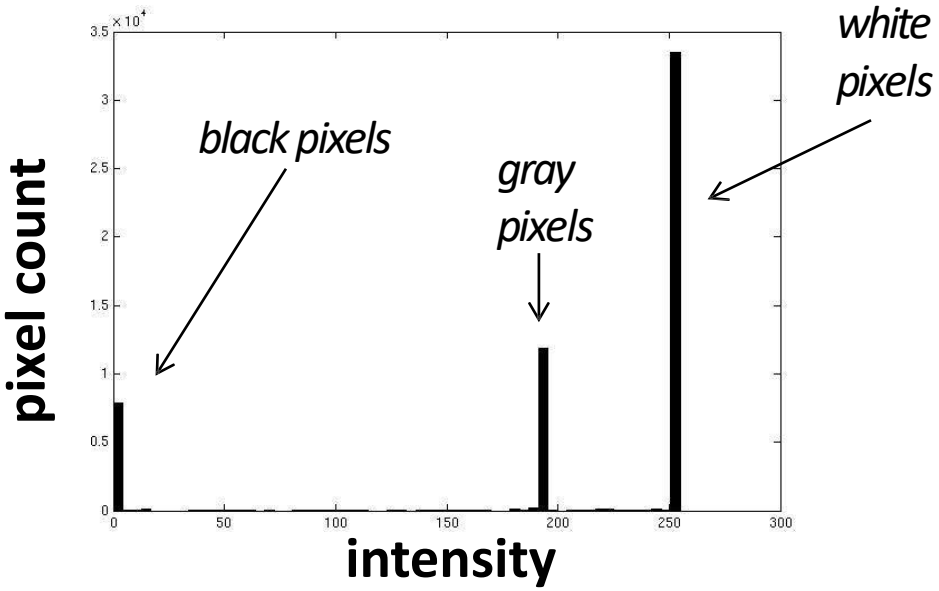

1000
J
500
0
1
2
3
4

Image segmentation: toy example



input image

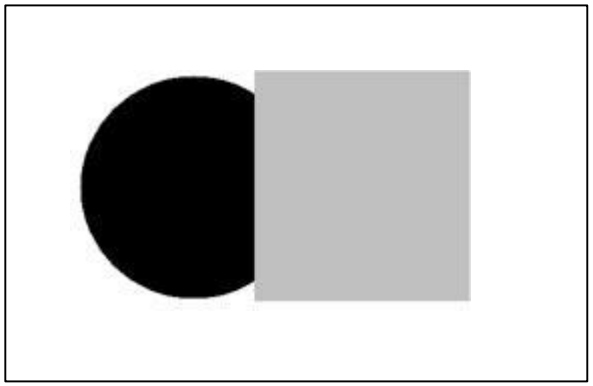


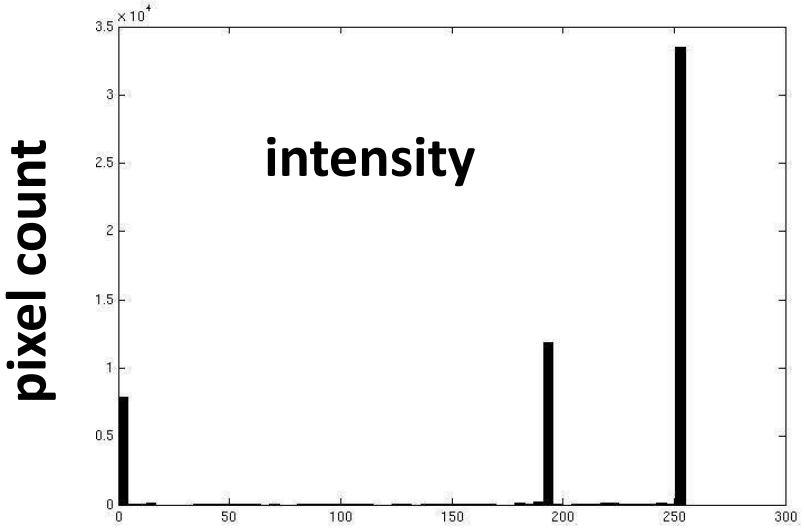
- These intensities define the three groups.
- We could label every pixel in the image according to which of these primary intensities it is.
  - i.e., *segment* the image based on the intensity feature.
- What if the image isn't quite so simple?

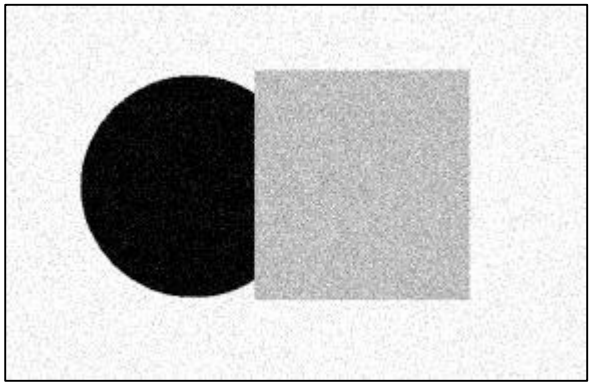
Source: K. Grauman

- Now how to determine the three main intensities that define our groups?
- We need to ***cluster.***

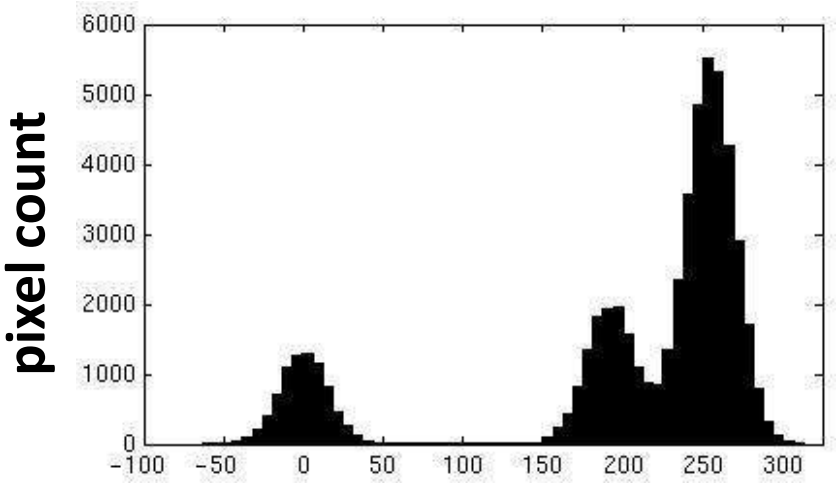
input image



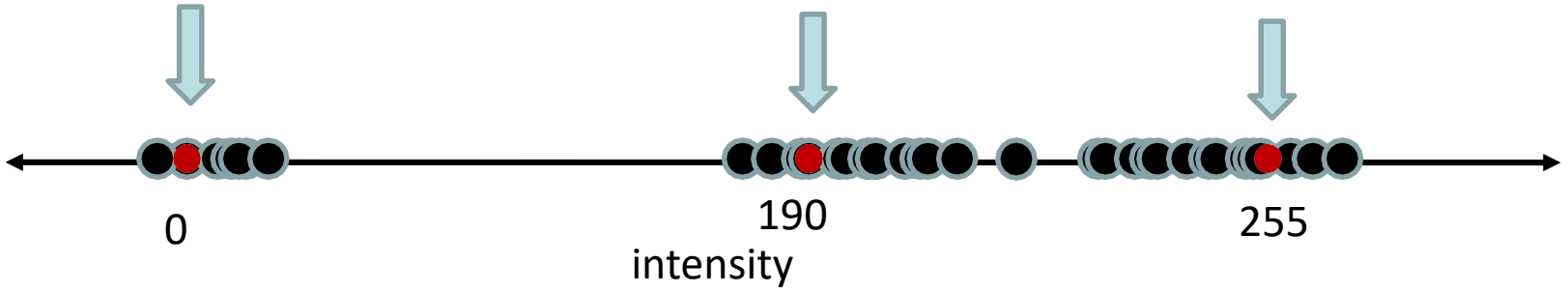


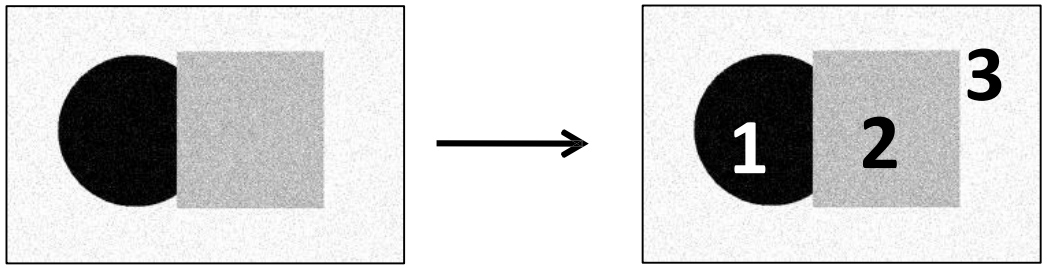


input image



Source: K. Grauman





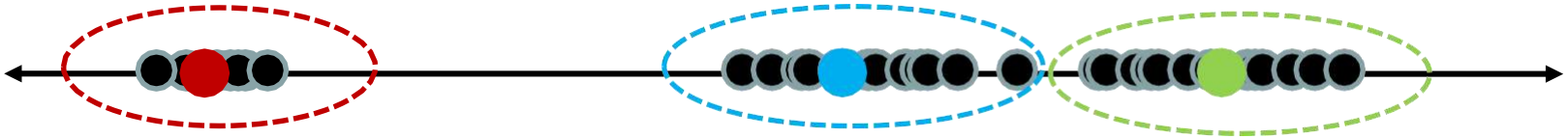
- Goal: choose three "centers" as the representative intensities, and label every pixel according to which of these centers it is nearest to.
- Best cluster centers are those that minimize SSD between all points and their nearest cluster center c_i:

$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} \|p - c_i\|^2$$

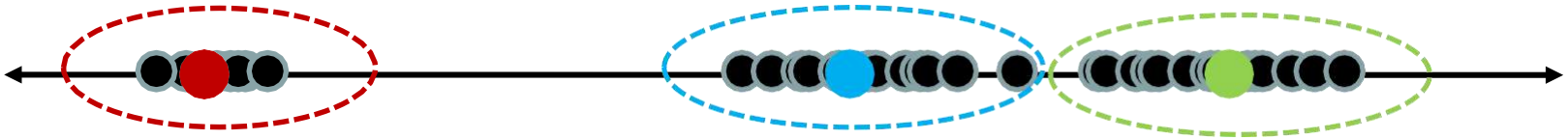
Source: K. Grauman

Clustering

- With this objective, it is a "chicken and egg" problem:
  - If we knew the **cluster centers**, we could allocate points to groups by assigning each to its closest center.



