Introduction to Computer Vision and Machine Learning

VISUM'19

Porto, PORTUGAL

Jose Costa Pereira

Research Scientist @Huawei R&D, London, UK

INESCTEC, Porto, Portugal

Thanks to Nuno Vasconcelos for the slide contents

Agenda

- Computer Vision
  - Cameras
  - Radiometry and light sources
  - Color
  - Filtering
  - Smoothing and noise
  - Edges
- Machine Learning
  - Bayes decision rule
  - Maximum Likelihood
  - Least Squares, regression
  - Clustering, k-means, EM
  - Principal component analysis
  - Support Vector Machines

Why Machine Learning?

- there are many processes in the world that are ruled by deterministic equations
  - e.g. $f = m.a$; linear systems and convolution, Fourier, various chemical laws
  - there may be some “noise”, “error”, “variability”, but we can live with those
  - we don’t need statistical learning
- learning is needed when
  - there is a need for predictions about variables in the world, Y
  - that depend on factors (other variables) X
  - in a way that is impossible or too difficult to derive an equation for.

Examples

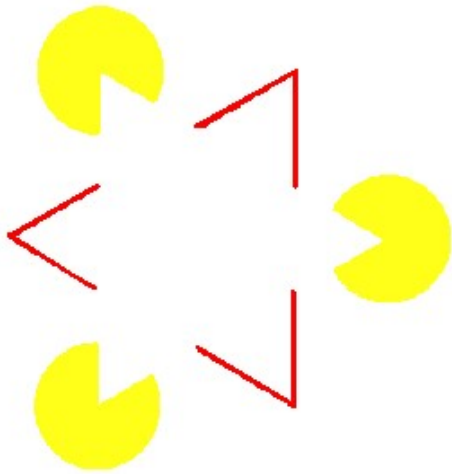
- data-mining view:
  - large amounts of data that do not follow deterministic rules
  - e.g. given an history of thousands of customer records and some questions that I can ask you, how can I predict that you will pay on time?
  - impossible to derive a theorem for this, must be learned
- while many associate learning with data-mining, it is by no means the only or even the more important application
- signal processing view:
  - signals combine in ways that depend on "hidden structure" (e.g. speech waveforms depend on language, grammar, etc.)
  - signals are usually subject to significant amounts of "noise" (which sometimes means "things we do not know how to model")

Examples (contd.)

- signal processing view:
  - e.g. the cocktail party problem,
  - although there are all these people talking, I can figure everything out.
  - how do I build a chip to separate the speakers?
  - model the hidden dependence as
    - a linear combination of independent sources
    - noise
  - many other examples in the areas of wireless communications, signal restoration, etc.

Examples (contd.)

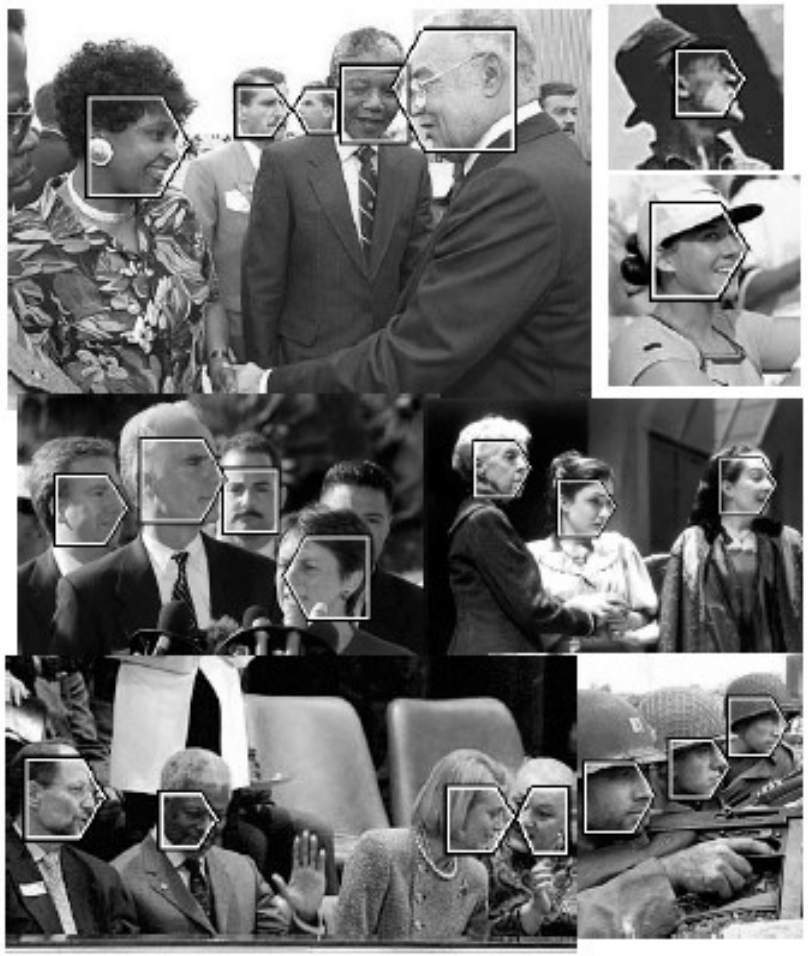
- perception/AI view:
  - it is a complex world, I cannot model everything in detail
  - rely on probabilistic models that explicitly account for the variability
  - use the laws of probability to make inferences, e.g. what is
    - P(burglar | alarm, no earthquake) is high
    - P(burglar | alarm, earthquake) is low
  - a whole field that studies "perception as Bayesian inference"
  - perception really just confirms what you already know
  - priors + observations = robust inference



Examples (contd.)

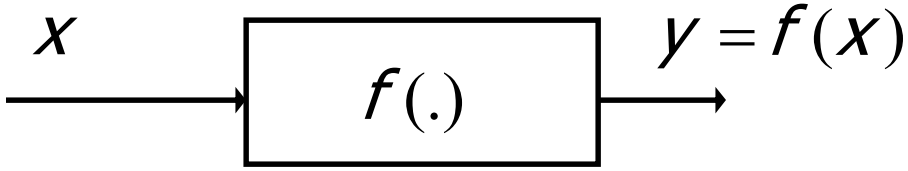
- communications view:
  - detection problems:

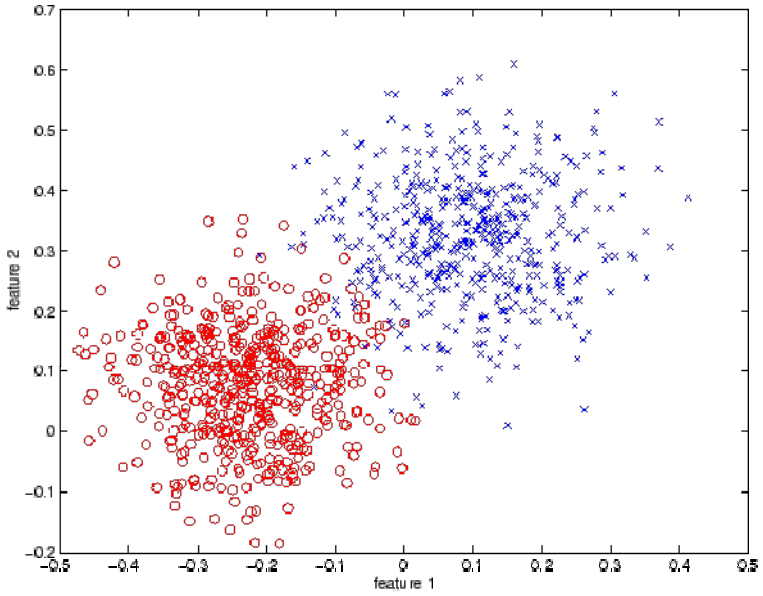
  - I see Y, and know something about the statistics of the channel. What was X?
  - this is the canonic detection problem that appears all over learning.
  - for example, face detection in computer vision: “I see pixel array Y. Is it a face?”



Statistical learning

- goal: given a collection of



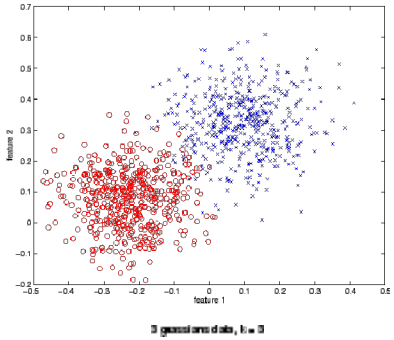


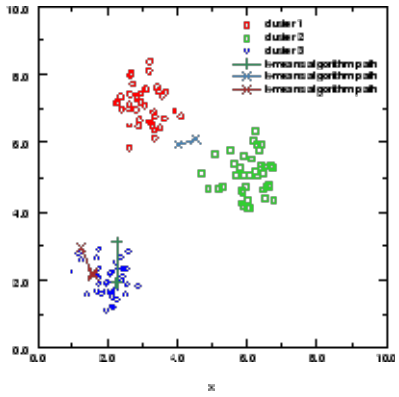
  data-points (X,Y), learn what function *f(.)* maps from X to Y

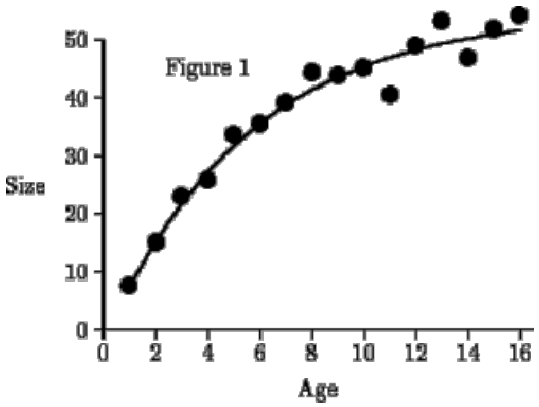
- this is called training.
- two major types of learning:
  - unsupervised: only X is known, usually referred to as clustering;
  - supervised: both are known during training, only X known at test time, usually referred to as classification or regression.

Supervised learning

- X can be anything, but the type of Y dictates the type of supervised learning problem
  - Y in {0,1} referred to as detection
  - Y in {0, ..., M-1} referred to as classification
  - Y real referred to as regression
- theory is quite similar, algorithms similar most of the time
- we will emphasize classification, but will talk about regression when particularly insightful





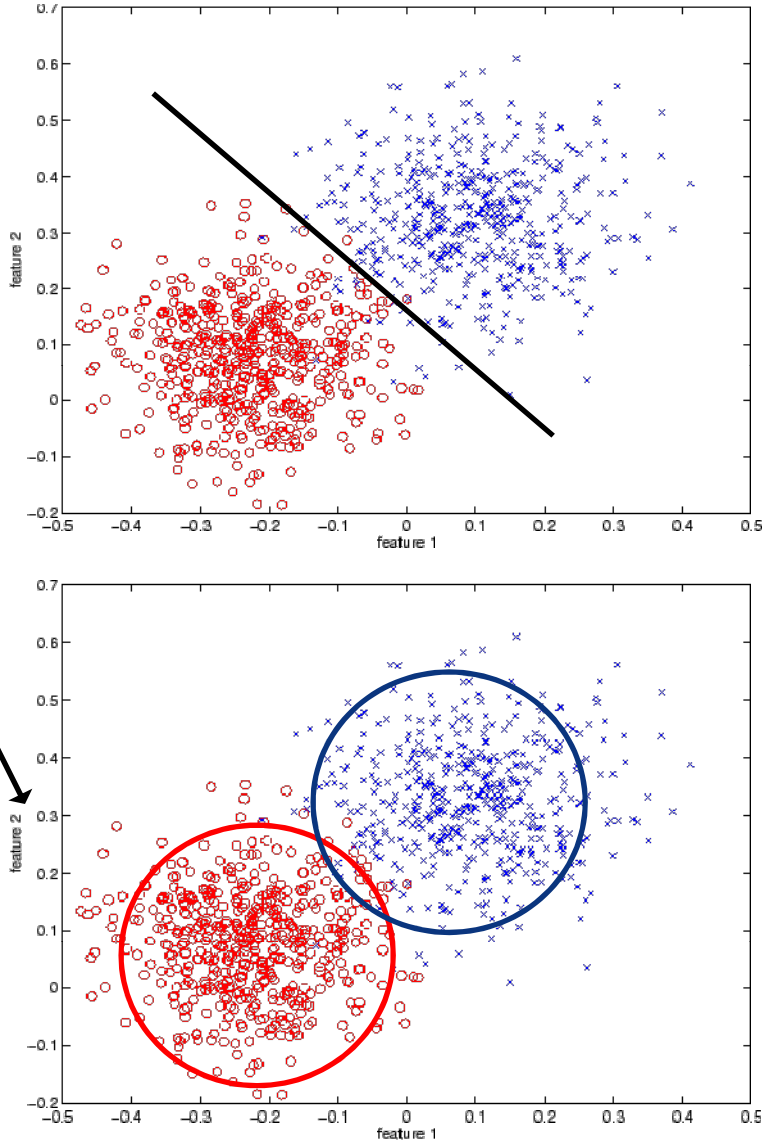


Example

- classifying fish:
  - fish roll down a conveyer belt
  - camera takes a picture
  - goal: is this a salmon or a sea-bass?
- Q: what is X? What features do I use to distinguish between the two fish?
- this is somewhat of an art-form. Frequently, the best is to ask experts.
- e.g. obvious! use length and scale width!

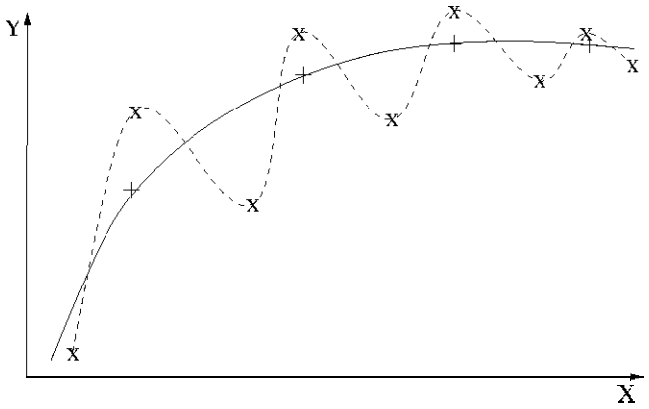
Classification/detection

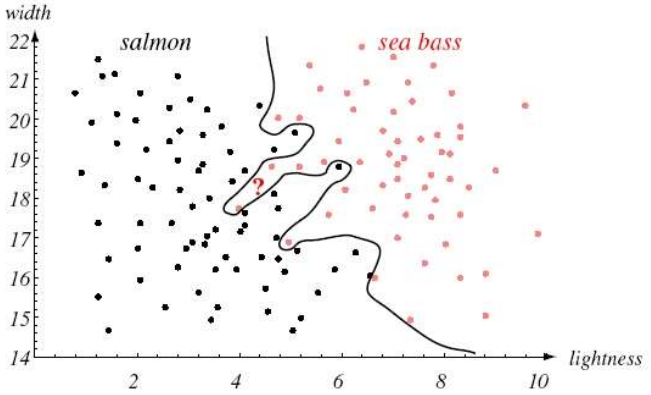
- two major types of classifiers
- discriminant: directly recover the decision boundary that best separates the classes;
- generative: fit a probability model to each class and then "analyze" the models to find the border.



Caution

- how do we know learning worked?
- we care about generalization, i.e. accuracy outside training set
- models that are too powerful can lead to over-fitting:
  - e.g. in regression I can always fit exactly *n* pts with polynomial of order *n-1.*
  - is this good? how likely is the error to be small outside the training set?
  - similar problem for classification

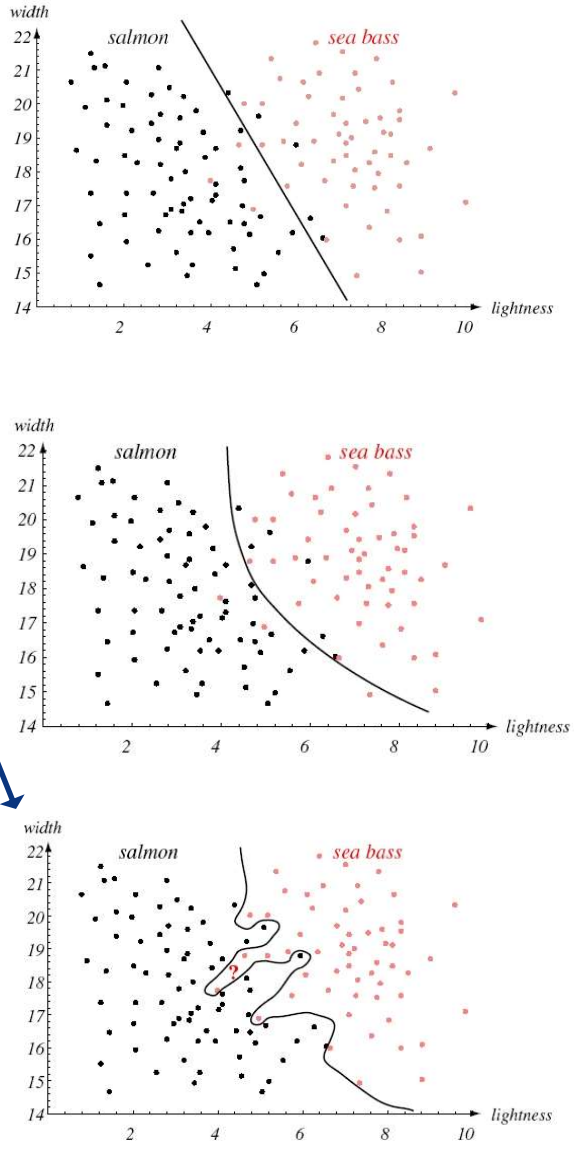




- fundamental rule: only test set results matter!!!

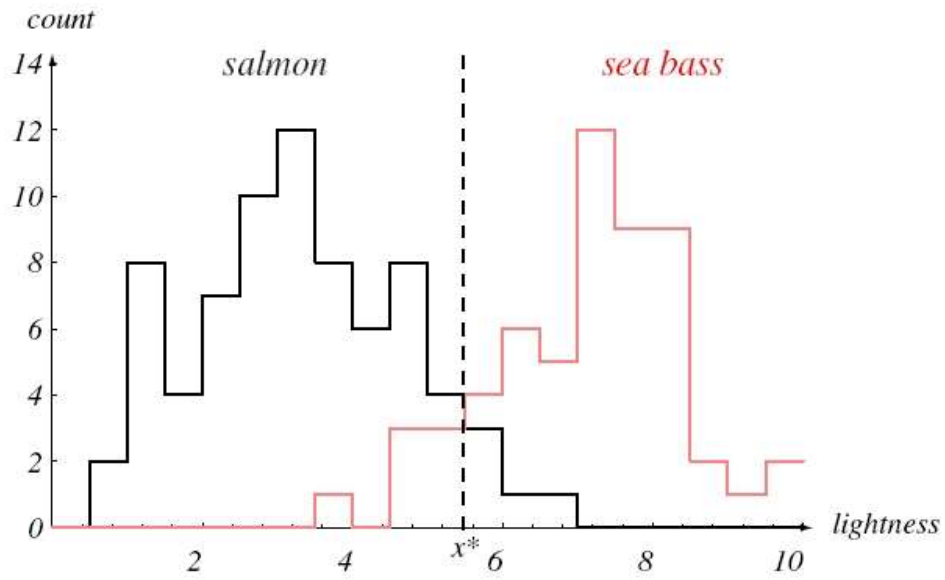
Generalization

- good generalization requires controlling the trade-off between training and test error
  - training error large, test error large
  - training error small, test error small
  - training error smallest, test error largest
- this trade-off is known by many names
- in the generative classification world it is usually due to the bias-variance trade-off of the class models



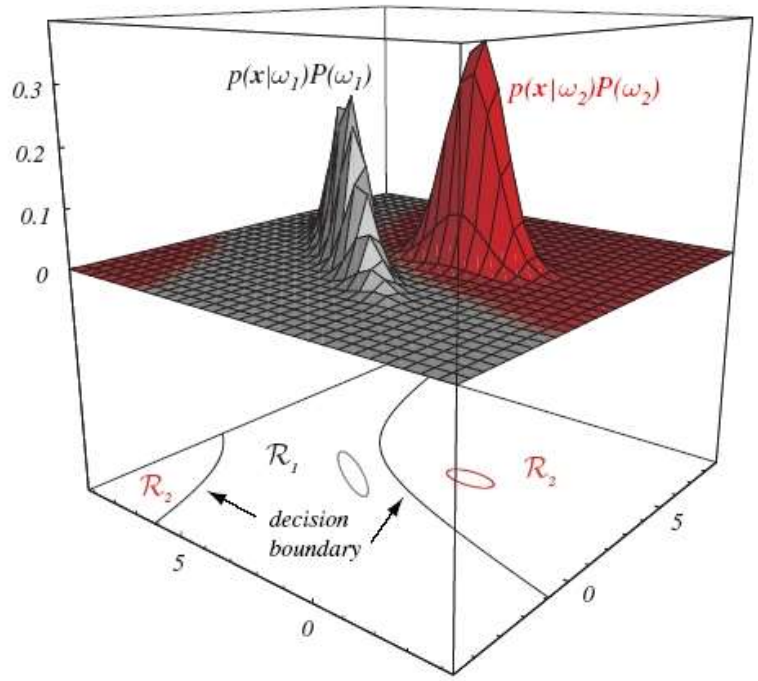
Generative learning

- each class is characterized by a probability density function (class conditional density)
- a model is adopted, e.g. a Gaussian
- training data used to estimate model parameters
- overall the process is referred to as density estimation
- the simplest example would be to use histograms



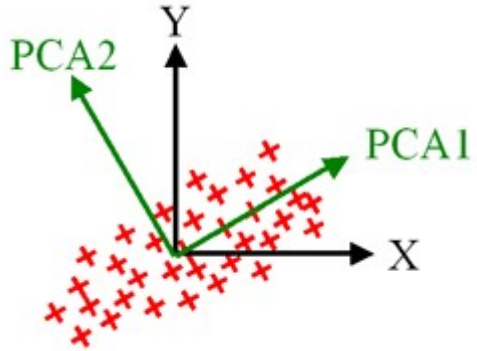
Decision rules

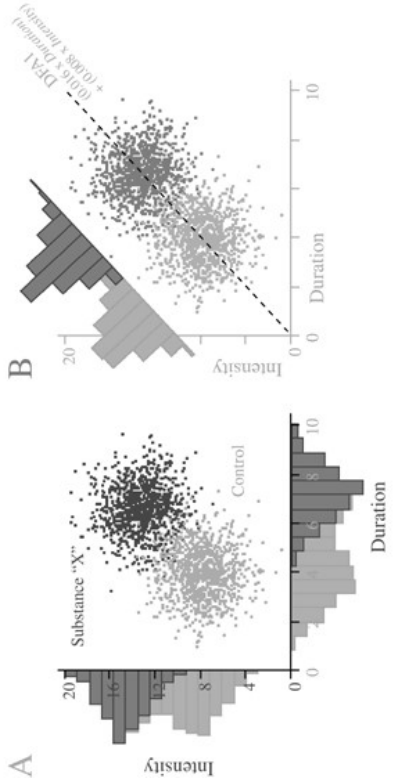
- given class models, Bayesian decision theory provides us with optimal rules for classification
- optimal here means minimum probability of error, for example
- we will
  - study BDT in detail, and
  - establish connections to other decision principles (e.g. linear discriminants)
  - show that Bayesian decisions are usually intuitive
- derive optimal rules for a range of classifiers



Features and dimensionality

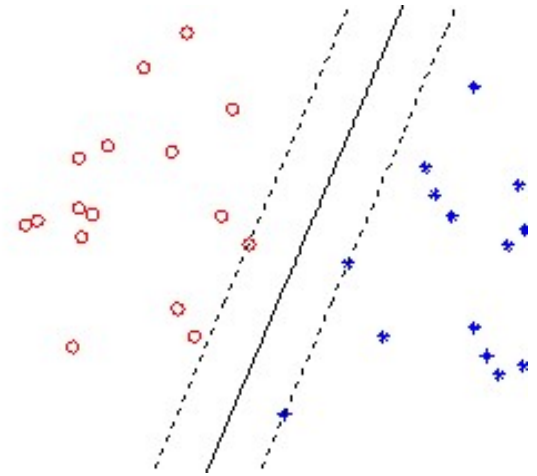
- for most of what we have seen so far
  - theory is well understood
  - algorithms available
  - limitations characterized
- usually, good features are an art-form
- we will look at some traditional techniques:
  - Principal components
  - Discriminant learning





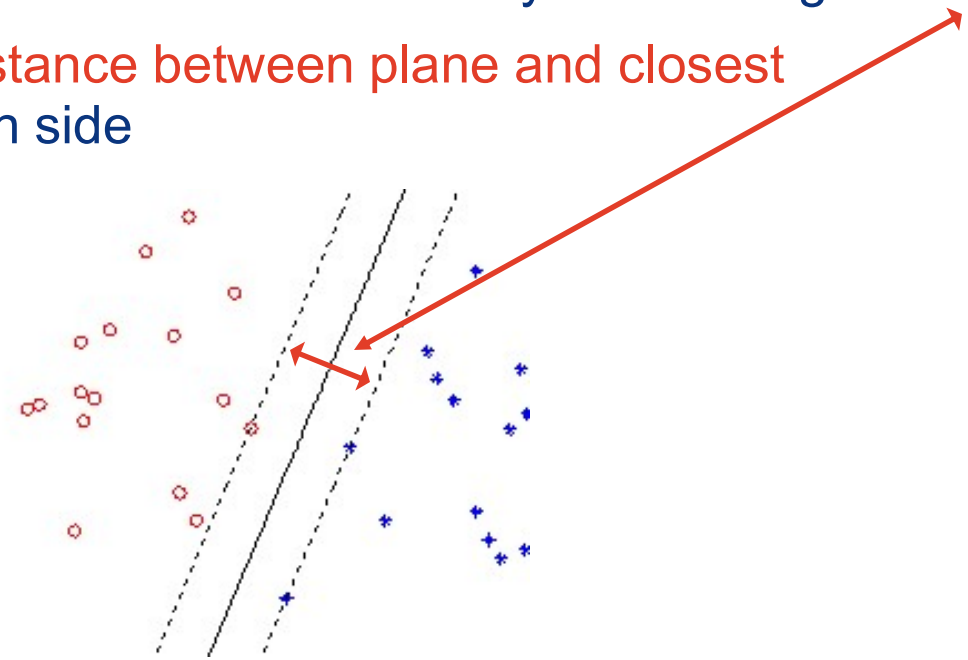
Discriminant learning

- instead of learning models and deriving boundary
- derive boundary directly
- many methods, the simplest case is the so-called hyperplane classifier
  - simply find the hyperplane that best separates the classes



Support vector machines

- how do we do this?
- A popular family of classifiers are called support vector machines
  - if the classes are separable,
  - the best performance is obtained by maximizing the margin
  - this is the distance between plane and closest point on each side

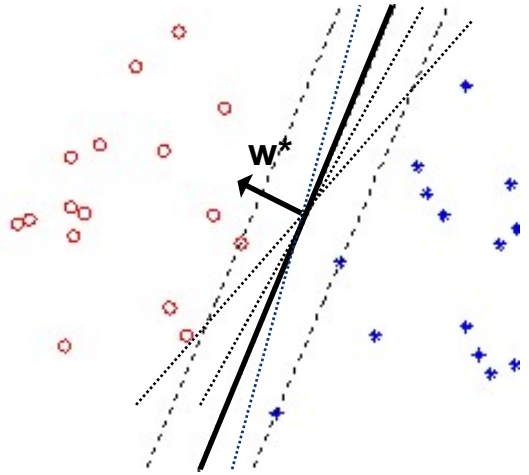


Support vector machines

- since, for separable classes, the training error can be made zero by classifying each point correctly
- this can be implemented by solving the optimization problem

$$w^* = \arg\max_{w} \text{ margin}(w)$$

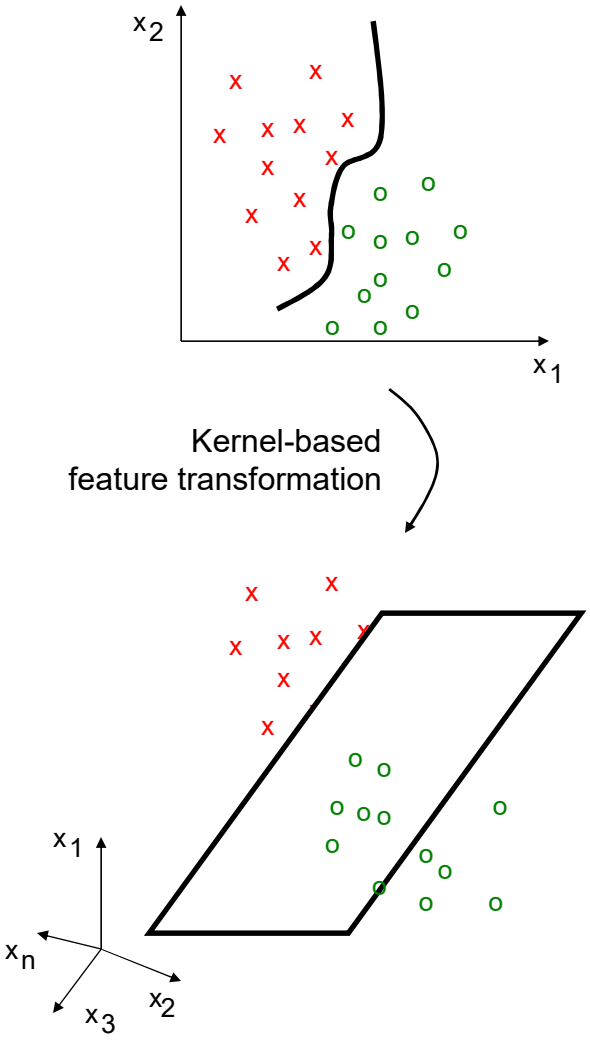
$$s.t.\ x_i \text{ is correctly classified}$$



- this is an optimization problem with n constraints, not trivial but solvable
- the solution is the support-vector machine (points on the margin are “support vectors”)
- what if a hyperplane is not good enough?

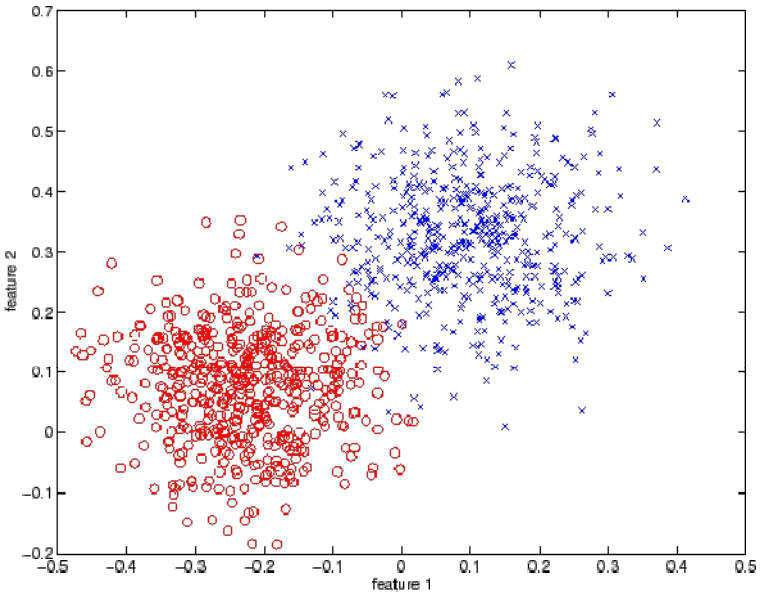
Kernels

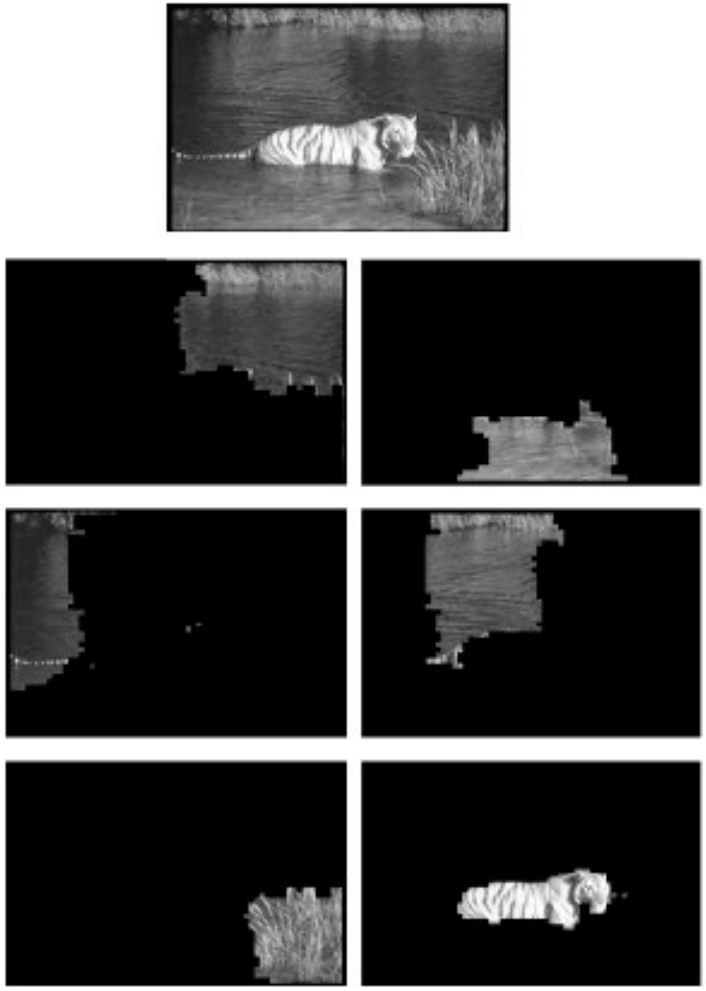
- the trick is to map the problem to a higher dimensional space:
  - non-linear boundary in original space
  - becomes hyperplane in transformed space
- this can be done efficiently by the introduction of a kernel function
- classification problem is mapped into a reproducing kernel Hilbert space
- kernels are at the core of the success of SVM classification
- most classical techniques (e.g. PCA, LDA, ICA, etc.) can be kernelized with significant improvement