  - If we knew the **group memberships**, we could get the centers by computing the mean per group.

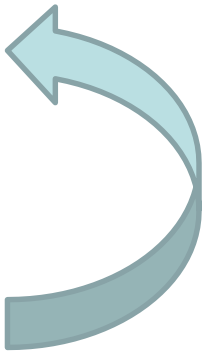


Source: K. Grauman

K-means clustering

- Basic idea: randomly initialize the *k* cluster centers, and iterate between the two steps we just saw.

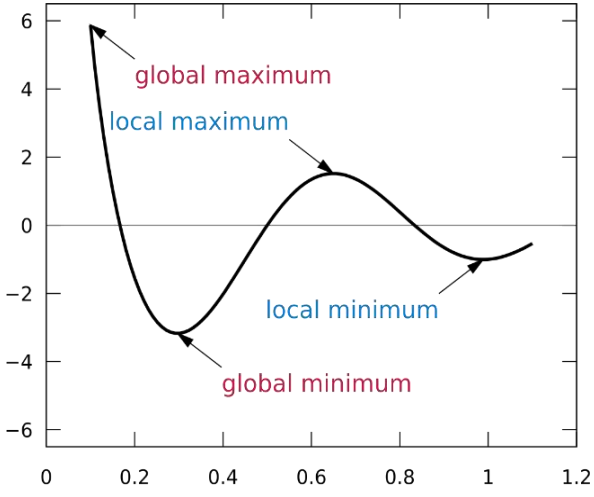
1. *Randomly* initialize the cluster centers, $c_1, \ldots, c_K$
2. *Given cluster centers*, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i
3. *Given points in each cluster*, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2



Properties

- Will always converge to some solution
- Can be a "local minimum" of objective:

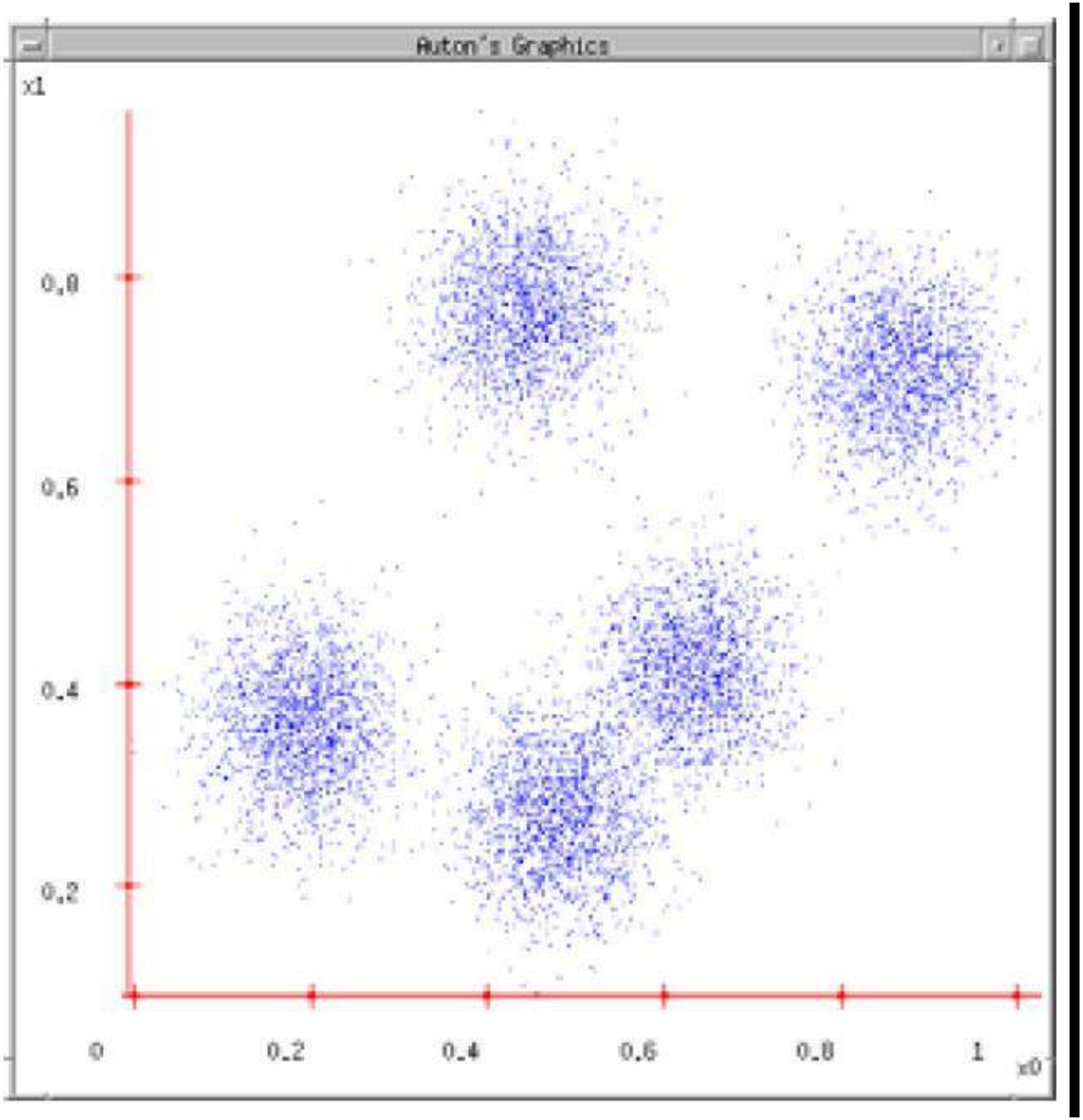
$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} \|p - c_i\|^2$$



Slide: Steve Seitz, image: Wikipedia

K-means

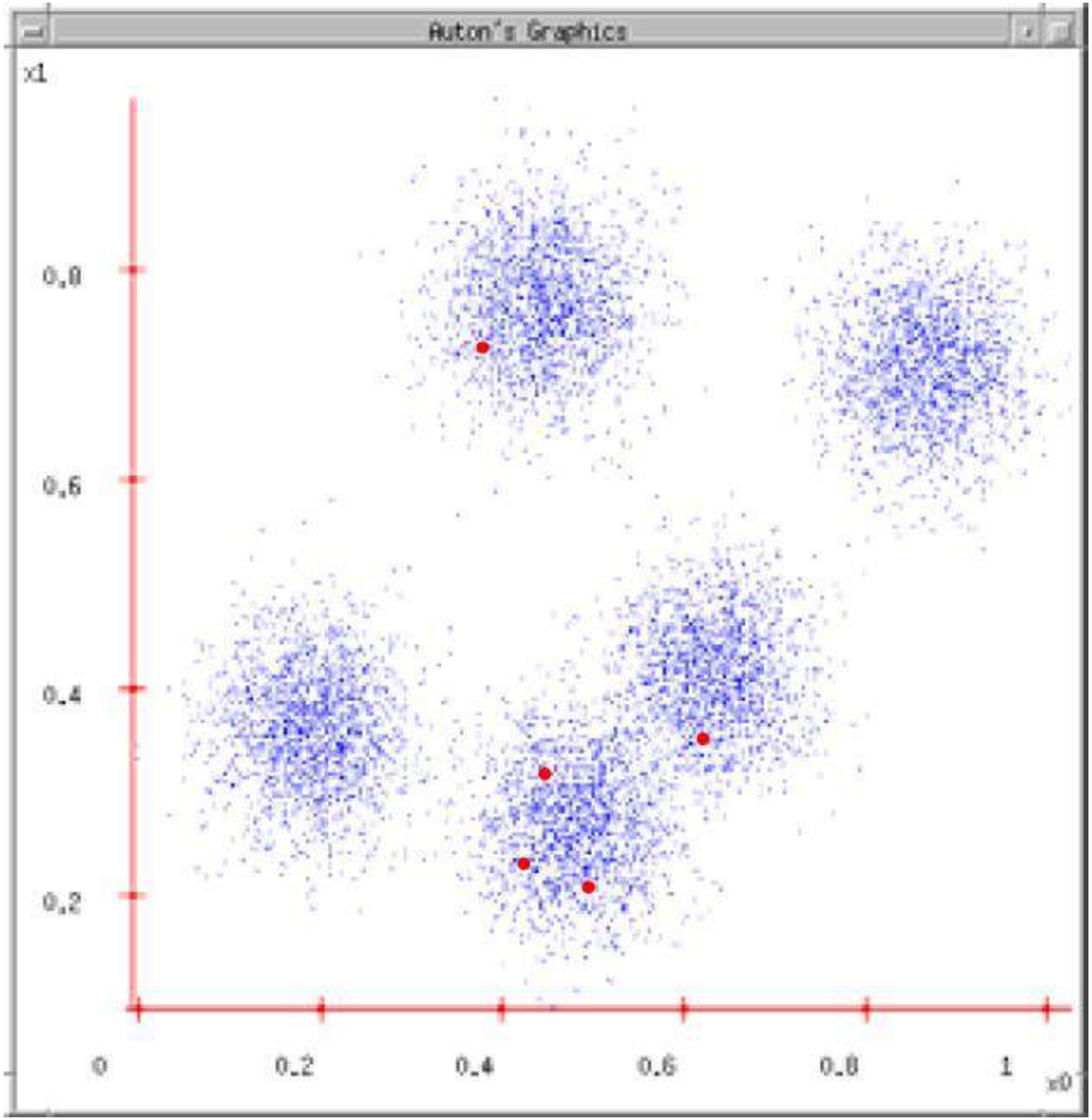
1. Ask user how many clusters they'd like. *(e.g. k=5)*