Unsupervised learning

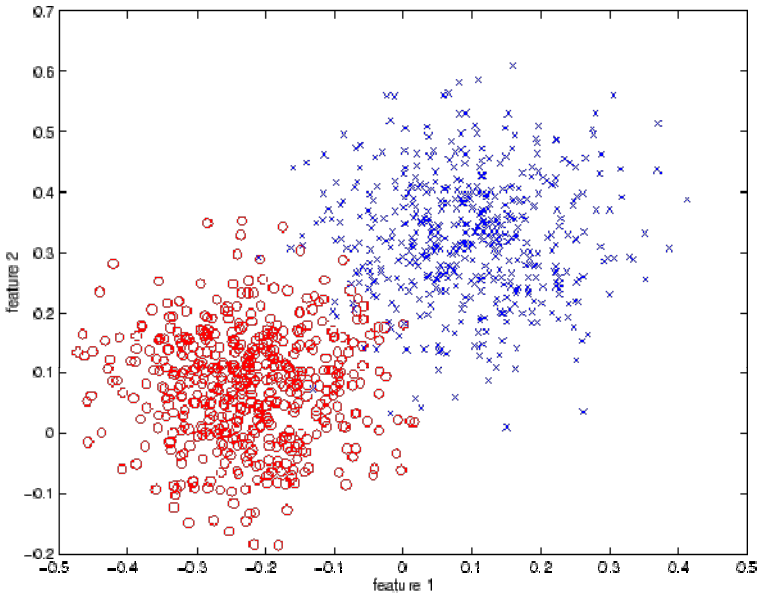
- so far, we have talked about supervised learning:
  - I know the class of each point
- in may problems this is not the case (e.g. image segmentation)





Unsupervised learning

- in these problems we are given X but not Y
- the standard algorithms for this are iterative:
  - start from best guess
  - given Y estimates fit class models
  - given class models re-estimate Ys



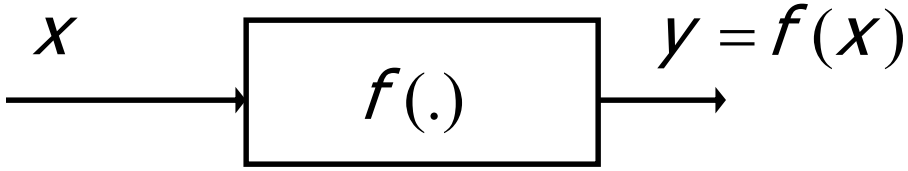
- the procedure usually converges to an optimal solution, although not necessarily the global optimum
- performance worst than that of supervised classifier, but this is the best we can do anyway…

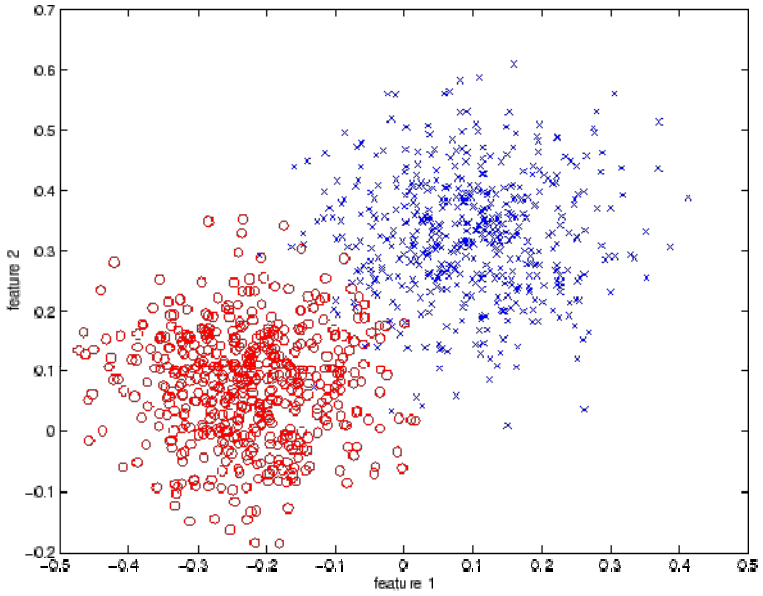
Summary

- statistical learning
  - tremendous amount of theory
  - but things invariably go wrong
  - too little data, noise, too many dimensions, training sets that do not reflect all possible variability, etc.
- good learning solutions require:
  - knowledge of the domain (e.g. “these are the features to use”)
  - knowledge of the available techniques, their limitations, etc.
- in the absence of either of these you will fail!
- we will cover the basics, but will talk about quite advanced concepts.
- easier scenario in which to understand them

Statistical learning

- goal: given a collection of





  data-points (X,Y), learn what function *f(.)* maps from X to Y

- this is called training
- two major types of learning:
  - unsupervised: only X is known, usually referred to as clustering;
  - supervised: both are known during training, only X known at test time, usually referred to as classification or regression.

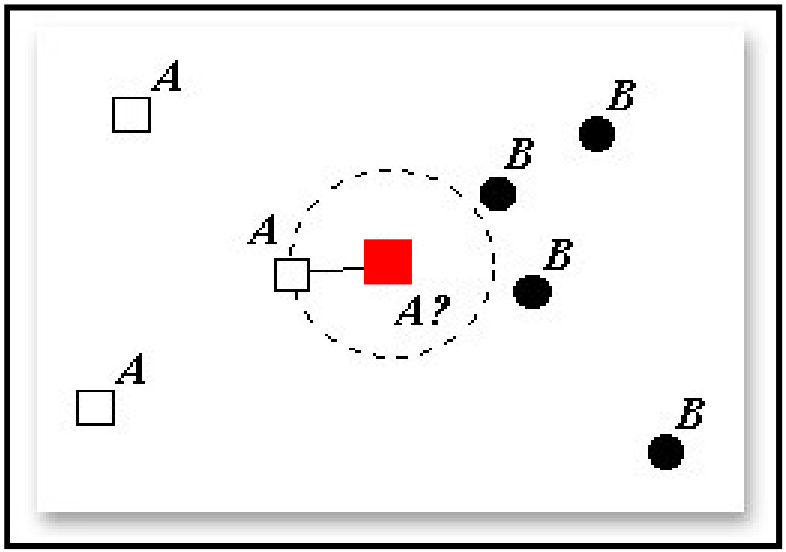
Nearest neighbor classifier

- we are considering supervised learning
- last time: nearest neighbor classifier
  - a training set $\mathcal{D} = \{(x_1,y_1), \ldots, (x_n,y_n)\}$
  - x_i is a vector of observations, y_i is the label
  - a vector x to classify
- the "decision rule" is

$$set \quad y = y_{i^*}$$

$$where$$

$$i^* = \arg\min_{i \in \{1,\ldots,n\}} d(x, x_i)$$



  - argmin means: "the i that minimizes the distance"

Metrics

- we have seen some examples:
  - in $\mathcal{R}^d$

$$\langle x, y \rangle = x^T y = \sum_{i=1}^{d} x_i y_i$$

continuous functions

$$\langle f(x), g(x) \rangle = \int f(x) g(x) dx$$

  - Euclidean norm

$$\|x\| = \sqrt{x^T x} = \sqrt{\sum_{i=1}^{d} x_i^2}$$

norm = energy

$$\|f(x)\| = \sqrt{\int f^2(x) dx}$$

  - Euclidean distance

$$d(x, y) = \|x - y\| = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

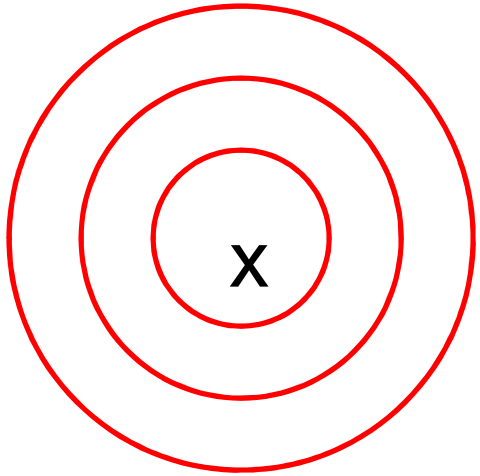
distance = energy of the difference

$$d(f, g) = \sqrt{\int [f(x) - g(x)]^2 dx}$$

Euclidean distance

- consider the Euclidean distance

$$d(x,y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$



- what are equidistant points to x?

$$d(x,y) = r \Leftrightarrow \sum_{i=1}^{d} (x_i - y_i)^2 = r^2$$

  - e.g. $(x_1 - y_1)^2 + (x_2 - y_2)^2 = r^2$

- the equidistant points to x are on spheres around x
- why would we need any other metric?

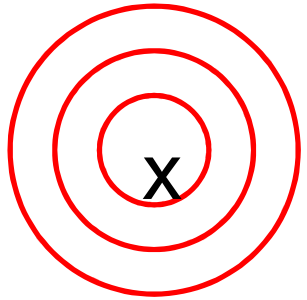
Dot products

- fish example:
  - features are L = fish length, W = scale width
  - measure L in meters and W in millimeters
    - typical L: 0.70m for salmon, 0.40m for sea-bass
    - typical W: 35mm for salmon, 40mm for sea-bass
  - I have three fish
    - $F_1 = (.7,35)$ $F_2 = (.4, 40)$ $F_3 = (.75, 37.8)$
    - F_1 clearly salmon, F_2 clearly sea-bass, F_3 looks like salmon
    - Yet: $d(F_1,F_3) = 2.8 > d(F_2,F_3) = 2.23$
  - there seems to be something wrong here
  - but if scale width is also measured in meters:
    - $F_1 = (.7,.035)$ $F_2 = (.4, .040)$ $F_3 = (.75, .0378)$
    - and now: $d(F_1,F_3) = .05 < d(F_2,F_3) = 0.35$
  - which seems to be right

Dot products

- suppose the scale width is also measured in meters:
  - I have three fish
    - F_1 = (.7,.035) F_2 = (.4, .040) F_3 = (.75, .0378)
    - and now

      $d(F_1,F_3) = .05 \quad < \quad d(F_2,F_3) = 0.35$
  - which seems to be right



- the problem is that the Euclidean distance depends on the units (or scaling) of each axis
  - e.g. if I multiply the second coordinate by 10

$$d(x,10y) = \sqrt{(x_1 - y_1)^2 + 100(x_2 - y_2)^2}$$

    its influence on the distance increases 100-fold!
- usually right units are not clear (e.g. car speed vs weight)

Dot products

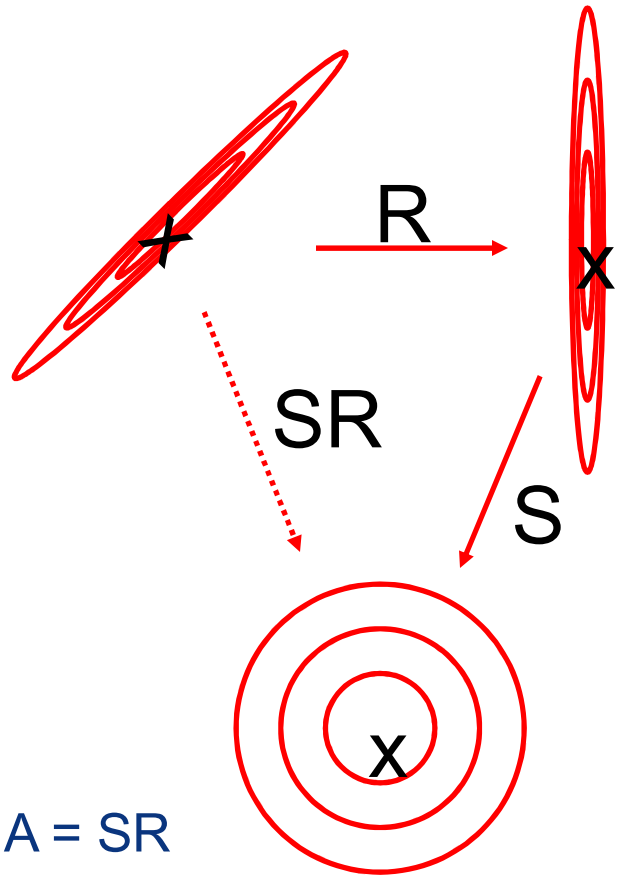
- the problem is that we need to work with the right units
- apply a transformation to a better scaled feature space

$$x' = Ax$$

- examples:
  - if A = R, R orthonormal, is equivalent to a rotation
  - another important case is scaling (A = S, diagonal)

$$\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \lambda_1 x_1 \\ \vdots \\ \lambda_n x_n \end{bmatrix}$$

  - we can even combine them by making A = SR



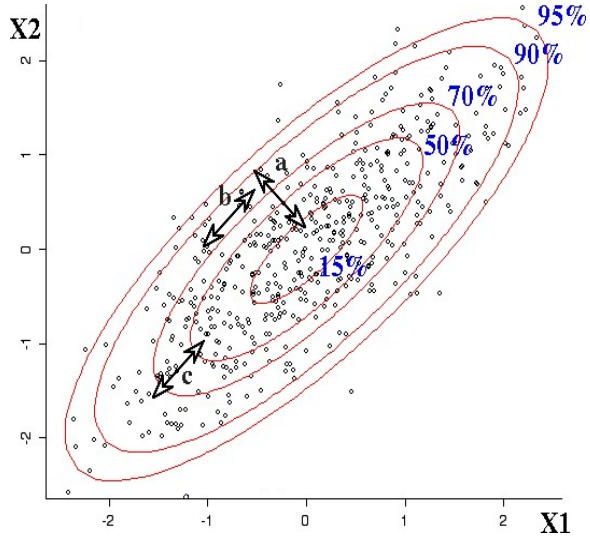
Dot products

- what is the Euclidean dot product in the new space?

$$\left(x'\right)^T y' = (Ax)^T Ax = x^T A^T Ax \Rightarrow \boxed{\langle x, y \rangle = x^T My}$$

- if I use this dot-product in the original space, I have the equivalent to working in the transformed space
- what is the right M?
  - we let the data tell us!
  - one possibility is to make it the inverse of the covariance matrix

$$\boxed{d(x,y) = (x-y)^T \Sigma^{-1} (x-y)}$$

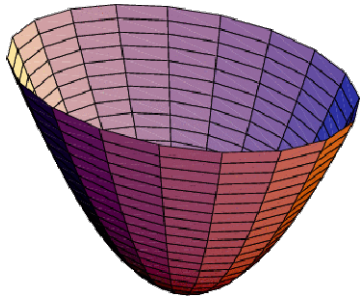


- this is the Mahalanobis distance
  - distance “adapted” to data covariance, “natural” units

The multivariate Gaussian

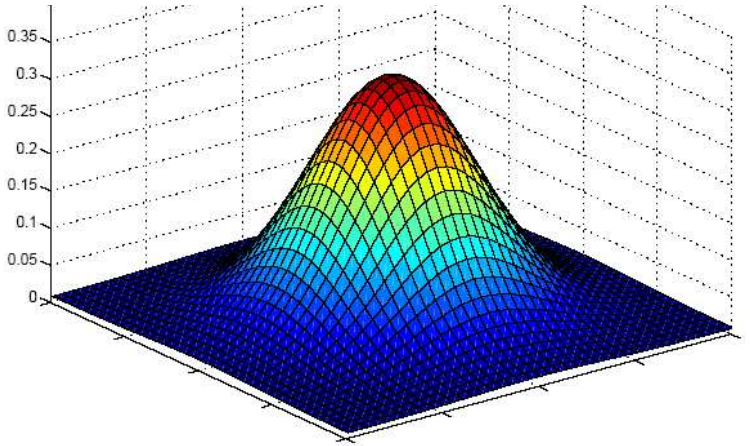
- note: using Mahalanobis = assuming Gaussian data
- Mahalanobis:

$$d(x,\mu) = (x-\mu)^T \Sigma^{-1} (x-\mu)$$



Gaussian:

$$P_X(x) = \frac{1}{\sqrt{(2\pi)^d \,|\Sigma|}} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right\}$$



- points of high probability are those of small distance to the center (mean)
- this can, once again, be interpreted as the right norm for a certain type of non-flat space

The multivariate Gaussian

- in fact, for Gaussian data, this tells us all we could possibly know
  - the pdf for a d-dimensional Gaussian of mean μ and covariance Σ is

$$P_X(x) = \frac{1}{\sqrt{(2\pi)^d \,|\Sigma|}} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}$$

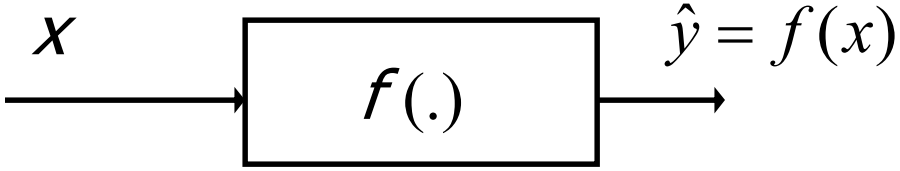
  - note that this is really just

$$P_X(x) = \frac{1}{K} \exp\left\{-\frac{1}{2} d(x,\mu)\right\}$$

  - i.e. the exp of the negative Mahalanobis distance
  - the constant K is needed only to make sure it integrates to 1

Optimal classifiers

- this discussion suggests that some metrics are better than others
- the meaning of "better" is connected to how well adapted the metric is to the properties of the data
- but can we be more rigorous? what do we mean by optimal?
- to talk about optimality we need to talk about cost or loss



$$L(y, \hat{y})$$

  - this the function that we want to minimize
  - depends on true y and prediction $\hat{y}$
  - tells us how good our predictor is

Loss functions

- classification problem
  - loss is function of classification errors. What errors can we have?
  - two types: false positives and false negatives
    - consider a face detection problem
    - if you see this and say

face

non-face

    - you have:

false-positive

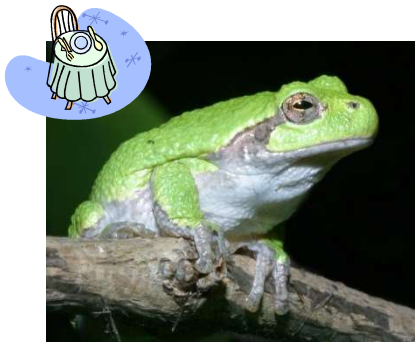
false-negative (miss)

  - obviously, we have the same sub-classes for non-errors
    - true-positives (hit) and true-negatives
  - the positive/negative part reflects what we say,
  - the true/false part reflects the real classes

Loss functions

- are some errors more important than others?
  - depends on the problem
  - consider a snake looking for lunch
  - the snake likes frogs
  - but dart frogs are highly poisonous
  - the snake must classify each frog it sees
    Y = {dart, regular}
  - the losses are clearly different

snake prediction	dart frog	regular frog
regular	∞	0
dart	0	10



Loss functions

- but not all snakes are the same
  - this one is a dart frog predator
  - it can still classify each frog it sees
    Y = {dart, regular}
  - it actually likes dart frogs better
  - but the other ones are good to eat too:

snake prediction	dart frog	regular frog
regular	10	0
dart	0	10

Risk

- the point is:
  - we have some loss function
  - we denote the cost of classifying X from class *i* as *j* by

$$L[i \rightarrow j]$$

  - one way to measure how good the classifier is
  - is the expected value of the loss, or Risk

$$R(x,i) = \sum_{j} L[j \rightarrow i] P_{Y|X}(j \mid x)$$

- this means
  - risk of classifying x as *i* is equal to
  - sum, over all classes, of the loss of classifying as *i* when truth is *j*
  - times probability that true class is *j* (given x)

Risk

- note that:
  - this immediately defines the optimal classifier
  - which is the one that minimizes the risk
  - for a given observation x, the optimal decision is

$$i^*(x) = \arg\min_i R(x,i)$$

$$= \arg\min_i \sum_j L[j \to i] P_{Y|X}(j \mid x)$$

  - and it has risk

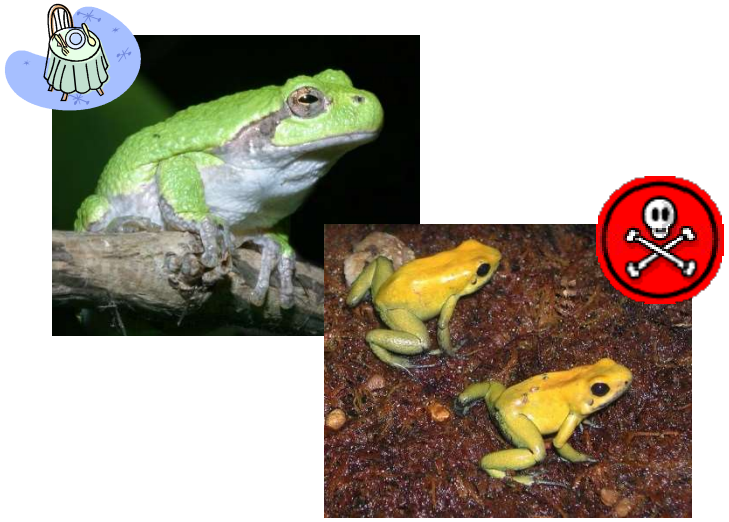
$$R^*(x) = \min_i \sum_j L[j \to i] P_{Y|X}(j \mid x)$$

Risk

- back to our example
  - the snake sees this

  - and makes probability assessments

$$P_{Y|X}(j \mid x) = \begin{cases} 0 & j = dart \\ 1 & j = regular \end{cases}$$



  - and computes the optimal decision

Risk

- info the snake has

$$P_{Y|X}(j \mid x) = \begin{cases} 0 & j = dart \\ 1 & j = regular \end{cases}$$

Losses

snake prediction	dart frog	regular frog
regular	∞	0
dart	0	10

- the risk of saying "regular" is

$$\sum_j L[j \to reg] P_{Y|X}(j \mid x) =$$

$$= L[reg \to reg] P_{Y|X}(reg \mid x) + L[dart \to reg] P_{Y|X}(dart \mid x)$$

$$= 0 \times 1 + \infty \times 0 = 0$$

Risk

- info the snake has

$$P_{Y|X}(j \mid x) = \begin{cases} 0 & j = dart \\ 1 & j = regular \end{cases}$$

Losses

snake prediction	dart frog	regular frog
regular	∞	0
dart	0	10

- the risk of saying "dart" is

$$\sum_j L[j \to dart] P_{Y|X}(j \mid x) =$$

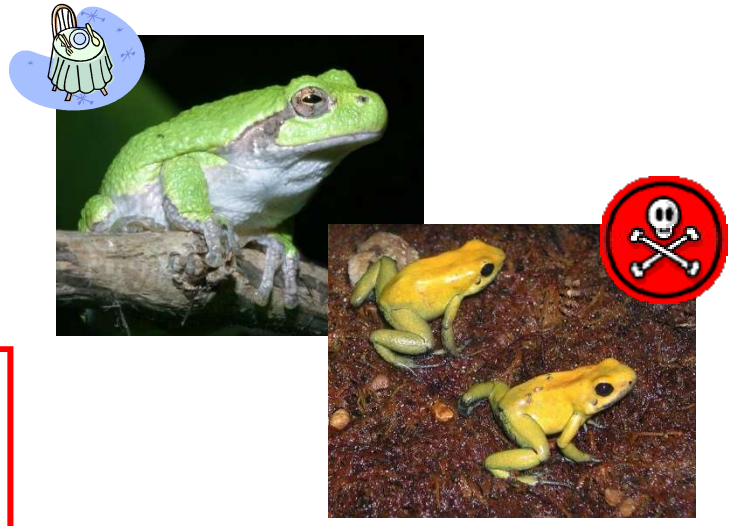
$$= L[reg \to dart] P_{Y|X}(reg \mid x) + L[dart \to dart] P_{Y|X}(dart \mid x)$$

$$= 10 \times 1 + 0 \times 0 = 10$$

- the snake says regular and has a good lunch ☺ (risk 0)

Risk

- What if the snake sees this?



- it has heard that dart frogs are colorful
- it assigns some probability to this being a dart

$$P_{Y|X}(j \mid x) = \begin{cases} 0.1 & j = dart \\ 0.9 & j = regular \end{cases}$$

Risk

- info the snake has

$$P_{Y|X}(j \mid x) = \begin{cases} 0.1 & j = dart \\ 0.9 & j = regular \end{cases}$$

Losses

snake prediction	dart frog	regular frog
regular	∞	0
dart	0	10

- the risk of saying "regular" is

$$\sum_j L[j \to reg] P_{Y|X}(j \mid x) =$$

$$= L[reg \to reg] P_{Y|X}(reg \mid x) + L[dart \to reg] P_{Y|X}(dart \mid x)$$

$$= 0 \times 0.9 + \infty \times 0.1 = \infty$$

Risk

- info the snake has

$$P_{Y|X}(j \mid x) = \begin{cases} 0.1 & j = dart \\ 0.9 & j = regular \end{cases}$$

- the risk of saying "dart" is

$$\sum_j L[j \to dart] P_{Y|X}(j \mid x) =$$
$$= L[reg \to dart] P_{Y|X}(reg \mid x) + L[dart \to dart] P_{Y|X}(dart \mid x)$$
$$= 10 \times 0.9 + 0 \times 0.1 = 9$$

Losses

snake prediction	dart frog	regular frog
regular	∞	0
dart	0	10

- the snake says "dart" and looks for another frog
  - even though this is a regular frog with 0.9 probability
- note that this is always the case unless $P_{Y|X}(dart|X) = 0$

Risk

- what about the snake that can eat darts?
  - the snake sees this

  - and makes probability assessments

$$P_{Y|X}(j \mid x) = \begin{cases} 0 & j = dart \\ 1 & j = regular \end{cases}$$

  - and computes the optimal decision

Risk

- info the snake has

$$P_{Y|X}(j \mid x) = \begin{cases} 0 & j = dart \\ 1 & j = regular \end{cases}$$

Losses

snake prediction	dart frog	regular frog
regular	10	0
dart	0	10

- the risk of saying "regular" is

$$\sum_j L[j \to reg] P_{Y|X}(j \mid x) =$$

$$= L[reg \to reg] P_{Y|X}(reg \mid x) + L[dart \to reg] P_{Y|X}(dart \mid x)$$

$$= 0 \times 1 + 10 \times 0 = 0$$

Risk

- info the snake has

$$P_{Y|X}(j \mid x) = \begin{cases} 0 & j = dart \\ 1 & j = regular \end{cases}$$

Losses

snake prediction	dart frog	regular frog
regular	10	0
dart	0	10

- the risk of saying "dart" is

$$\sum_j L[j \to dart] P_{Y|X}(j \mid x) =$$

$$= L[reg \to dart] P_{Y|X}(reg \mid x) + L[dart \to dart] P_{Y|X}(dart \mid x)$$

$$= 10 \times 1 + 0 \times 0 = 10$$

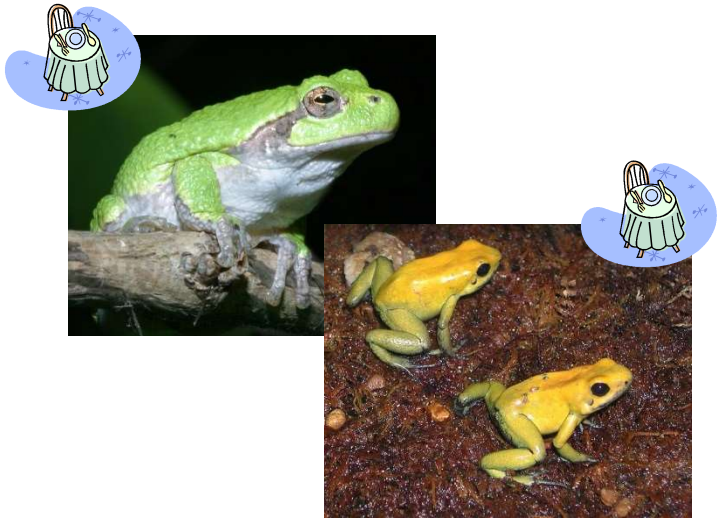
- snake says regular; consistent with probability estimates

Risk

- on the next week
  - the snake sees this

  - let's assume that it makes the same probability assignments as the other snake

$$P_{Y|X}(j \mid x) = \begin{cases} 0.1 & j = dart \\ 0.9 & j = regular \end{cases}$$



Risk

- info the snake has

$$P_{Y|X}(j \mid x) = \begin{cases} 0.1 & j = dart \\ 0.9 & j = regular \end{cases}$$

Losses

snake prediction	dart frog	regular frog
regular	10	0
dart	0	10

- the risk of saying "regular" is

$$\sum_j L[j \to reg] P_{Y|X}(j \mid x) =$$

$$= L[reg \to reg] P_{Y|X}(reg \mid x) + L[dart \to reg] P_{Y|X}(dart \mid x)$$

$$= 0 \times 0.9 + 10 \times 0.1 = 1$$

Risk

- info the snake has

$$P_{Y|X}(j \mid x) = \begin{Bmatrix} 0.1 & j = dart \\ 0.9 & j = regular \end{Bmatrix}$$

- the risk of saying "dart" is

$$\sum_j L[j \to dart] P_{Y|X}(j \mid x) =$$
$$= L[reg \to dart] P_{Y|X}(reg \mid x) + L[dart \to dart] P_{Y|X}(dart \mid x)$$
$$= 10 \times 0.9 + 0 \times 0.1 = 9$$

Losses

snake prediction	dart frog	regular frog
regular	10	0
dart	0	10

- the snake says "regular"
- once again, it is consistent with the probability estimates

Risk

- in summary, if both snakes have

$$P_{Y|X}(j \mid x) = \begin{cases} 0 & j = dart \\ 1 & j = regular \end{cases}$$

- then both say “regular”
- however, if

$$P_{Y|X}(j \mid x) = \begin{cases} 0.1 & j = dart \\ 0.9 & j = regular \end{cases}$$

  - the vulnerable snake says “dart”
  - the predator says “regular”
- the infinite loss for saying regular when frog is dart, makes the vulnerable snake much more cautious!

Risk

- the last case, i.e.
  - zero loss for no error and equal loss for two error types
- i.e. the "0/1" loss

$$L[i \to j] = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases}$$

snake prediction	dart frog	regular frog
regular	1	0
dart	0	1

- under this loss

$$i^*(x) = \arg\min_i \sum_j L[j \to i] P_{Y|X}(j \mid x)$$

$$= \arg\min_i \sum_{j \neq i} P_{Y|X}(j \mid x)$$

Risk

- i.e.

$$i^*(x) = \arg\min_i \sum_{j \neq i} P_{Y|X}(j \mid x)$$
$$= \arg\min_i \left[1 - P_{Y|X}(i \mid x)\right]$$
$$= \arg\max_i P_{Y|X}(i \mid x)$$

- and the optimal decision is
  - to pick the class that has largest posterior probability
  - given the observation x
- this is called the Bayes decision rule for the 0/1 loss
  - we will simplify our discussion by assuming this loss
  - but you should always be aware that other losses may be used

BDR

- consider the evaluation of the BDR for 0/1 loss

$$i^*(x) = \arg\max_i P_{Y|X}(i \mid x)$$

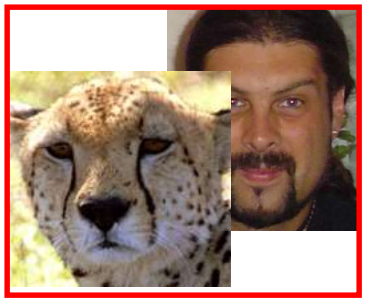
  - this is also called the maximum a posteriori probability rule
  - it turns out that it is usually not trivial to evaluate the posterior probabilities $P_{Y|X}(j|x)$
  - this is due to the fact that we are trying to infer the cause (class) from the consequence (observation x)
  - e.g. imagine that I want to evaluate

$$P_{Y|X}(person \mid \text{“has two eyes”})$$

  - this really depends on what the other classes are

Posterior probabilities

- if the two classes are “people” and “cars”
  - then $P_{Y|X}(person \mid \text{“has two eyes”}) = 1$
- but if the classes are “people” and “cats”
  - then $P_{Y|X}(person \mid \text{“has two eyes”}) = ½$
- how do we deal with this problem?
  - we note that it is much easier to infer consequence from cause
  - e.g., it is easy to infer that
    $$P_{X|Y}(\text{“has two eyes”} \mid person) = 1$$
  - this does not depend on any other classes
  - we do not need any additional information
  - given class, just count the frequency of observation



Bayes rule

- how do we go from $P_{X|Y}(x|j)$ to $P_{Y|X}(j|x)$?
- we use Bayes rule

$$P_{Y|X}(i \mid x) = \frac{P_{X|Y}(x \mid i) P_Y(i)}{P_X(x)}$$

  - and it turns out that this can be simplified
  - consider the two-class problem, i.e. Y=0 or Y=1
  - the BDR is

$$i^*(x) = \arg\max_i P_{Y|X}(i \mid x)$$

$$= \begin{cases} 0, & \text{if } P_{Y|X}(0 \mid x) \geq P_{Y|X}(1 \mid x) \\ 1, & \text{if } P_{Y|X}(0 \mid x) < P_{Y|X}(1 \mid x) \end{cases}$$

BDR

- pick “0” when $P_{Y|X}(0 \mid x) \geq P_{Y|X}(1 \mid x)$ and “1” otherwise
- using Bayes rule

$$P_{Y|X}(0 \mid x) \geq P_{Y|X}(1 \mid x) \Leftrightarrow$$

$$\frac{P_{X|Y}(x \mid 0)P_Y(0)}{P_X(x)} \geq \frac{P_{X|Y}(x \mid 1)P_Y(1)}{P_X(x)}$$

- noting that $P_X(x)$ is a non-negative quantity this is the same as
- pick “0” when

$$P_{X|Y}(x \mid 0)P_Y(0) \geq P_{X|Y}(x \mid 1)P_Y(1)$$

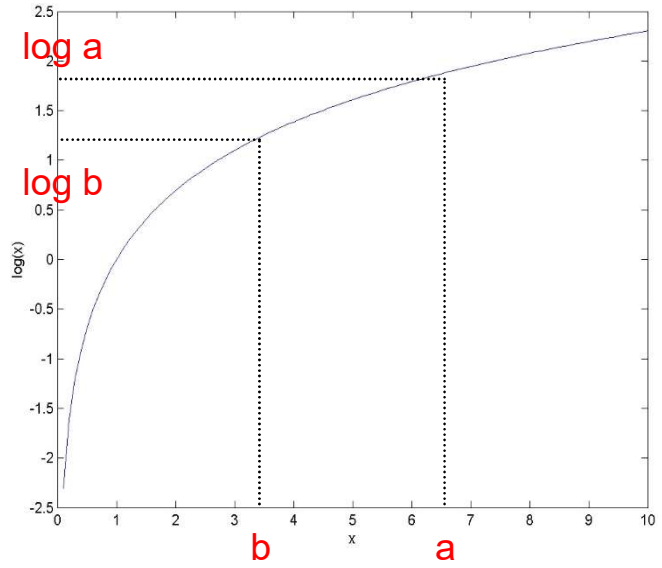
- i.e.