Source: A. Moore

K-means

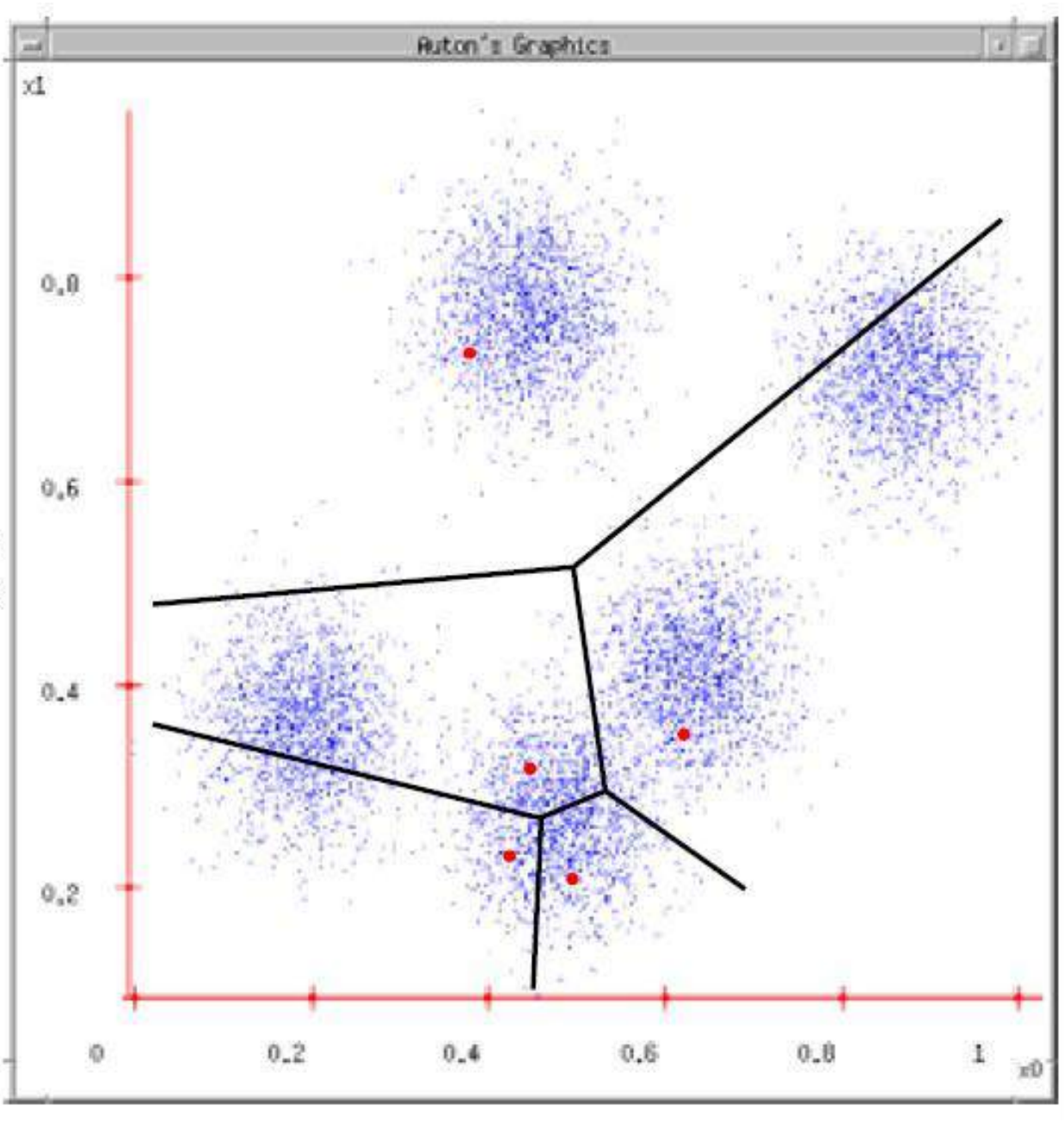
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



Source: A. Moore

K-means

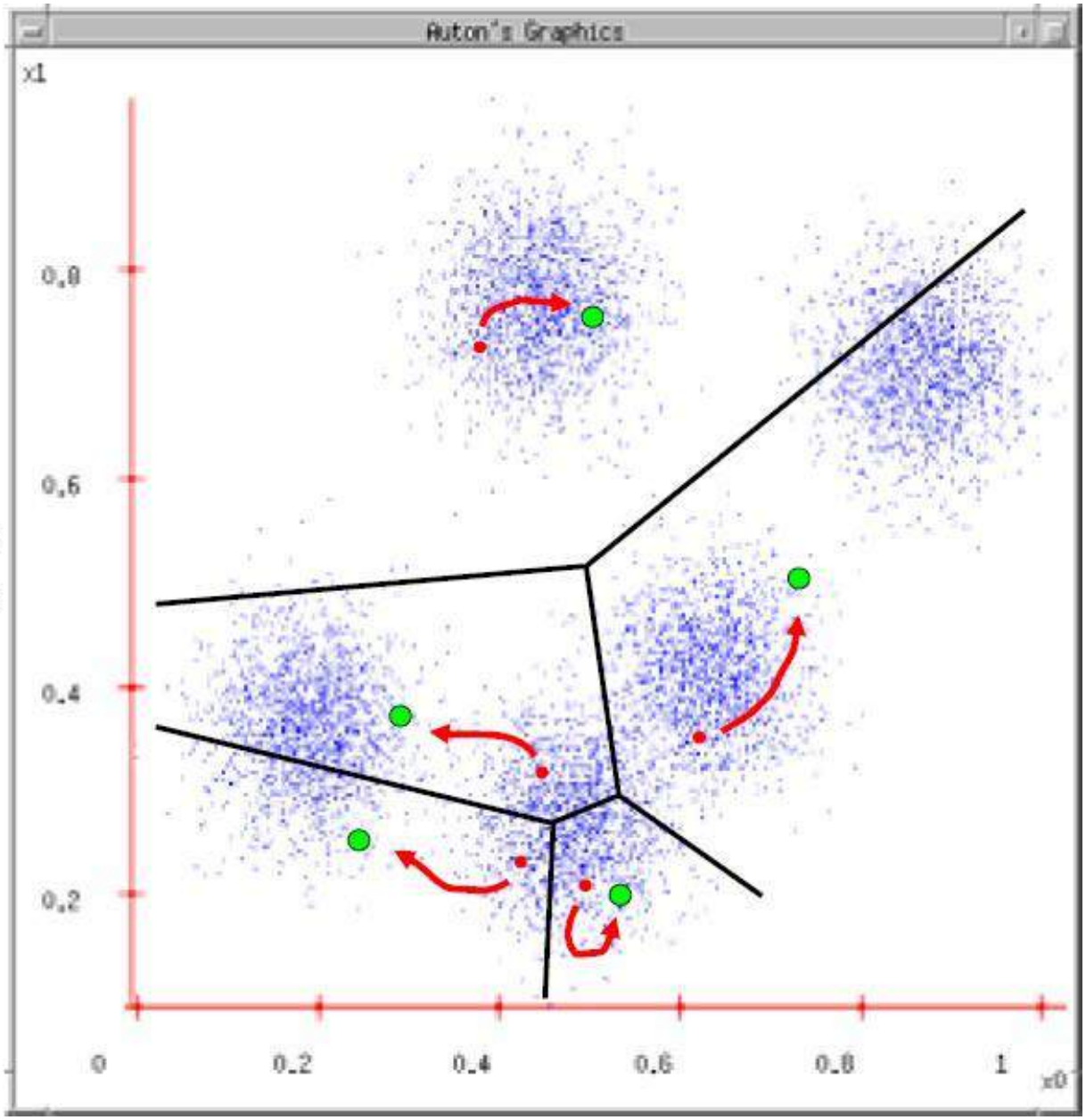
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



Source: A. Moore

K-means

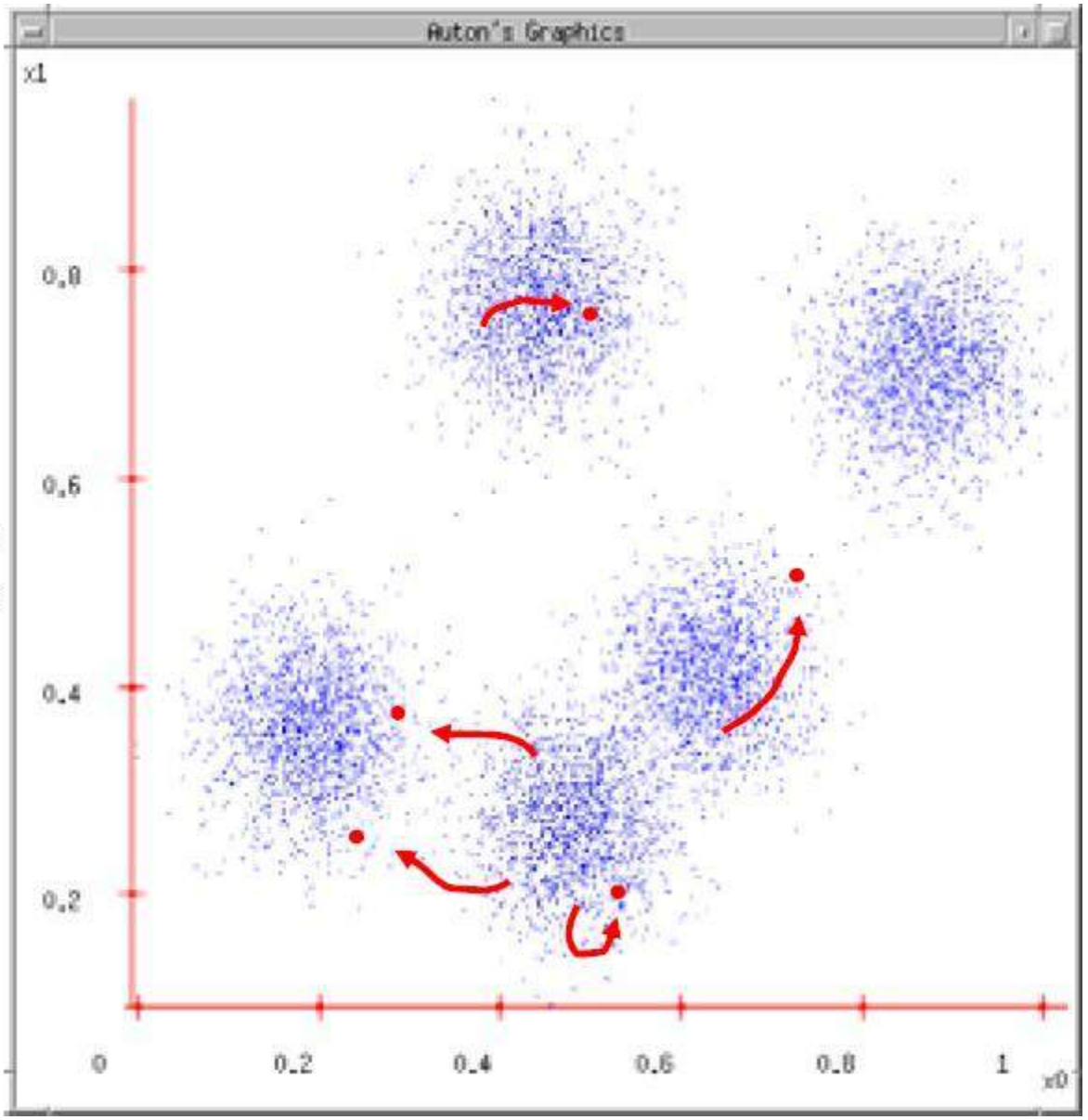
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