$$i^*(x) = \arg\max_i P_{X|Y}(x \mid i)P_Y(i)$$

The log trick

- sometimes the product is hard to compute
  - one helpful trick is to take logs
  - note that the log is a monotonically increasing function

$$a > b \Leftrightarrow \log a > \log b$$



  - from which

$$\begin{aligned} i^*(x) &= \arg\max_i P_{X|Y}(x \mid i) P_Y(i) \\ &= \arg\max_i \log\left(P_{X|Y}(x \mid i) P_Y(i)\right) \\ &= \arg\max_i \log P_{X|Y}(x \mid i) + \log P_Y(i) \end{aligned}$$

BDR

- in summary
  - for the 0/1 loss, the following three decision rules are
  - optimal and equivalent

- 1) $$i^*(x) = \arg\max_i P_{Y|X}(i \mid x)$$
- 2) $$i^*(x) = \arg\max_i \left[P_{X|Y}(x \mid i) P_Y(i) \right]$$
- 3) $$i^*(x) = \arg\max_i \left[\log P_{X|Y}(x \mid i) + \log P_Y(i) \right]$$

  - 1) is usually hard to use, 3) is frequently easier than 2)

The Gaussian classifier

- one important case is that of Gaussian classes
  - the pdf of class i is a Gaussian of mean μ_i and covariance Σ_i

$$P_{X|Y}(x \mid i) = \frac{1}{\sqrt{(2\pi)^d \mid \Sigma_i \mid}} \exp\left\{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right\}$$

- the BDR is

$$i^*(x) = \arg\max_i \left[-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i) - \frac{1}{2}\log(2\pi)^d \left|\Sigma_i\right| + \log P_Y(i) \right]$$

Implementation

- to design a Gaussian classifier:
  - start from a collection of datasets
  - $\mathcal{D}^{(i)} = \{x_1^{(i)}, ..., x_n^{(i)}\}$ set of examples from class i
  - for each class estimate the Gaussian parameters

$$\mu_i = \frac{1}{n}\sum_j x_j^{(i)} \qquad \Sigma_i = \frac{1}{n}\sum_j (x_j^{(i)} - \mu_i)(x_j^{(i)} - \mu_i)^T \qquad P_Y(i) = \frac{n}{T}$$

where T is the total number of examples (all classes)

- the BDR is

$$i^*(x) = \arg\max_i \left[-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i) - \frac{1}{2}\log(2\pi)^d |\Sigma_i| + \log P_Y(i) \right]$$

Gaussian classifier

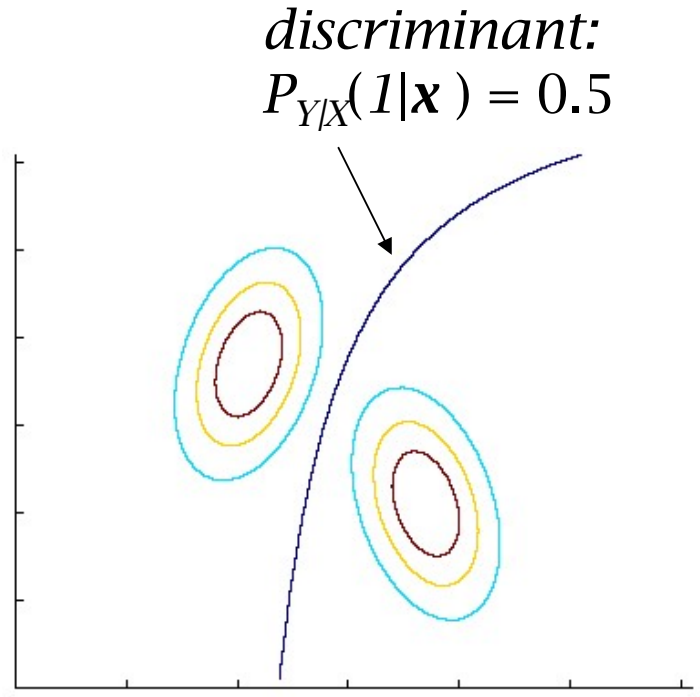
- this can be written as

$$i^*(x) = \arg\min_i \left[d_i(x, \mu_i) + \alpha_i \right]$$

with

$$d_i(x, y) = (x - y)^T \Sigma_i^{-1} (x - y)$$

$$\alpha_i = \log(2\pi)^d |\Sigma_i| - 2 \log P_Y(i)$$



- and can be seen as nearest neighbors with a funny metric
  - each class has its own distance
  - this is the sum of Mahalanobis for that class plus the α constant
  - we effectively have different metrics in different regions of the space

Gaussian classifier

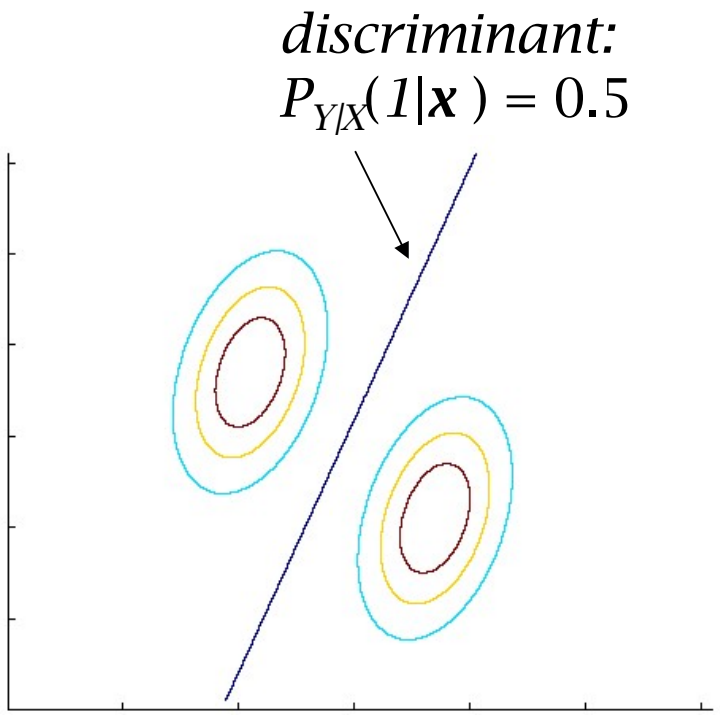
- a special case of interest is when
  - all classes have the same $\Sigma_i = \Sigma$

$$i^*(x) = \arg\min_i \left[d(x, \mu_i) + \alpha_i \right]$$

with

$$d(x, y) = (x - y)^T \Sigma^{-1} (x - y)$$

$$\alpha_i = -2 \log P_Y(i)$$



- note:
  - when all classes have equal probability, α_i can be dropped
  - this really becomes close to the NN classifier with Mahalanobis distance
  - instead of finding the nearest neighbor, it looks for the nearest "prototype" μ_i

Gaussian classifier

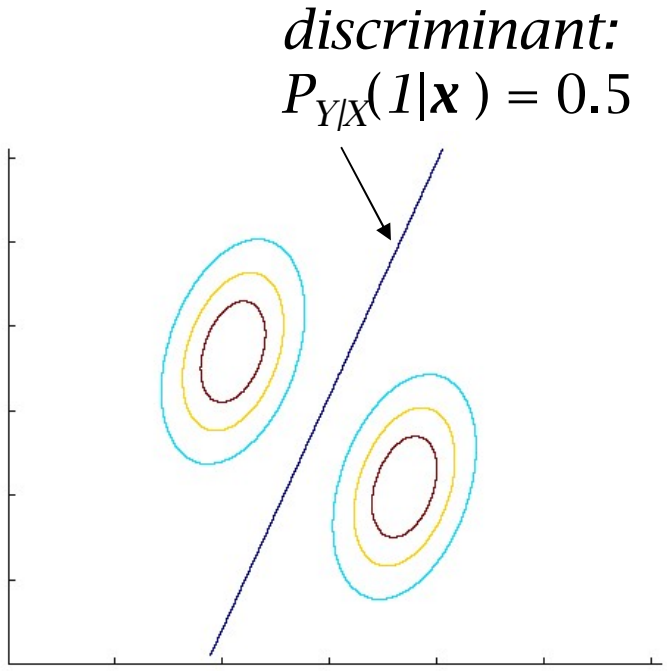
- $\Sigma_i=\Sigma$, two classes
  - one important property of this case is that decision boundary is a hyper-plane
  - can be shown by computing the set of points *x* such that

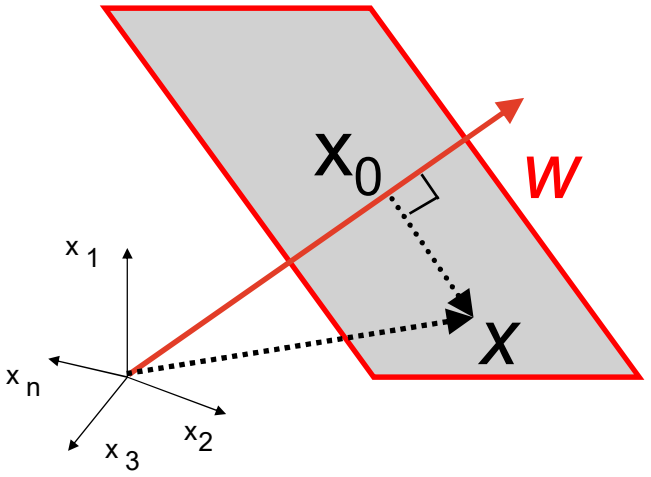
$$d(x,\mu_0)+\alpha_0 = d(x,\mu_1)+\alpha_1$$

  - and showing that they satisfy

$$w^T(x-x_0)=0$$

  - this is the equation of a plane with normal *W* that passes through x_0





Gaussian classifier

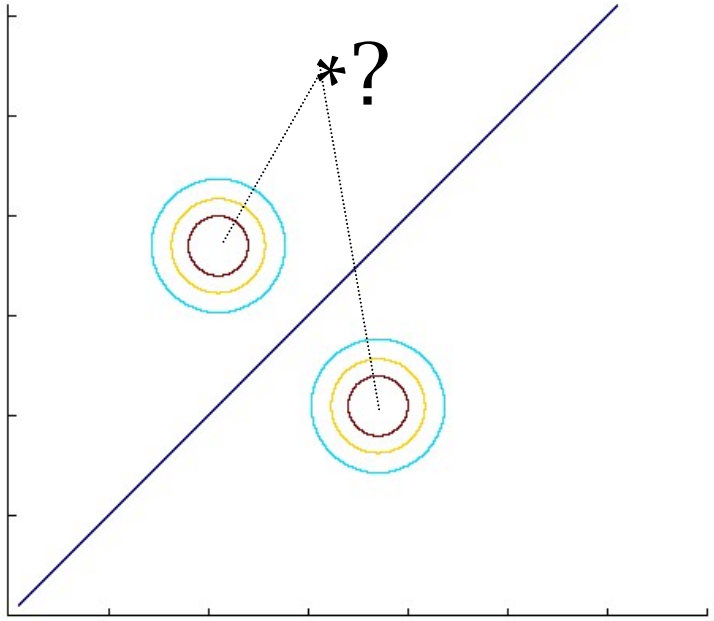
- if all the covariances are the identity $\Sigma_i = I$

$$i^*(x) = \arg\min_i \left[d(x, \mu_i) + \alpha_i \right]$$

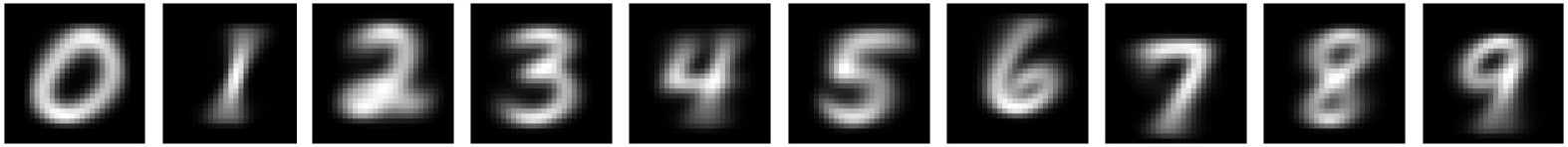
with

$$d(x, y) = \| x - y \|^2$$

$$\alpha_i = -2 \log P_Y(i)$$



- this is just template matching with class means as templates
  - e.g. for digit classification



  - compare complexity to nearest neighbors!

The sigmoid

- we have derived all of this from the log-based BDR

$$i^*(x) = \arg\max_i \left[\log P_{X|Y}(x \mid i) + \log P_Y(i) \right]$$

- when there are only two classes, it is also interesting to look at the original definition

$$i^*(x) = \arg\max_i g_i(x)$$

with

$$g_i(x) = P_{Y|X}(i \mid x) = \frac{P_{X|Y}(x \mid i) P_Y(i)}{P_X(x)}$$

$$= \frac{P_{X|Y}(x \mid i) P_Y(i)}{P_{X|Y}(x \mid 0) P_Y(0) + P_{X|Y}(x \mid 1) P_Y(1)}$$

The sigmoid

- note that this can be written as

$$i^*(x) = \arg\max_i g_i(x)$$

$$g_1(x) = 1 - g_0(x)$$

$$g_0(x) = \frac{1}{1 + \frac{P_{X|Y}(x \mid 1) P_Y(1)}{P_{X|Y}(x \mid 0) P_Y(0)}}$$

- and, for Gaussian classes, the posterior probabilities are

$$g_0(x) = \frac{1}{1 + \exp\{d_0(x - \mu_0) - d_1(x - \mu_1) + \alpha_0 - \alpha_1\}}$$

- where, as before,

$$d_i(x, y) = (x - y)^T \Sigma_i^{-1} (x - y)$$

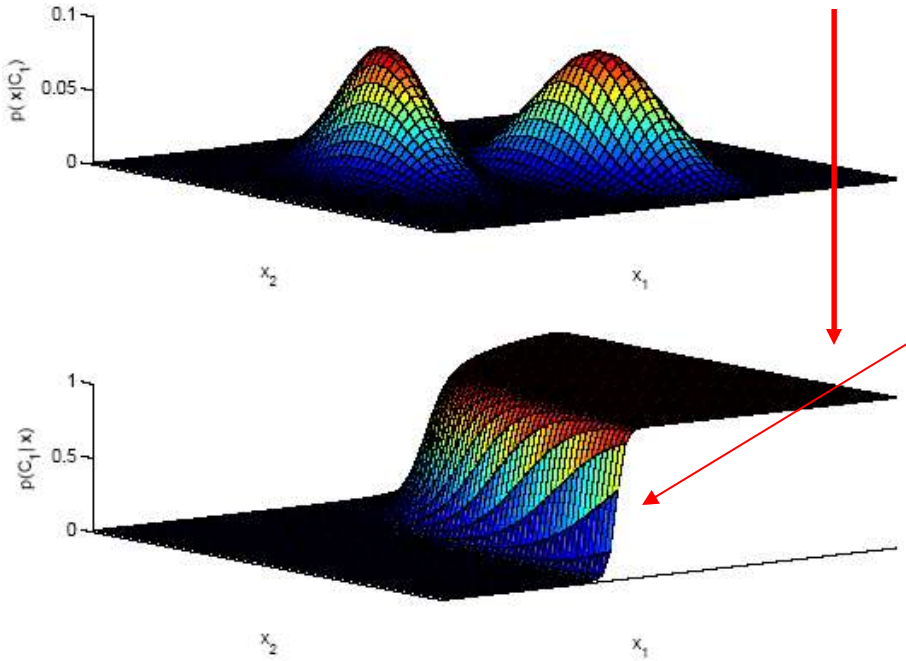
$$\alpha_i = \log(2\pi)^d |\Sigma_i| - 2 \log P_Y(i)$$

The sigmoid

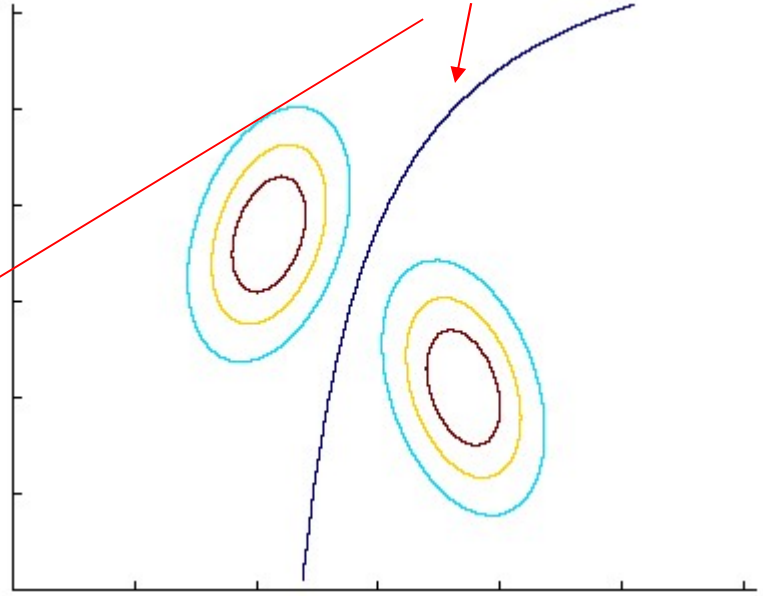
- the posterior

$$g_0(x) = \frac{1}{1+\exp\{d_0(x-\mu_0) - d_1(x-\mu_1) + \alpha_0 - \alpha_1\}}$$

- is a sigmoid and looks like this

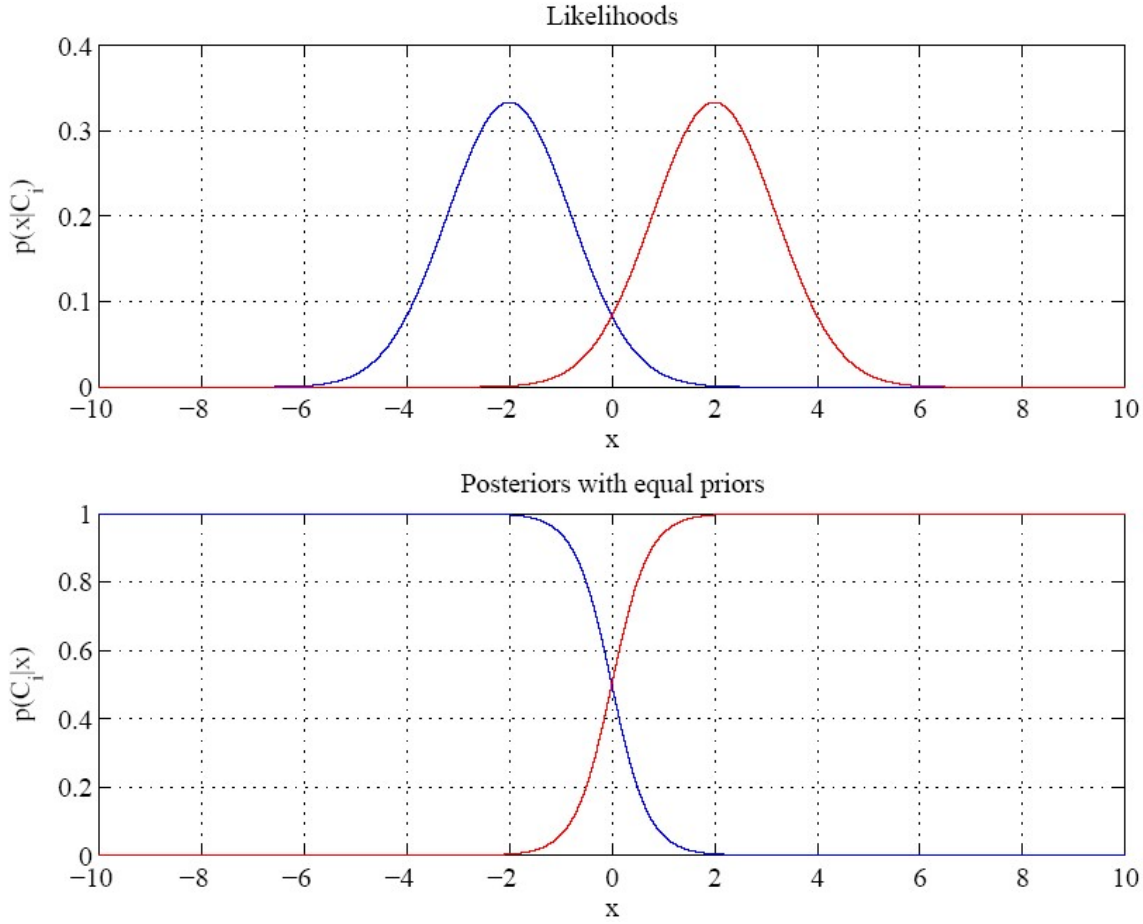


discriminant:
$P(C_1|\boldsymbol{x}) = 0.5$



The sigmoid

- the sigmoid appears in neural networks
  - it is the true posterior for Gaussian problems where the covariances are the same

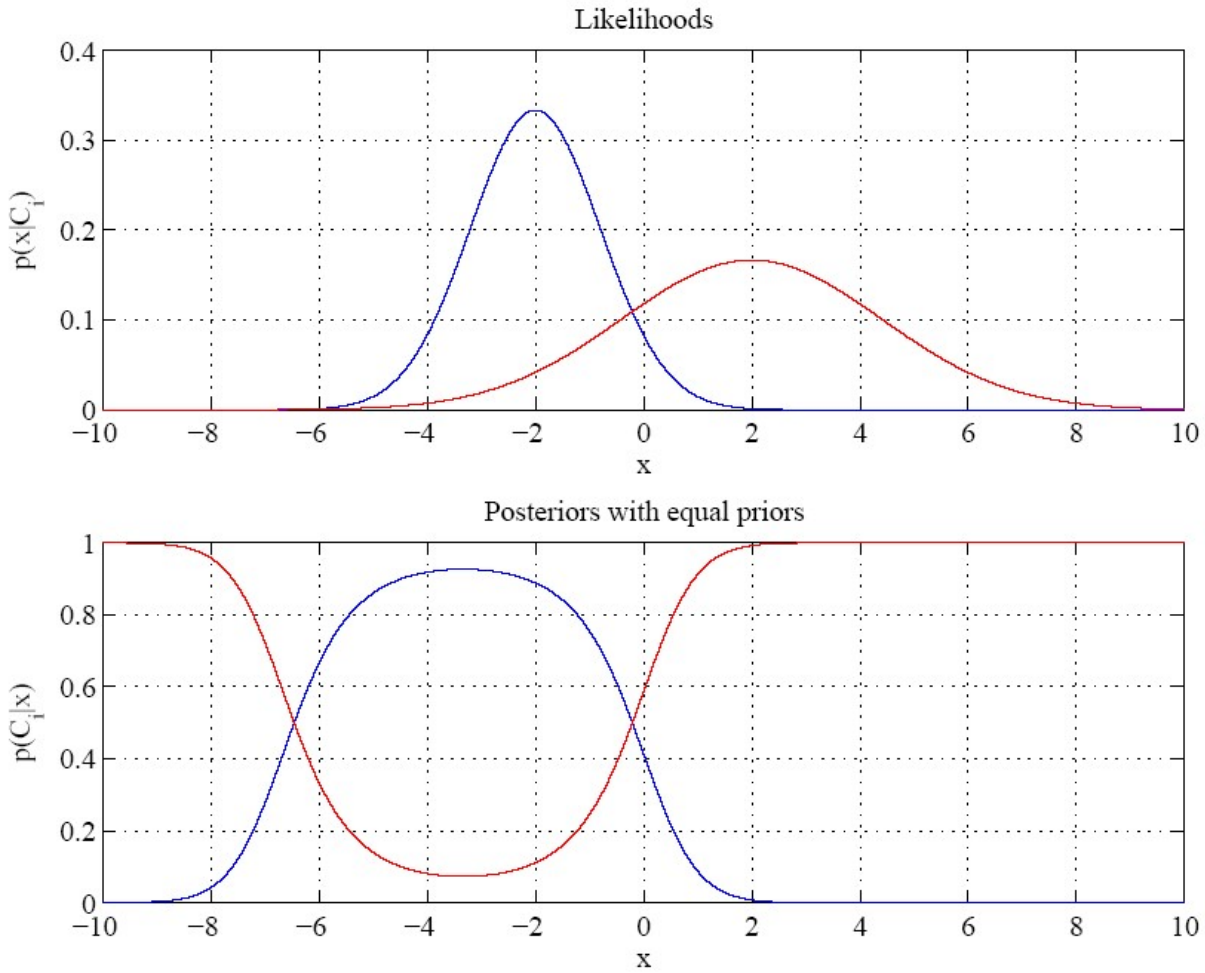


Equal variances

Single boundary at halfway between means

The sigmoid

- but not necessarily when the covariances are different



Variances are different

Two boundaries

The (real) Gaussian classifier

- instead of the BDR

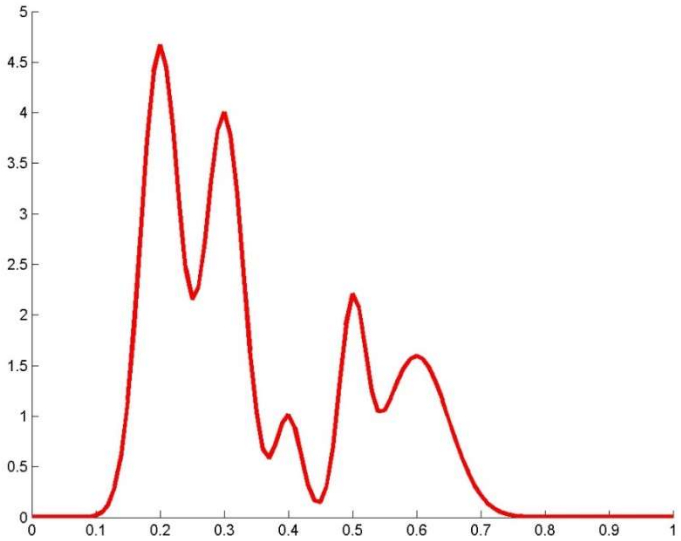
$$i^*(x) = \arg\max_i \left[-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i) - \frac{1}{2}\log(2\pi)^d |\Sigma_i| + \log P_Y(i) \right]$$

- we use the estimate

$$i^*(x) = \arg\max_i \left[-\frac{1}{2}(x-\hat{\mu}_i)^T \hat{\Sigma}_i^{-1}(x-\hat{\mu}_i) - \frac{1}{2}\log(2\pi)^d |\hat{\Sigma}_i| + \log \hat{P}_Y(i) \right]$$

Important

- warning: at this point all optimality claims for the BDR cease to be valid!!
- the BDR is guaranteed to achieve the minimum loss when we use the true probabilities
- when we "plug in" the probability estimates, we could be implementing a classifier that is quite distant from the optimal



  - e.g. if the $P_{X|Y}(x|i)$ look like the example above
  - I could never approximate them well by a parametric models like a MV Gaussian

Maximum likelihood

- this seems pretty serious
  - how should I get these probabilities then?
- we rely on the maximum likelihood (ML) principle
- this has three steps:
  - 1) we choose a parametric model for all probabilities
  - to make this clear we denote the vector of parameters by Θ and the class-conditional distributions by

$$P_{X|Y}(x \mid i; \Theta)$$

  - note that this means that Θ is NOT a random variable (otherwise it would have to show up as subscript)
  - it is simply a parameter, and the probabilities are a function of this parameter

Maximum likelihood

- three steps:
  - 2) we assemble a collection of datasets $\mathcal{D}^{(i)} = \{x_1^{(i)}, ..., x_n^{(i)}\}$ set of examples from class *i*
  - 3) we select the parameters of class *i* to be the ones that maximize the probability of the data from that class

$$\Theta_i = \arg\max_{\Theta} P_{X|Y}\left(D^{(i)} \mid i;\Theta\right)$$
$$= \arg\max_{\Theta} \log P_{X|Y}\left(D^{(i)} \mid i;\Theta\right)$$

  - like before, it does not really make any difference to maximize probabilities or their *log*

Maximum likelihood

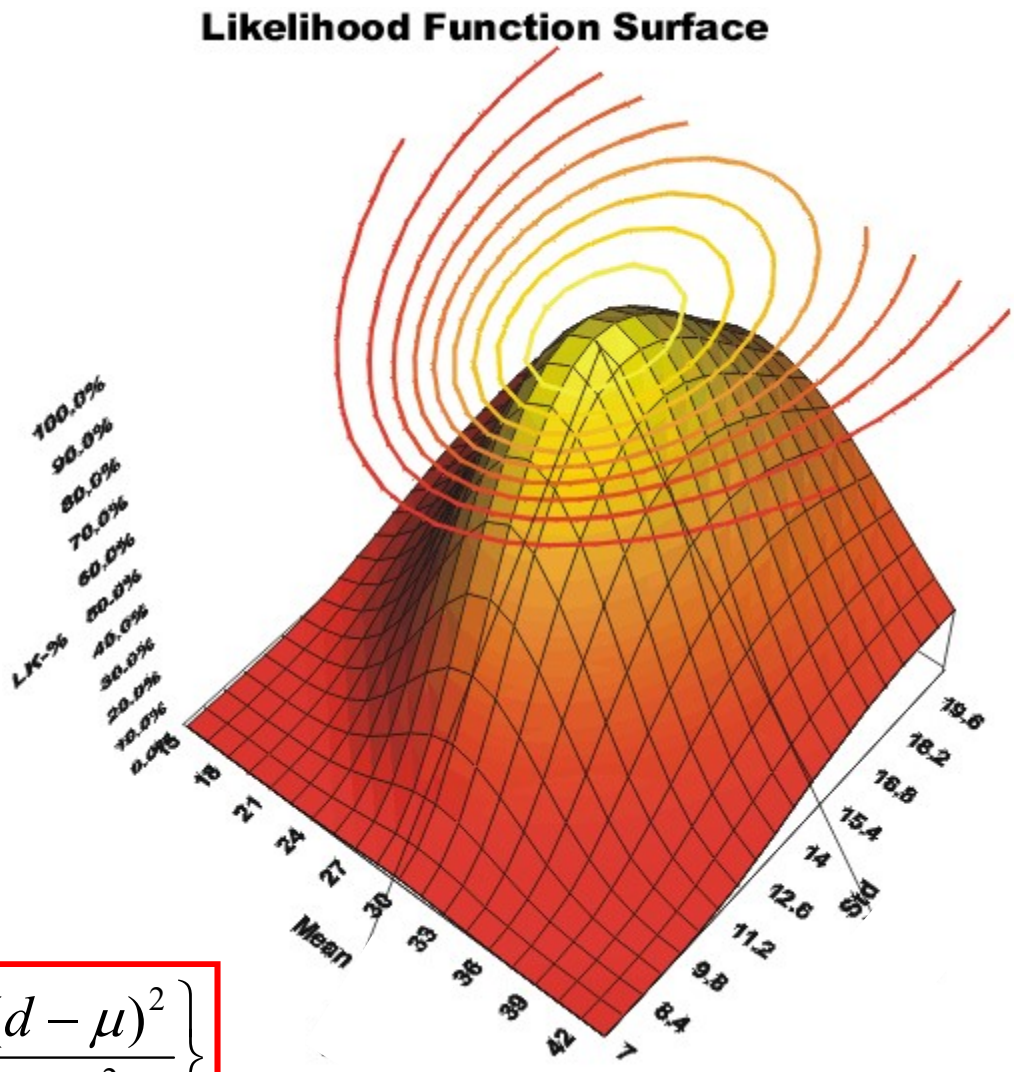
- since
  - each sample $\mathcal{D}^{(i)}$ is considered independently
  - parameter Θ_i estimated only from sample $\mathcal{D}^{(i)}$
- we simply have to repeat the procedure for all classes
- so, from now on we omit the class variable

$$\Theta^* = \arg\max_{\Theta} P_X(D;\Theta)$$
$$= \arg\max_{\Theta} \log P_X(D;\Theta)$$

- the function $P_X(\mathcal{D};\Theta)$ is called the likelihood of the parameter Θ with respect to the data
- or simply the likelihood function

Maximum likelihood

- note that the likelihood function is a function of the parameters Θ
- it does not have the same shape as the density itself
- e.g. the likelihood function of a Gaussian is not bell-shaped
- the likelihood is defined only after we have a sample



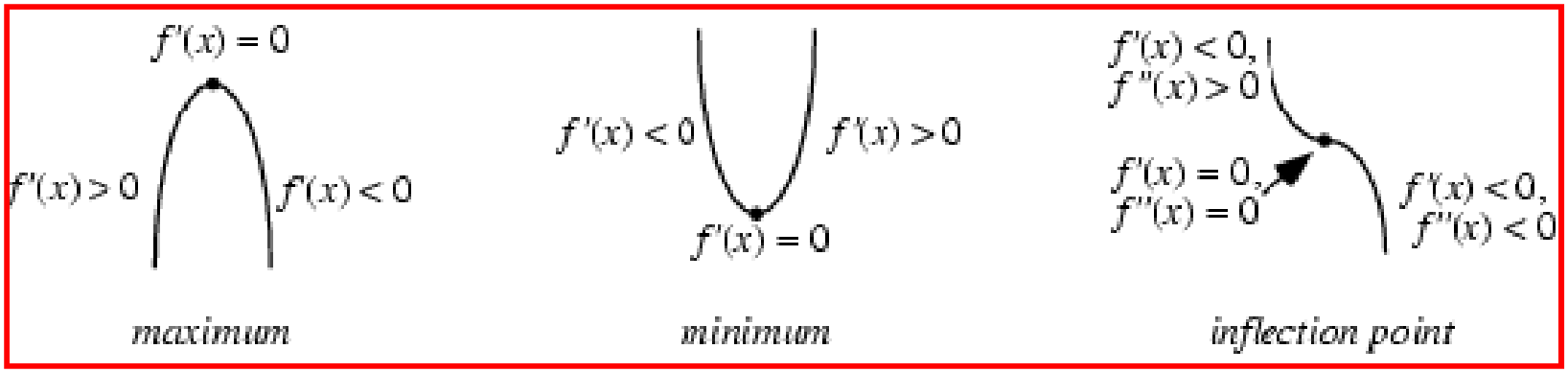
$$P_X(d;\Theta) = \frac{1}{\sqrt{(2\pi)\sigma^2}} \exp\left\{-\frac{(d-\mu)^2}{2\sigma^2}\right\}$$