Source: A. Moore

K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



Source: A. Moore

K-means clustering

- Visualization

https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

- Java demo

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

- Matlab demo

http://www.cs.pitt.edu/~kovashka/cs1699_fa15/kmeans_demo.m

Time Complexity

- Let n = number of instances, d = dimensionality of the features, k = number of clusters
- Assume computing distance between two instances is O(d)
- Reassigning clusters:
  - O(kn) distance computations, or O(knd)
- Computing centroids:
  - Each instance vector gets added once to a centroid: O(nd)
- Assume these two steps are each done once for a fixed number of iterations I: O($Iknd$)
  - Linear in all relevant factors

Adapted from Ray Mooney

Another way of writing objective

- **K-means:** Let r_{nk}= 1 if instance n belongs to cluster k, 0 otherwise

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}}$$

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- k-means the centre of a cluster is not necessarily one of the input data points (it is the average between the points in the cluster).

- **K-medoids** (more general distances):

$$\widetilde{J} = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\mathcal{V}(\mathbf{x}_n, \boldsymbol{\mu}_k)$$

- k-medoids chooses data points as centers (medoids or exemplars) and can be used with arbitrary distances
- k-medoids more robust to noise and outliers as compared to *k*-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

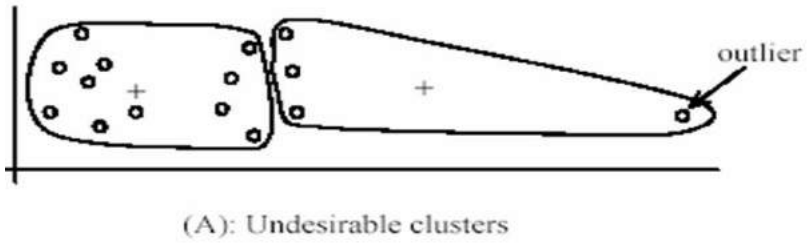
K-means: pros and cons

Pros

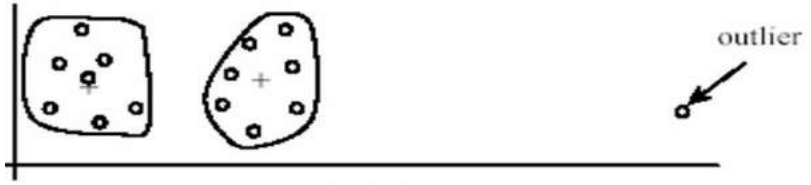
- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

Cons/issues

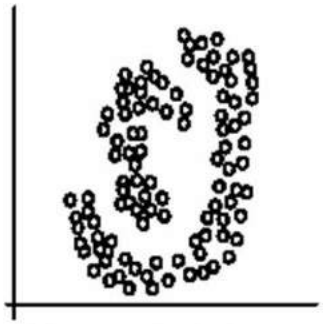
- Setting k?
- Sensitive to initial centers
  - Use heuristics or output of another method
- Sensitive to outliers
- Detects spherical clusters



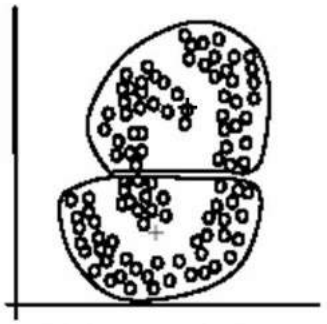
(A): Undesirable clusters



(B): Ideal clusters



(A): Two natural clusters



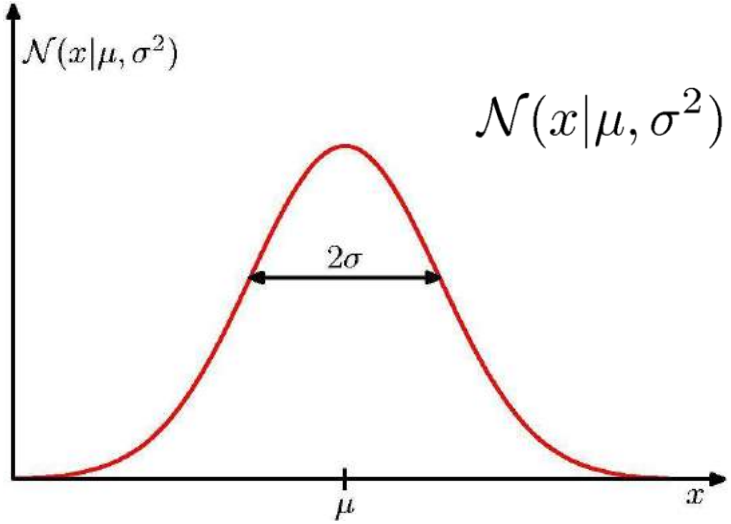
(B): *k*-means clusters

Adapted from K. Grauman

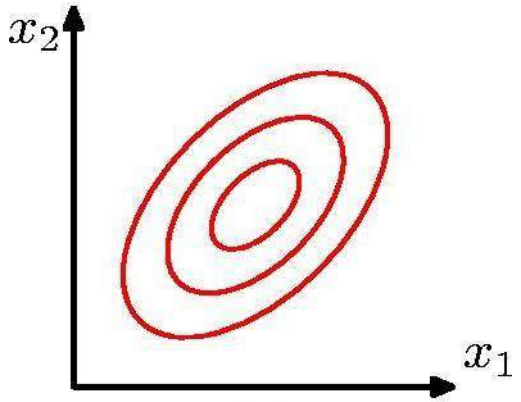
Probabilistic Clustering

- Represent the probability distribution of the data as a *mixture model*
  - captures uncertainty in cluster assignments
  - gives model for data distribution
  - *Bayesian* mixture model allows us to determine K
- Consider mixtures of *Gaussians*

Review: Gaussian Distribution



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



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

Mixtures of Gaussians

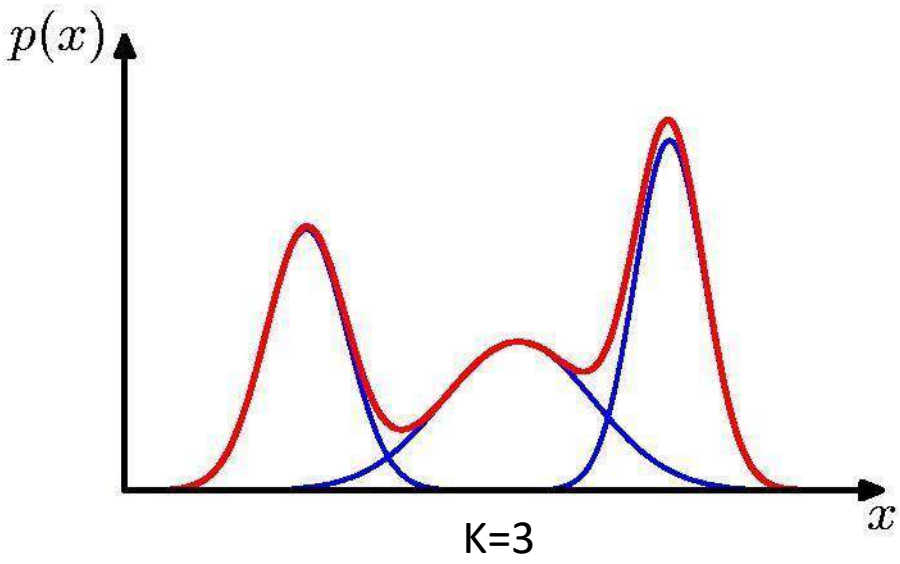
- Combine simple models into a complex model:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Component