Maximum likelihood

- given a sample, to obtain ML estimate we need to solve

$$\Theta^* = \arg\max_{\Theta} P_X(D;\Theta)$$

- when Θ is a scalar this is high-school calculus



- we have a maximum when
  - first derivative is zero
  - second derivative is negative

Maximum likelihood

- let's consider the Gaussian example

$$f(T) = \frac{1}{\sigma_T \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{T-\bar{T}}{\sigma_T}\right)^2}$$

- given a sample $\{T_1, \ldots, T_N\}$ of independent points
- the likelihood is

$$L(T_1, T_2, \ldots, T_N | \bar{T}, \sigma_T) = L = \prod_{i=1}^{N} \left[\frac{1}{\sigma_T \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{T_i-\bar{T}}{\sigma_T}\right)^2} \right]$$

$$L = \frac{1}{(\sigma_T \sqrt{2\pi})^N} e^{-\frac{1}{2}\sum_{i=1}^{N}\left(\frac{T_i-\bar{T}}{\sigma_T}\right)^2}$$

Maximum likelihood

- and the log-likelihood is

$$\Lambda = \ln L = -\frac{N}{2}\ln(2\pi) - N\ln\sigma_T - \frac{1}{2}\sum_{i=1}^{N}\left(\frac{T_i - \bar{T}}{\sigma_T}\right)^2$$

- the derivative with respect to the mean is zero when

$$\frac{\partial(\Lambda)}{\partial\bar{T}} = \frac{1}{\sigma_T^2}\sum_{i=1}^{N}(T_i - \bar{T}) = 0$$

- or

$$\bar{T} = \frac{1}{N}\sum_{i=1}^{N}T_i$$

- note that this is just the sample mean

Maximum likelihood

- and the log-likelihood is

$$\Lambda = \ln L = -\frac{N}{2}\ln(2\pi) - N\ln\sigma_T - \frac{1}{2}\sum_{i=1}^{N}\left(\frac{T_i - \bar{T}}{\sigma_T}\right)^2$$

- the derivative with respect to the variance is zero when

$$\frac{\partial(\Lambda)}{\partial\sigma_T} = -\frac{N}{\sigma_T} + \frac{1}{\sigma_T^3}\sum_{i=1}^{N}(T_i - \bar{T})^2 = 0$$

- or

$$\hat{\sigma}_T^2 = \frac{1}{N}\sum_{i=1}^{N}(T_i - \bar{T})^2$$

- note that this is just the sample variance

Maximum likelihood

- example:
  - if sample is {10,20,30,40,50}

$$\bar{T} = \frac{1}{N}\sum_{i=1}^{N} T_i$$

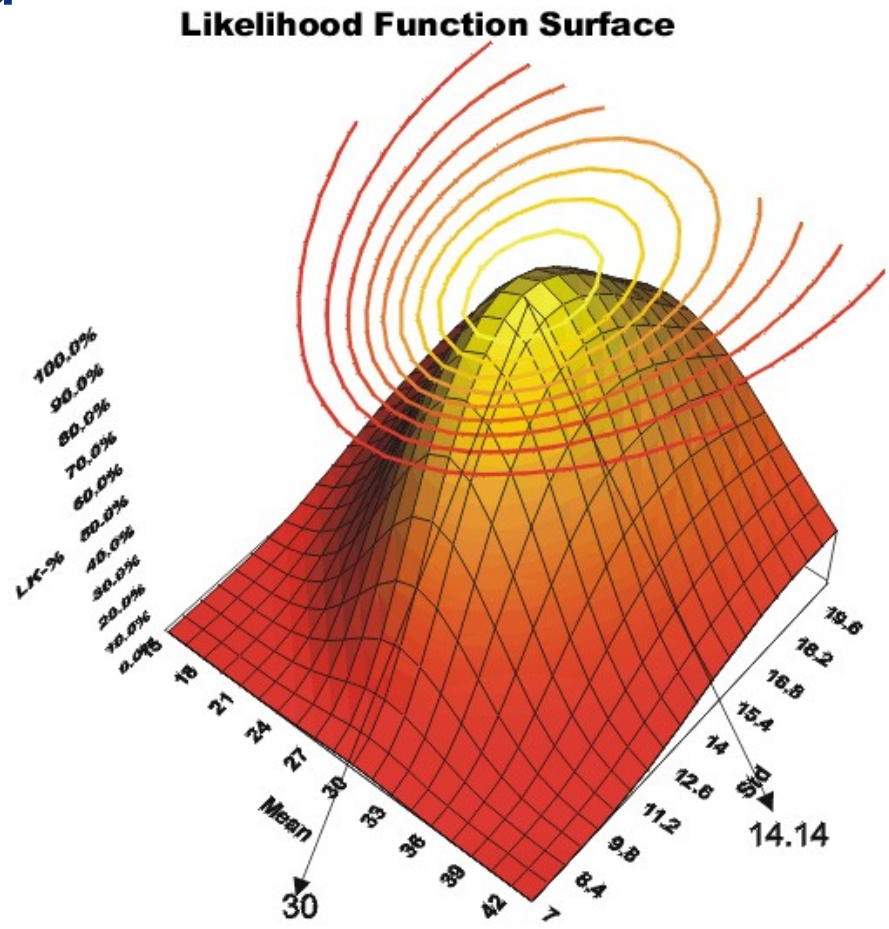
$$= \frac{10+20+30+40+50}{5}$$

$$= 30$$

$$\hat{\sigma}_T = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(T_i - \bar{T})^2}$$

$$= \sqrt{\frac{(10-30)^2+(20-30)^2+(30-30)^2+(40-30)^2+(50-30)^2}{5}}$$

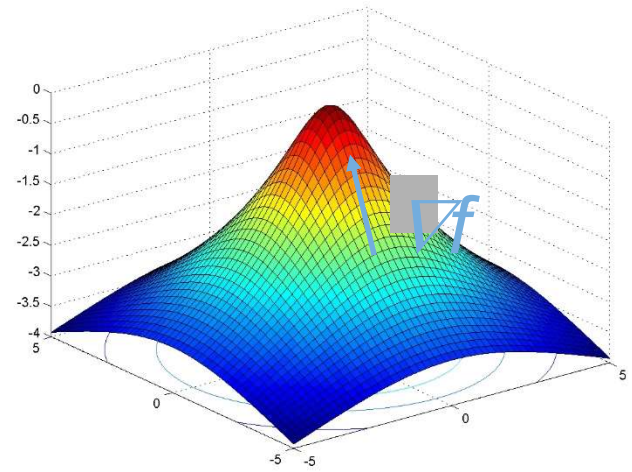
$$= 14.1421$$



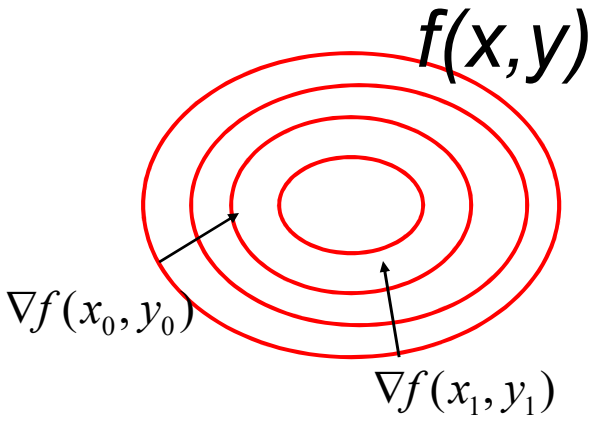
The gradient

- in higher dimensions, the generalization of the derivative is the gradient
- the gradient of a function *f(w)* at *z* is

$$\nabla f(z) = \left(\frac{\partial f}{\partial w_0}(z), \cdots, \frac{\partial f}{\partial w_{n-1}}(z) \right)^T$$

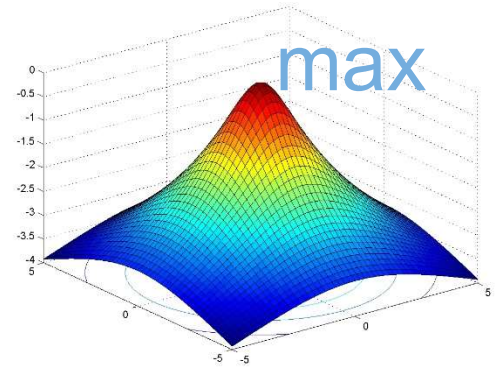


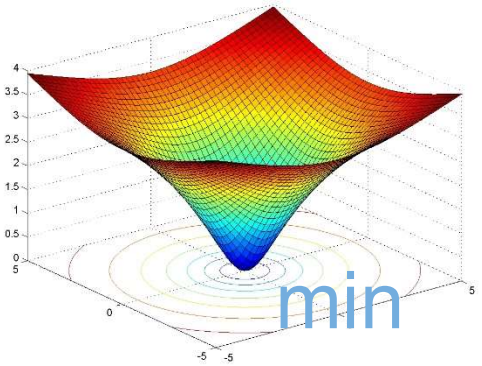
- the gradient has a nice geometric interpretation
  - it points in the direction of maximum growth of the function
  - which makes it perpendicular to the contours where the function is constant

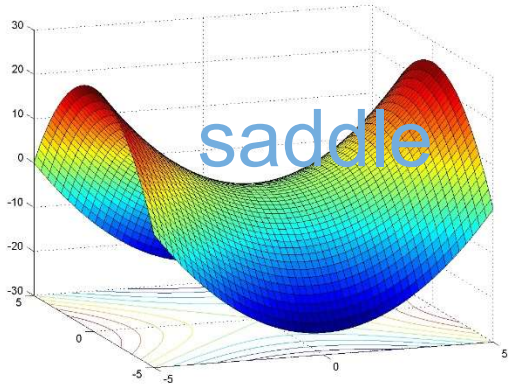


The gradient

- note that if $\nabla f = 0$
  - there is no direction of growth
  - also - $\nabla f = 0$, and there is no direction of decrease
  - we are either at a local minimum or maximum or “saddle” point
- conversely, at local min or max or saddle point
  - no direction of growth or decrease
  - $\nabla f = 0$
- this shows that we have a critical point if and only if $\nabla f = 0$
- to determine which type we need second order conditions







The Hessian

- the extension of the second-order derivative is the Hessian matrix

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_0^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_0 \partial x_{n-1}}(x) \\ \vdots & & \\ \frac{\partial^2 f}{\partial x_{n-1} \partial x_0}(x) & \cdots & \frac{\partial^2 f}{\partial x_{n-1}^2}(x) \end{bmatrix}$$

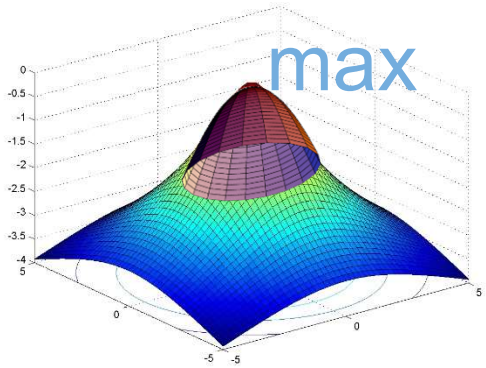
  - at each point x, gives us the quadratic function

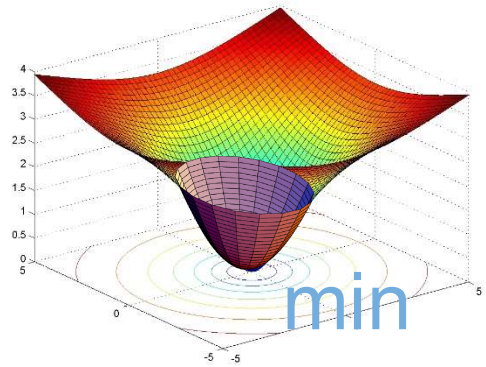
$$x^t \nabla^2 f(x) x$$

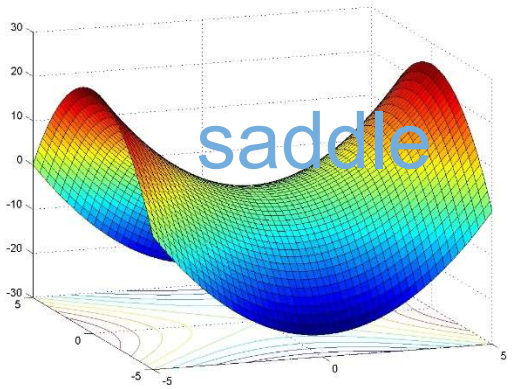
    that best approximates f(x)

The Hessian

- this means that, when gradient is zero at x, we have
  - a maximum when function can be approximated by an "upwards-facing" quadratic
  - a minimum when function can be approximated by a "downwards-facing" quadratic
  - a saddle point otherwise





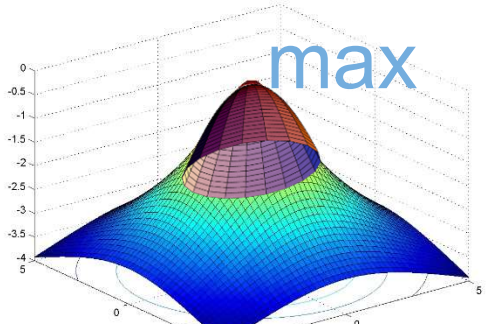


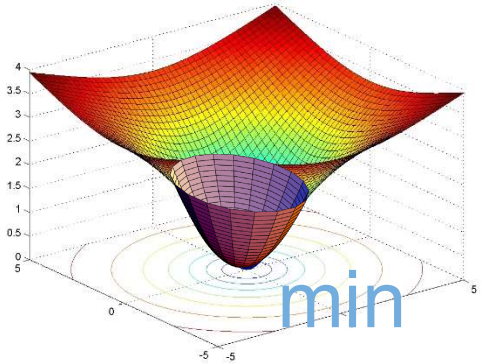
The Hessian

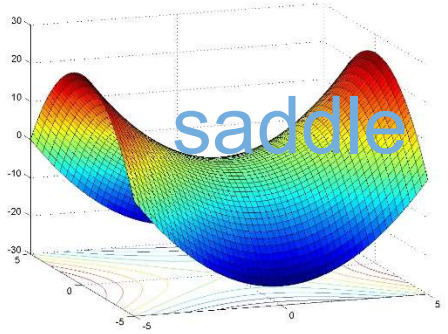
- this is something that we already saw
- for any matrix M, the function

$$x^t M x$$

- is
  - upwards facing quadratic when M is negative definite
  - downwards facing quadratic when M is positive definite
  - saddle otherwise
- hence, all that matters is the positive definiteness of the Hessian
- we have a maximum when Hessian is negative definite







Optimality conditions

- in summary
- w^* is a local minimum of $f(w)$ if and only if
  - f has zero gradient at w^*

$$\nabla f(w^*) = 0$$

  - and the Hessian of f at w^* is positive definite

$$d^t \nabla^2 f(w^*) d \geq 0, \quad \forall d \in \Re^n$$

  - where

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_0^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_0 \partial x_{n-1}}(x) \\ \vdots & & \\ \frac{\partial^2 f}{\partial x_{n-1} \partial x_0}(x) & \cdots & \frac{\partial^2 f}{\partial x_{n-1}^2}(x) \end{bmatrix}$$

Maximum likelihood

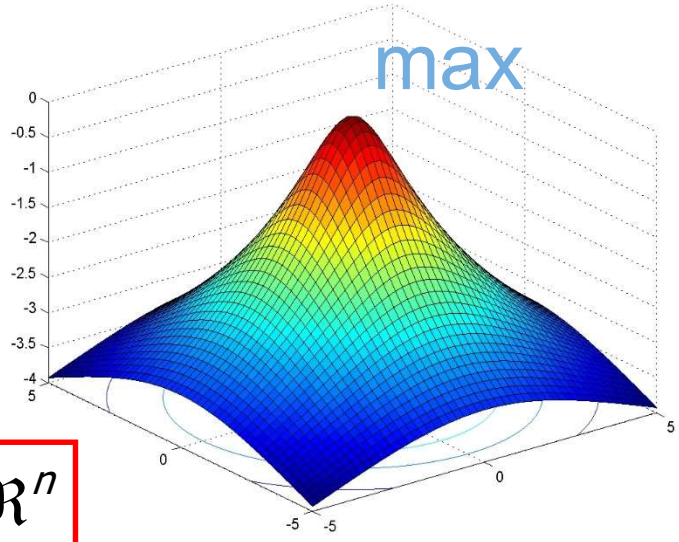
- given a sample, to obtain ML estimate we need to solve

$$\Theta^* = \arg\max_{\Theta} P_X(D;\Theta)$$

- the solutions are the parameters such that

$$\nabla_{\Theta} P_X(x;\Theta) = 0$$

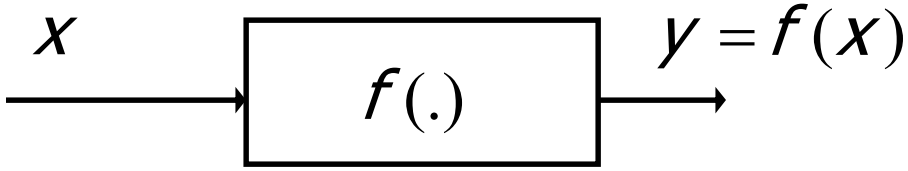
$$\theta^t \nabla_{\Theta}^{\,2} P_X(x;\theta)\theta \geq 0, \quad \forall \theta \in \Re^n$$



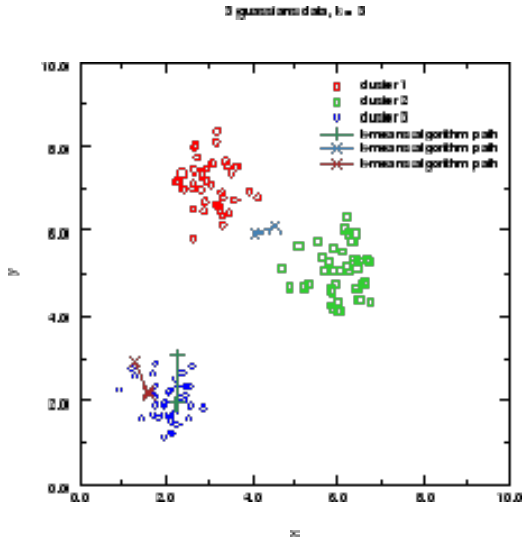
- note that you always have to check the second-order condition

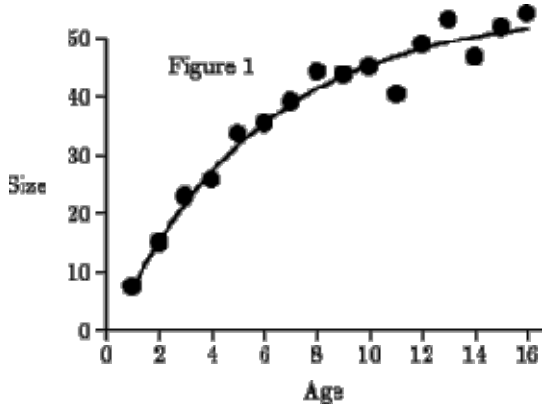
Statistical learning

- goal: given a function



and a collection of example data-points, learn what the function *f(.)* is.