Mixing coefficient

$$\forall k : \pi_k \geqslant 0 \qquad \sum_{k=1}^{K} \pi_k = 1$$



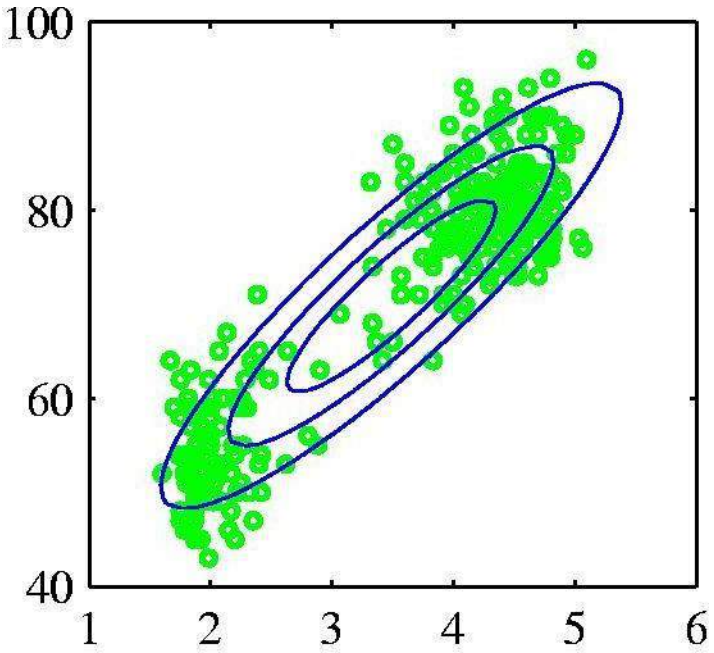
K=3

- Find parameters through EM (Expectation Maximization) algorithm

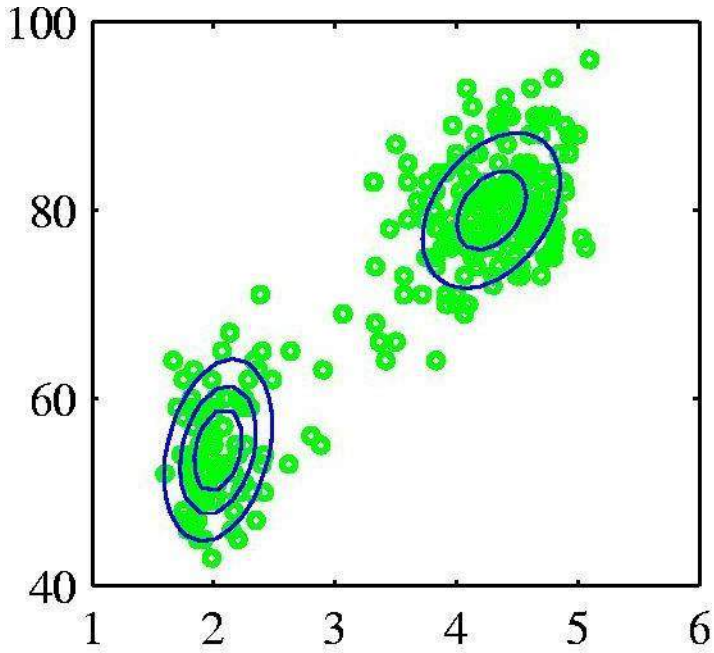
Adapted from Chris Bishop

Probabilistic version: Mixtures of Gaussians

- Old Faithful data set



Single Gaussian

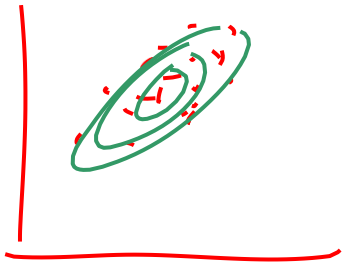


Mixture of two Gaussians

Gaussian Mixture Model

- Data set

$$D = \{\mathbf{x}_n\} \quad n = 1, \ldots, N$$



- Consider first a single Gaussian
- Assume observed data points generated independently

$$p(D|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

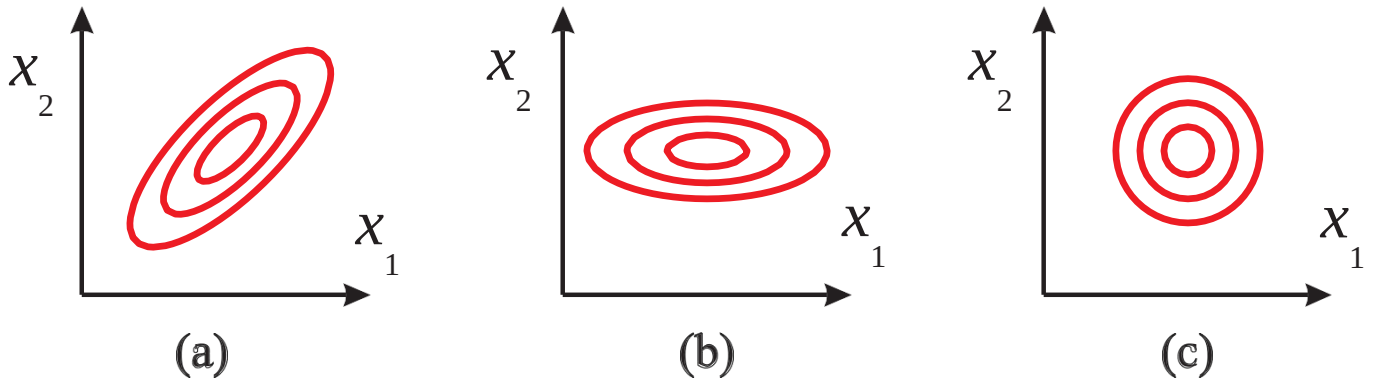
- Viewed as a function of the parameters, this is known as the *likelihood function*

The Gaussian Distribution

- Multivariate Gaussian

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

mean covariance



Gaussian Mixture Model

- K-dimensional binary random variable z having a 1-of-K representation in which a particular element z_k is equal to 1 and all other elements are equal to 0.
- The values of z_k therefore satisfy $z_k \in \{0,1\}$
- K possible states for the vector z according to which element is nonzero.
- Joint distribution p(x,z) in terms of a marginal distribution p(z) and a conditional distribution p(x|z),
- Marginal distribution over z is specified in terms of the mixing coefficients π_k, such that $p(z_k = 1) = \pi_k$

Sampling from the Gaussian

- To generate a data point:
  - first pick one of the components with probability π_k
  - then draw a sample $\mathbf{x}_n$ from that component
- Repeat these two steps for each new data point

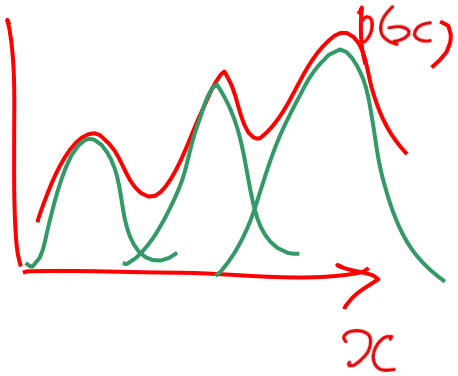
Fitting the Gaussian Mixture

- We wish to invert this process – given the data set, find the corresponding parameters:
  - mixing coefficients
  - means
  - covariances
- If we knew which component generated each data point, the maximum likelihood solution would involve fitting each component to the corresponding cluster
- Problem: the data set is unlabelled
- We shall refer to the labels as *latent* (= hidden) variables

Gaussian Mixture Model

- Linear super-position of Gaussians

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$



- Normalization and positivity require

$$\sum_{k=1}^{K} \pi_k = 1 \qquad 0 \leqslant \pi_k \leqslant 1$$

- Can interpret the mixing coefficients as prior probabilities

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

Gaussian Mixture Model

- z uses a 1-of-K representation, we can also write this distribution in the form

$$p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$$

- Conditional distribution of x given a particular value for z is a Gaussian

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

- Joint distribution is given by p(z)p(x|z), and the marginal distribution of x is then obtained by summing the joint distribution over all possible states of z to given

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Gaussian Mixture Model

- Conditional probability of z given x
- use $\gamma(z_k)$ to denote $p(z_k = 1 \mid x)$, whose value can be found using Bayes' theorem

$$\gamma(z_k) \equiv p(z_k = 1|\mathbf{x}) = \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\mathbf{x}|z_j = 1)}$$

$$= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}.$$

- π_k as the prior probability of $z_k = 1$, and the quantity $\gamma(z_k)$ as the corresponding posterior probability once we have observed x.

Maximum Likelihood

Log of likelihood function:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

- Maximizing the log likelihood function for a Gaussian mixture model turns out to be a more complex problem than for the case of a single Gaussian.
- The difficulty arises from the presence of the summation over k that appears inside the logarithm, so that the logarithm function no longer acts directly on the Gaussian.

GMM Problems and Solutions

- How to maximize the log likelihood
  - solved by expectation-maximization (EM) algorithm
- How to avoid singularities in the likelihood function
  - solved by a Bayesian treatment
- How to choose number K of components
  - also solved by a Bayesian treatment

Expectation Maximization (EM) Algorithm

- Conditions for MLE: Setting the derivatives of ln p(X|π,μ,Σ)

in with respect to the means μk of the Gaussian components to zero, we obtain

$$0 = -\sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}}_{\gamma(z_{nk})} \boldsymbol{\Sigma}_k(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

rearranging we obtain:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n$$

where we have defined

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

Expectation Maximization (EM) Algorithm

- μ_k for the kth Gaussian component is obtained by taking a weighted mean of all of the points in the data set
- Weighting factor for data point x_n is given by the posterior probability $\gamma(z_{nk})$ that component k was responsible for generating xn
- If we set the derivative of ln p(X|π,μ,Σ) with respect to Σk to 0, and follow a similar line of reasoning, making use of the result for the maximum likelihood solution for the covariance matrix of a single Gaussian

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}$$

Expectation Maximization (EM) Algorithm

- Maximize ln p(X|π,μ,Σ) with respect to the mixing coefficients π_k with constraint

$$\sum_{k=1}^{K} \pi_k = 1 \qquad 0 \leqslant \pi_k \leqslant 1$$

- Using a Lagrange multiplier and maximizing the following quantity

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

which gives

$$0 = \sum_{n=1}^{N} \frac{\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda$$

- If we now multiply both sides by πk and sum over k, we find λ = −N.
- Rearranging we obtain

$$\pi_k = \frac{N_k}{N}$$

Expectation Maximization (EM) Algorithm

- We first choose some initial values for the means, covariances, and mixing coefficients.
- Then we alternate between the following two updates that we shall call the E step and the M step
- In the expectation step, or E step, we use the current values for the parameters to evaluate the posterior probabilities,
- We then use these probabilities in the maximization step, or M step, to re-estimate the means, covariances, and mixing
- In practice, the algorithm is deemed to have converged when the change in the log likelihood function, or alternatively in the parameters, falls below some threshold

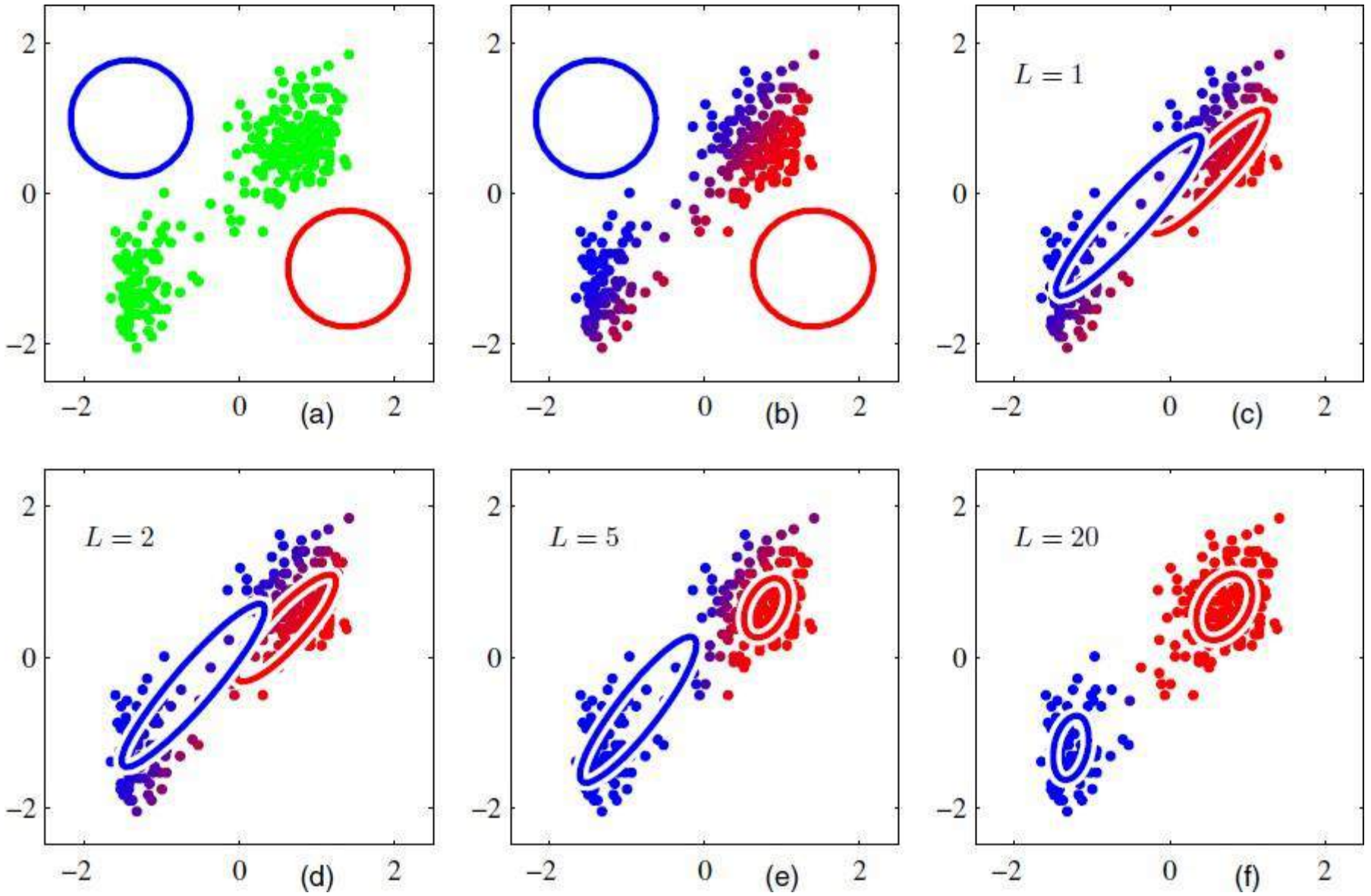
EM Algorithm – Informal Derivation

- The solutions are not closed form since they are coupled
- Suggests an iterative scheme for solving them:
  - make initial guesses for the parameters
  - alternate between the following two stages:
    1. E-step: evaluate responsibilities
    2. M-step: update parameters using ML results
- Each EM cycle guaranteed not to decrease the likelihood

Initialization

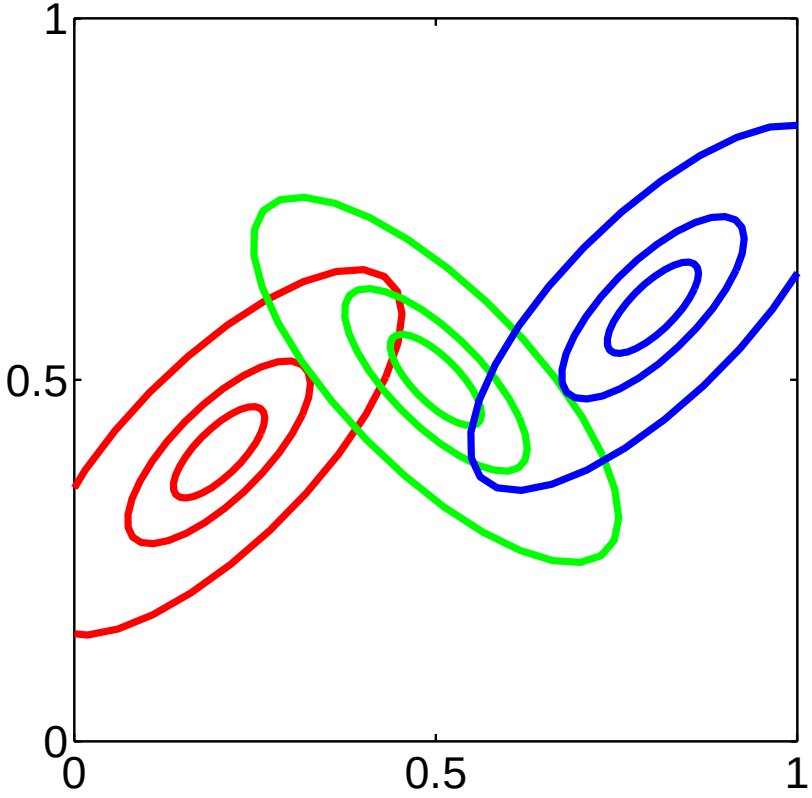
E step

M step

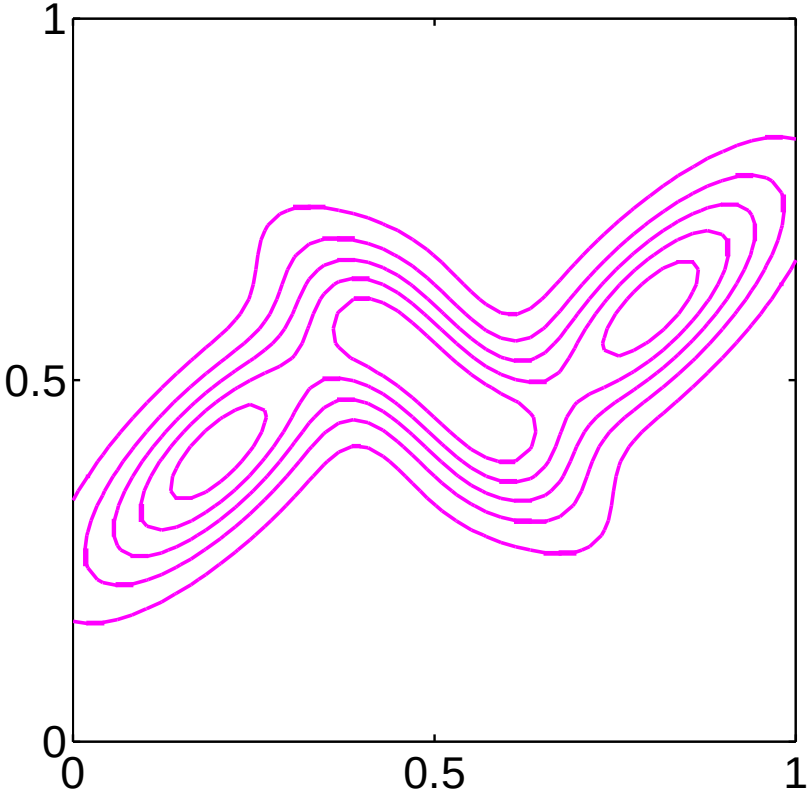


Figures from Chris Bishop

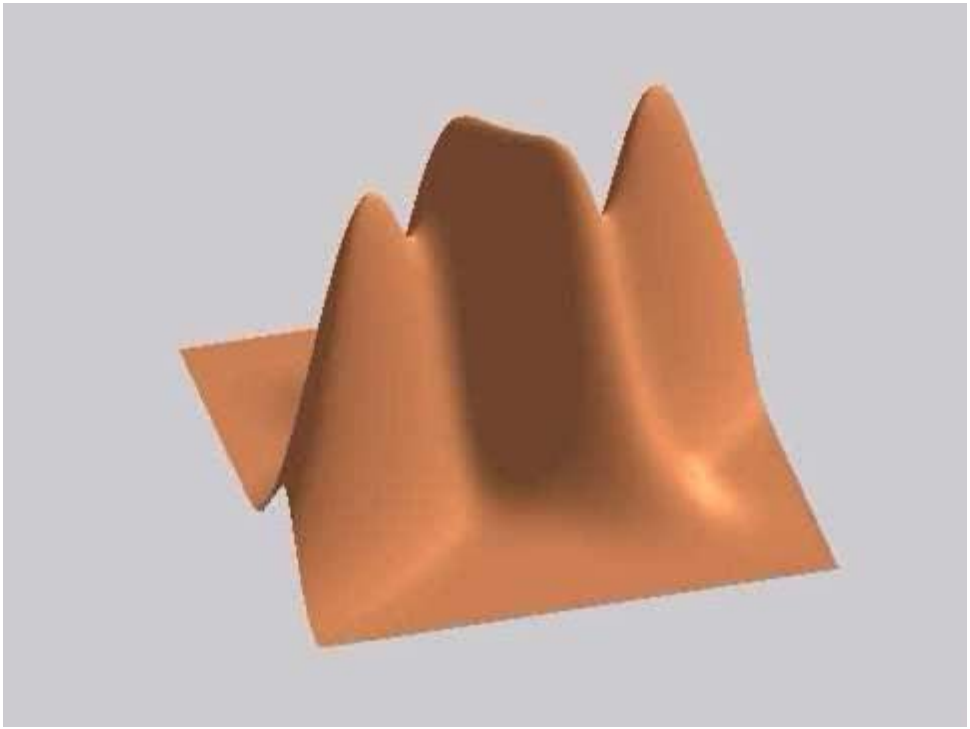
Example: Mixture of 3 Gaussians



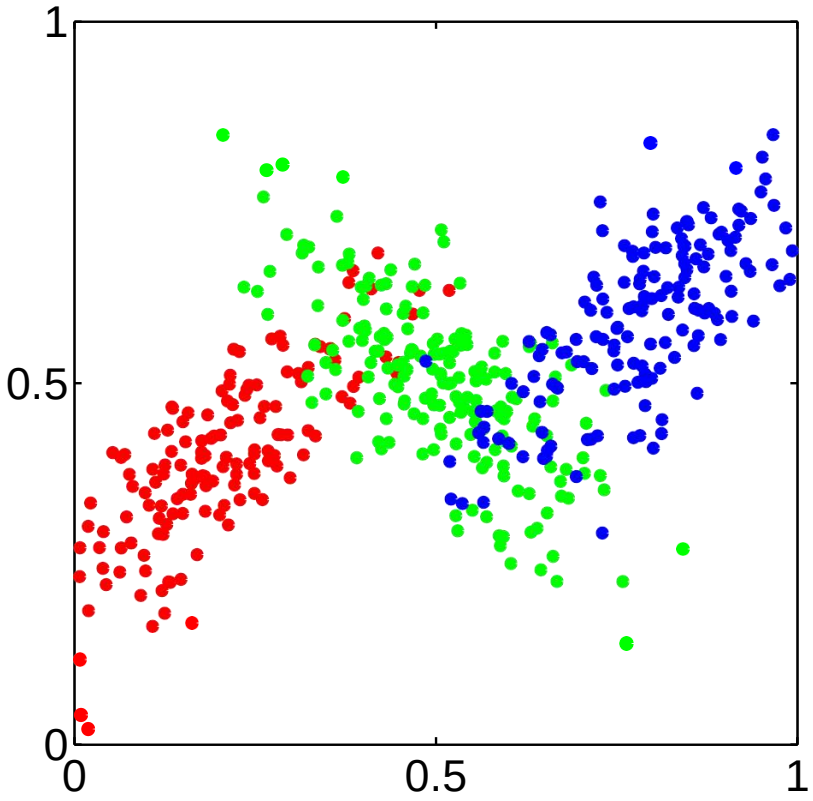
Contours of Probability Distribution



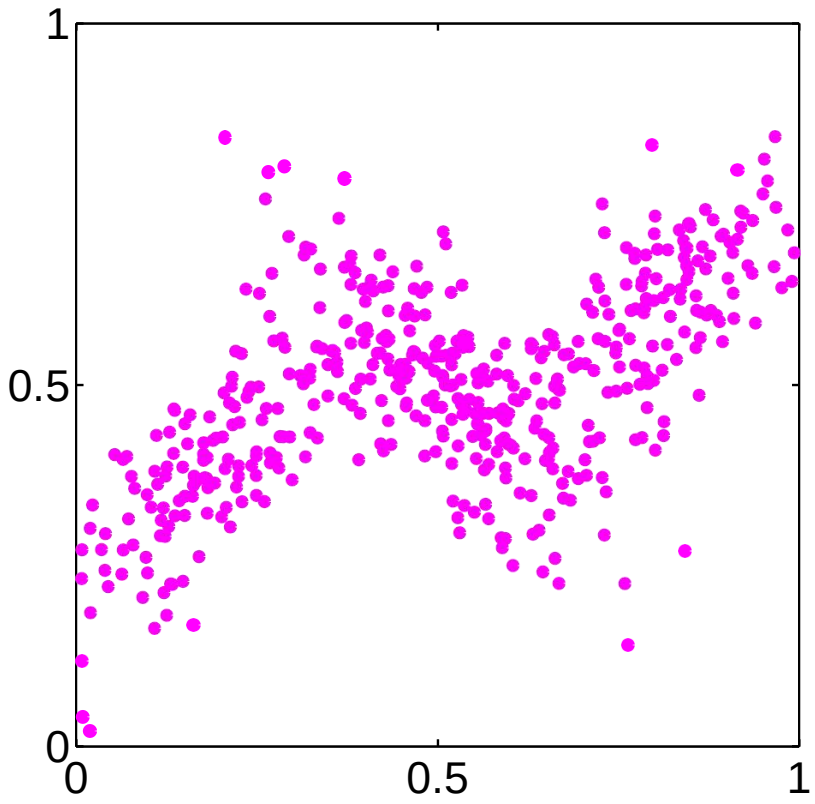
Surface Plot



Synthetic Data Set

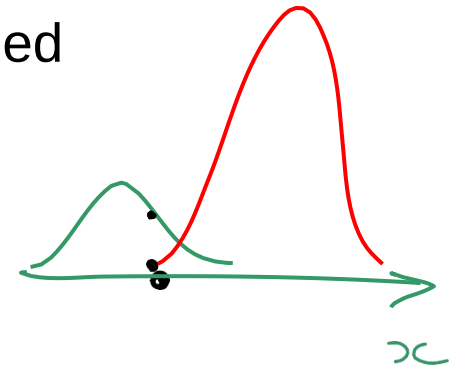


Synthetic Data Set Without Labels



Posterior Probabilities

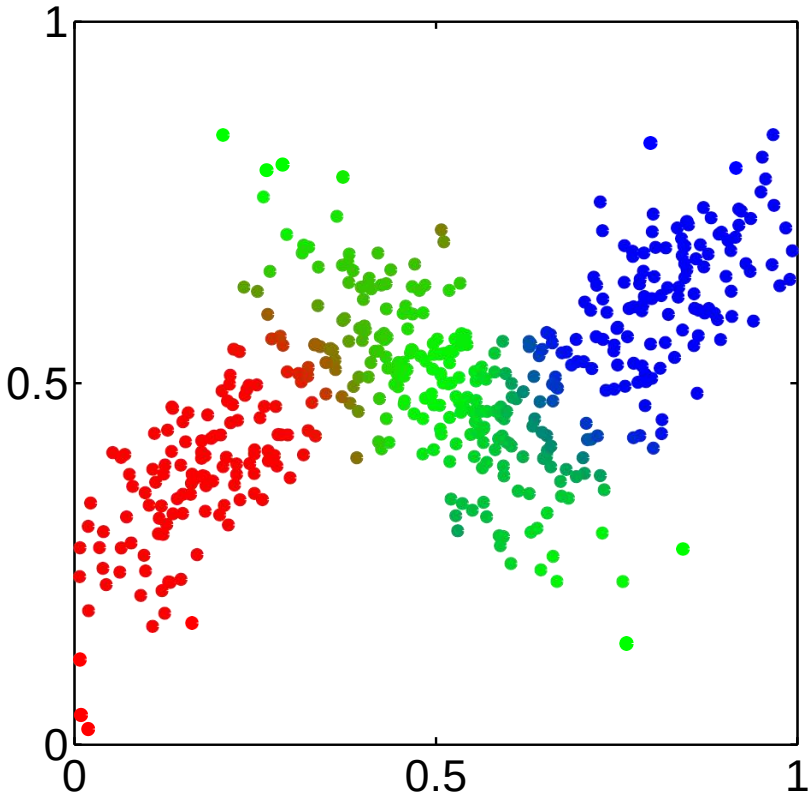
- We can think of the mixing coefficients as prior probabilities for the components
- For a given value of $\mathbf{x}$ we can evaluate the corresponding posterior probabilities, called *responsibilities*
- These are given from Bayes' theorem by



$$\gamma_k(\mathbf{x}) \equiv p(k|\mathbf{x}) = \frac{p(k)p(\mathbf{x}|k)}{p(\mathbf{x})}$$

$$= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

Posterior Probabilities (colour coded)



EM algorithm for GMM

1. Initialize the means μk, covariances Σk and mixing coefficients πk, and evaluate the initial value of the log likelihood.
2. **E step:** Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

EM algorithm for GMM

3. **M step:** Re-estimate the parameters using the current responsibilities

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})\mathbf{x}_n$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})\left(\mathbf{x}_n-\boldsymbol{\mu}_k^{\text{new}}\right)\left(\mathbf{x}_n-\boldsymbol{\mu}_k^{\text{new}}\right)^{\mathrm{T}}$$

$$\pi_k^{\text{new}} = \frac{N_k}{N} \qquad \text{where} \quad N_k=\sum_{n=1}^{N}\gamma(z_{nk})$$

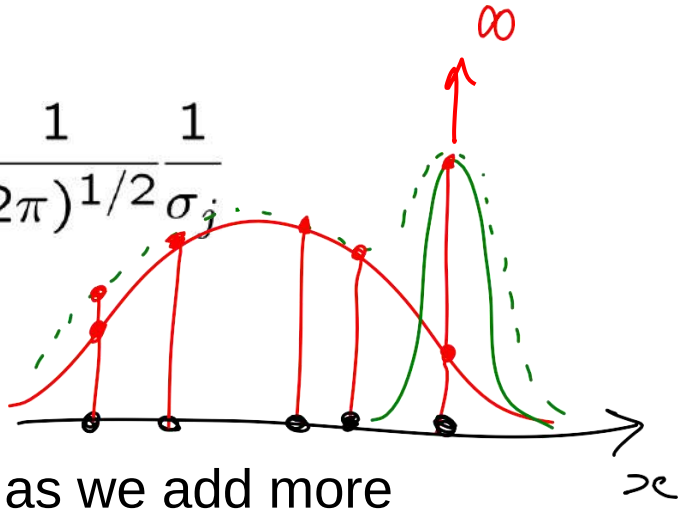
4. Evaluate the log likelihood

$$\ln p(\mathbf{X}|\boldsymbol{\mu},\boldsymbol{\Sigma},\boldsymbol{\pi}) = \sum_{n=1}^{N}\ln\left\{\sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right\}$$

Over-fitting in Gaussian Mixture Models

- Singularities in likelihood function when a component ‘collapses’ onto a data point:

$$\mathcal{N}(\mathbf{x}_n|\mathbf{x}_n, \sigma_j^2\mathbf{I}) = \frac{1}{(2\pi)^{1/2}}\frac{1}{\sigma_j}$$



then consider $\sigma_j \to 0$

- Likelihood function gets larger as we add more components (and hence parameters) to the model
  - not clear how to choose the number K of components

Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

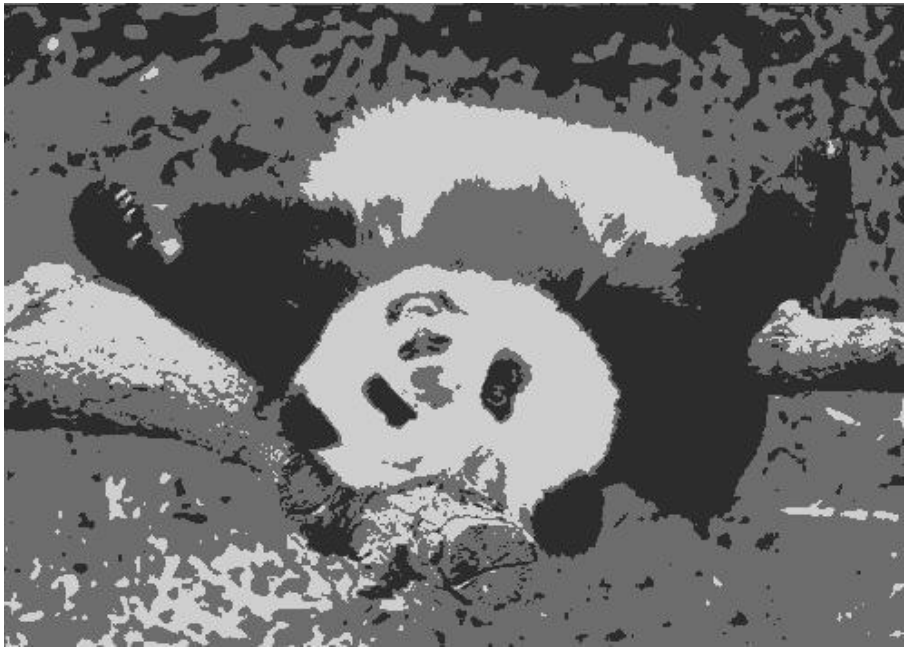
Grouping pixels based on **intensity** similarity

Feature space: intensity value (1-d)

Source: K. Grauman

K=2

K=3



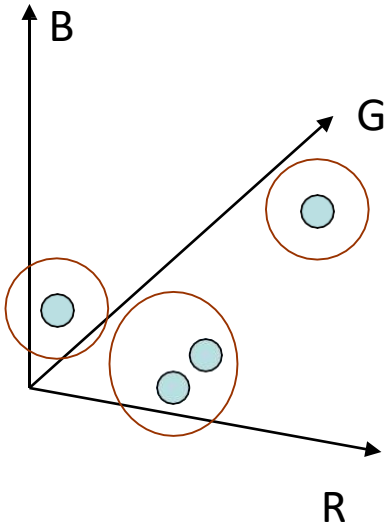
quantization of the feature space; segmentation label map

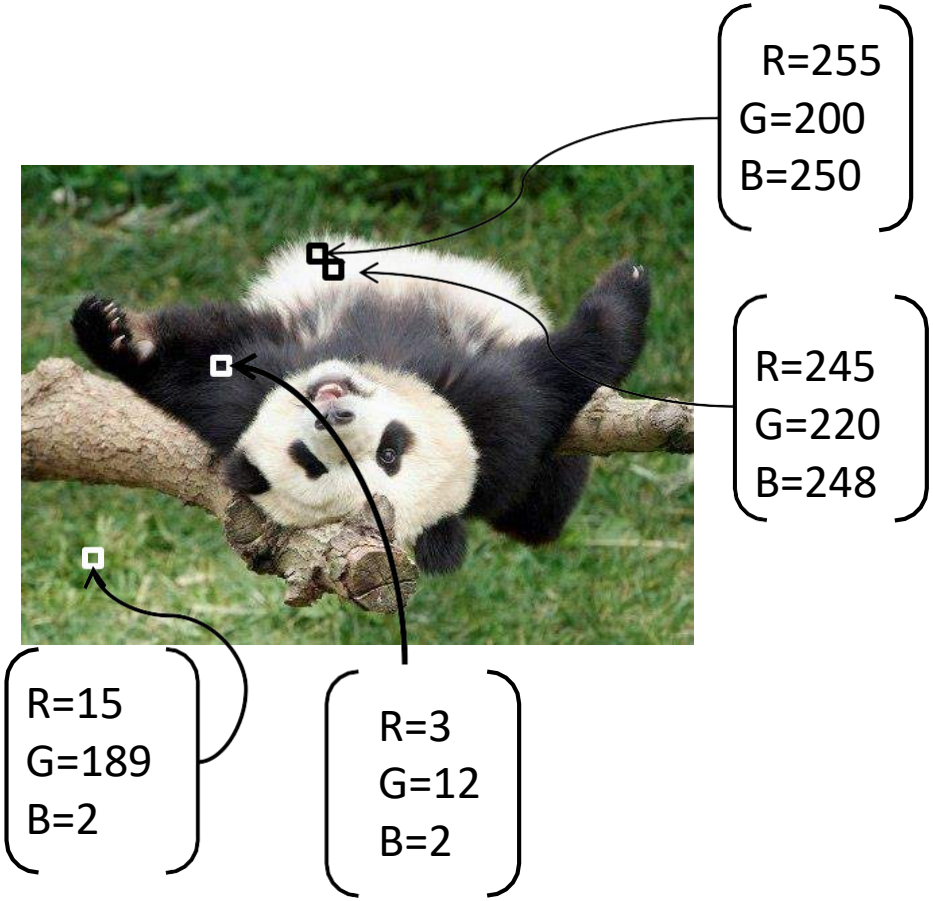
Source: K. Grauman

Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **color** similarity





Feature space: color value (3-d)

Source: K. Grauman

Application: Face Recognition

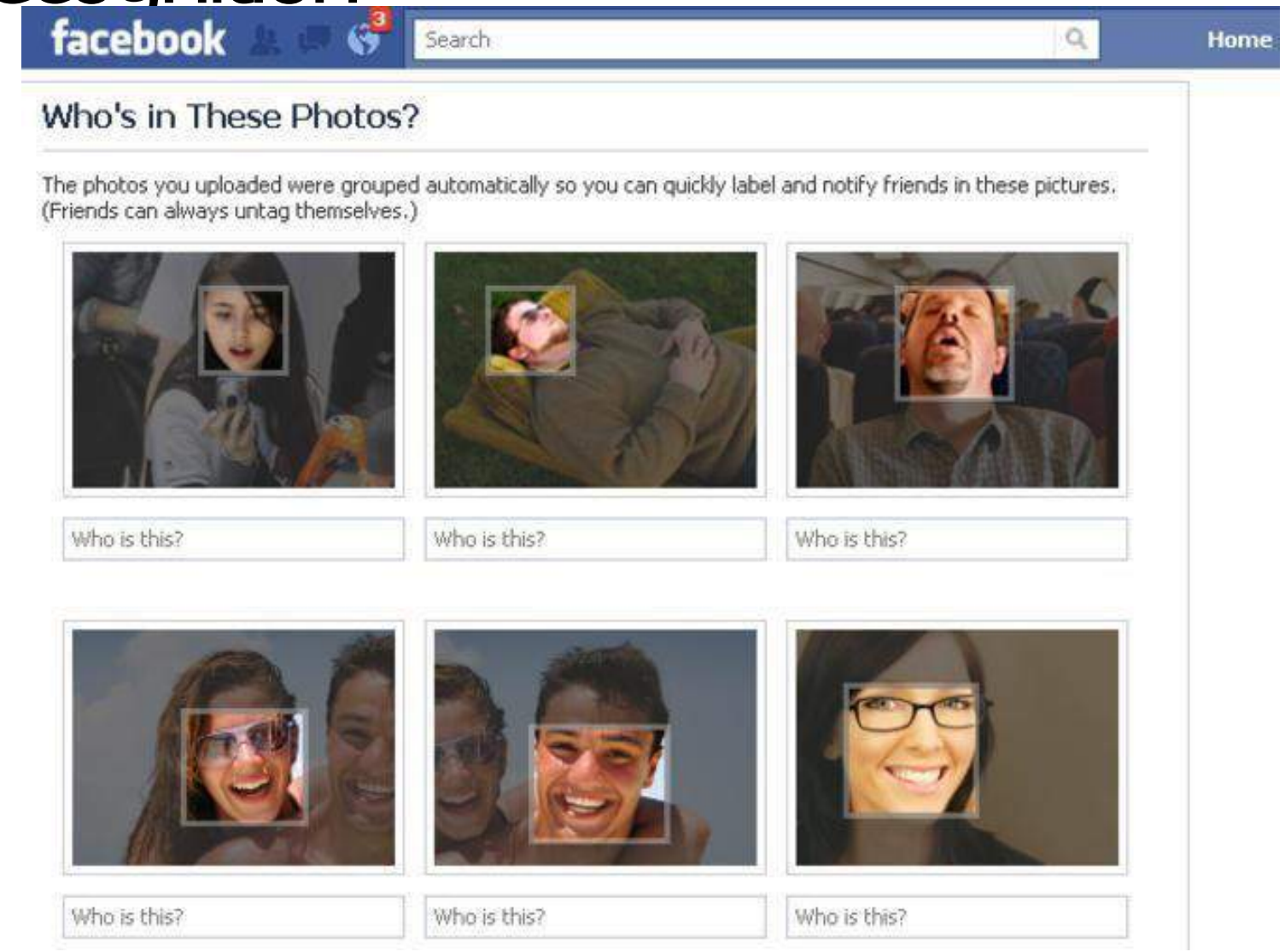


Image from cnet.com

References

Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition

https://www.youtube.com/watch?v=TG6Bh-NFhA0

https://www.youtube.com/watch?v=qMTuMa86NzU