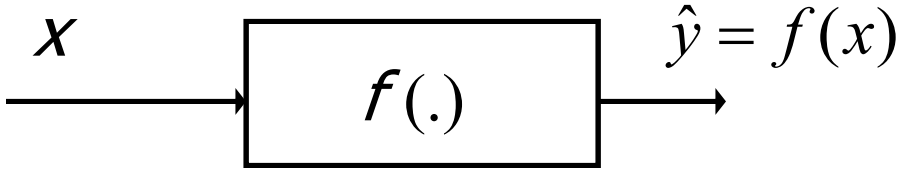




- two major types of problems:
  - Y in {0, ..., M-1} referred to as classification
  - Y real referred to as regression
- we have dealt mostly with classification

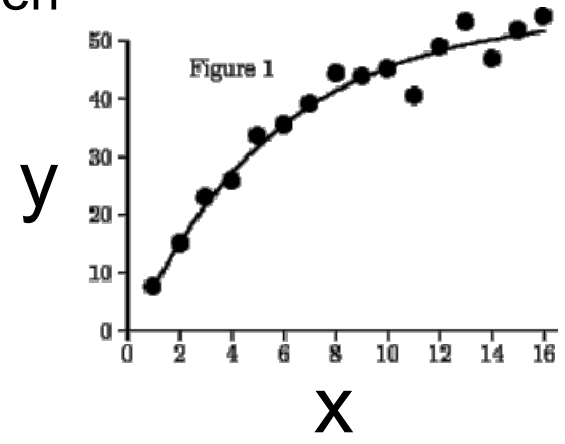
Optimal regression

- note that this can be framed in our initial formulation of optimizing the loss of the learning system



$$L(y, \hat{y})$$

- for a regression problem this still applies
  - the interpretation of f(.) as a predictor even becomes more intuitive
- function f(.) is parameterized by some vector Θ
- regression consists of finding the parameters that minimize loss



- two types of regression problems: linear vs non-linear

Examples

- linear regression:
  - line fitting

$$f(x;\Theta) = \theta_1 x + \theta_0$$

  - polynomial fitting

$$f(x;\Theta) = \sum_{i=0}^{K} \theta_i x^i$$

  - truncated Fourier series

$$f(x;\Theta) = \sum_{i=0}^{K} \theta_i \sin(ix)$$

- Non-linear regression:
  - neural networks

$$f(x;\Theta) = \frac{1}{1+e^{-\theta_1 x - \theta_0}}$$

  - sinusoidal decompositions

$$f(x;\Theta) = \sum_{i=0}^{K} \sin(\theta_i x)$$

  - …

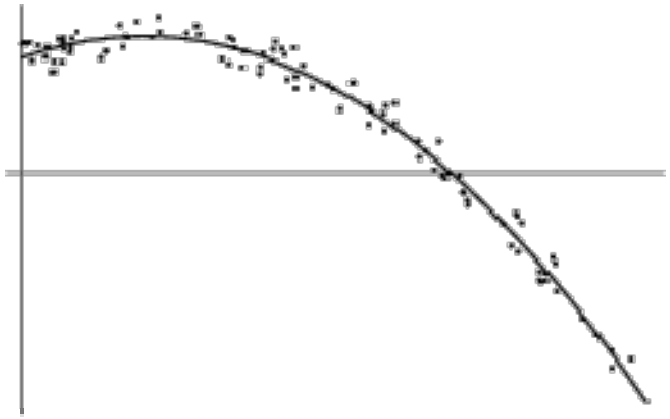
Least squares

- we considered linear problems
  - recall: what matters is linearity in the parameters
- general solution:
  - write the model as

$$f(x;\Theta) = \gamma(x)^T \Theta$$

  - e.g., for a line,

$$f(x;\Theta) = \theta_1 x + \theta_0$$



$$\gamma(x) = \begin{bmatrix} 1 \\ x \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

- this can be generalized to any model if we make

$$\gamma(x) = \begin{bmatrix} \gamma_0(x) \\ \vdots \\ \gamma_k(x) \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_k \end{bmatrix}$$

Examples

- note that $\gamma(x)$ can be arbitrary non-linear functions of x
  - line fitting

$$f(x;\Theta) = \theta_1 x + \theta_0 \qquad \gamma(x)^T = [1 \quad x]$$

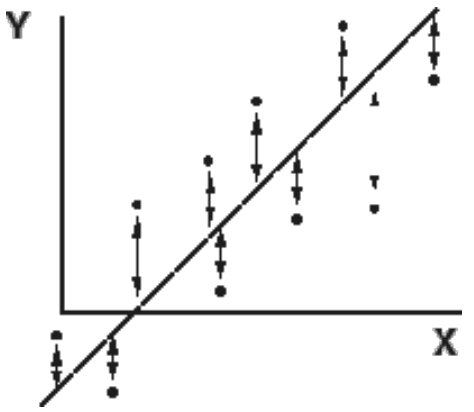
  - polynomial fitting

$$f(x;\Theta) = \sum_{i=0}^{K} \theta_i x^i \qquad \gamma(x)^T = [1 \quad \ldots \quad x^K]$$

  - truncated Fourier series

$$f(x;\Theta) = \sum_{i=0}^{K} \theta_i \sin(ix) \qquad \gamma(x)^T = [0 \quad \ldots \quad \sin(Kx)]$$

Least squares



- the loss is the Euclidean norm

$$L = \left\| y - \Gamma(x)\Theta \right\|^2$$

- where

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad \Gamma(x) = \begin{bmatrix} \gamma(x_1)^T \\ \vdots \\ \gamma(x_n)^T \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_k \end{bmatrix}$$

- can also be written as,

$$L = \sum_i \left(y_i - \gamma(x_i)^T \Theta \right)^2$$

- e.g., for the line, $L = \sum_i \left(y_i - \theta_1 x_i - \theta_0 \right)^2$

Examples

- the most important component is the matrix $\Gamma(x)$
  - line fitting

$$f(x;\Theta) = \theta_1 x + \theta_0$$

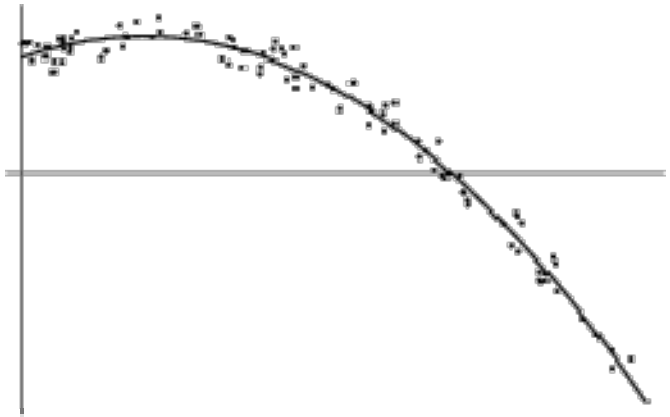
$$\Gamma(x) = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

polynomial fitting

$$f(x;\Theta) = \sum_{i=0}^{K} \theta_i x^i$$

$$\Gamma(x) = \begin{bmatrix} 1 & \dots & x_1^K \\ & \vdots & \\ 1 & \dots & x_n^K \end{bmatrix}$$

Least squares



- to minimize

$$L = \left\| y - \Gamma(x)\Theta \right\|^2$$

  - we simply have to find x such that

$$\nabla_\Theta L = -2\Gamma(x)^T \left[y - \Gamma(x)\Theta \right] = 0$$

$$\nabla_\Theta^2 L = 2\Gamma(x)^T \Gamma(x) \succ 0$$

  - these hold when

$$\Theta^* = \left[\Gamma(x)^T \Gamma(x) \right]^{-1} \Gamma(x)^T y$$

- least squares solution consists of multiplying by the pseudo-inverse of $\Gamma(x)$

$$\Gamma(x)^\Pi = \left[\Gamma(x)^T \Gamma(x) \right]^{-1} \Gamma(x)^T$$

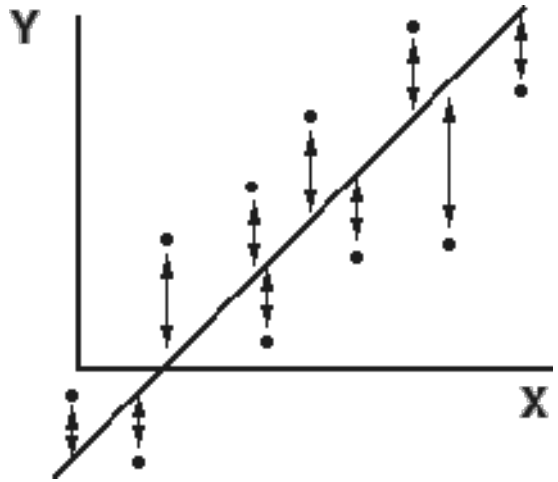
Least squares

- here is a way of thinking about this
  - we have a system of equations

$$y = \Gamma(x)\Theta$$

  - this cannot be solved because $\Gamma(x)$ is not invertible
  - e.g. for the line

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1\, x_1 \\ \vdots\ \vdots \\ 1\, x_n \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$



  - we multiply both sides by $\Gamma(x)^T$

$$\Gamma(x)^T y = \Gamma(x)^T \Gamma(x)\Theta$$

Least squares

- this is now a solvable system

$$\begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

- whose solution is given by the pseudo-inverse

$$\Theta^* = \left[\Gamma(x)^T \Gamma(x)\right]^{-1} \Gamma(x)^T y$$

- and we have just seen that this is the best solution for the original problem in the least squares sense

$$\Theta^* = \arg\min_{\Theta} \left\| y - \Gamma(x)\Theta \right\|^2$$

Geometric interpretation

- there is also a nice geometric way to derive the least squares solution
- we want to minimize

$$L = \|y - \Gamma(x)\Theta\|^2$$

- given the known matrix

$$\Gamma(x) = \begin{bmatrix} | & & | \\ \Gamma_1 & \dots & \Gamma_K \\ | & & | \end{bmatrix}$$

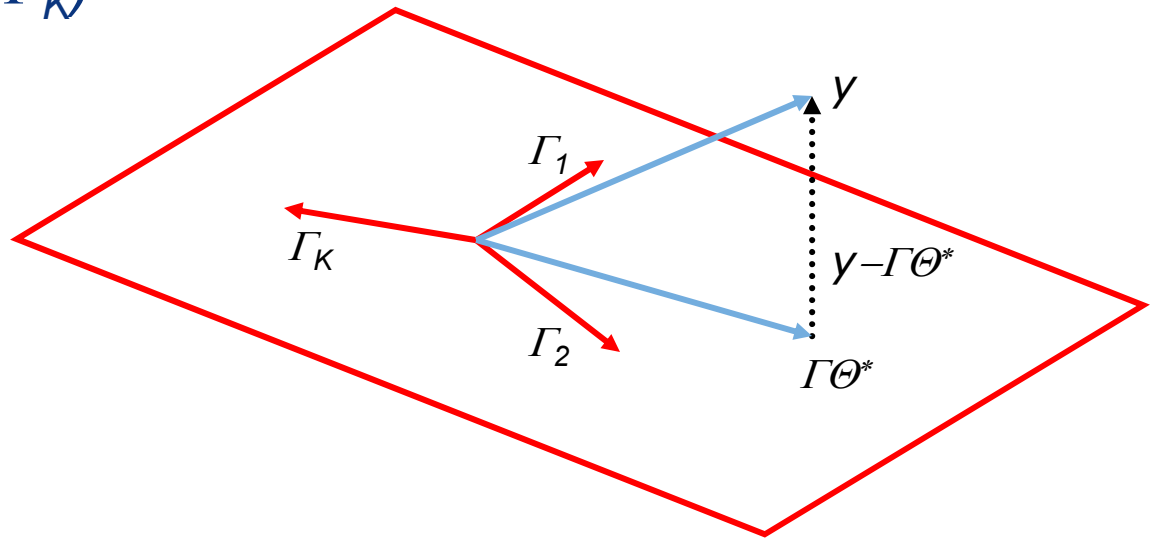
- the vector

$$\Gamma(x)\Theta = \begin{bmatrix} | & & | \\ \Gamma_1 & \dots & \Gamma_K \\ | & & | \end{bmatrix} \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_K \end{bmatrix} = \begin{bmatrix} | \\ \sum_i \theta_i \Gamma_i \\ | \end{bmatrix}$$

is a linear combination of the column vectors Γ_i

Geometric interpretation

- this means that $\Gamma\Theta$ is a vector in the column space of $(\Gamma_1, \ldots, \Gamma_K)$

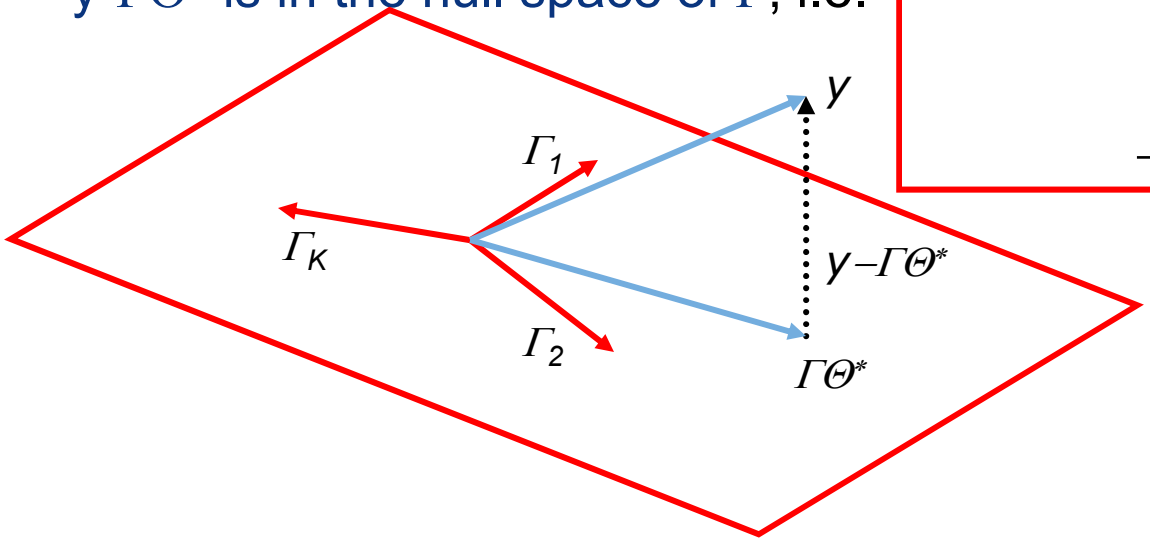


- assume that y is as shown
  - what is the value of $\Gamma\Theta$ closest to y?
  - it has to be the projection of y on the hyper-plane

Geometric interpretation

- let's denote this by $\Gamma\Theta^*$. Then,
  - y-$\Gamma\Theta^*$ is in the null space of Γ, i.e.

$$\begin{aligned}\left(y-\Gamma\Theta^*\right) &\perp \Gamma_1\\ &\perp \Gamma_2\\ &\vdots\\ &\perp \Gamma_K\end{aligned}$$



- or

$$\begin{cases}\Gamma_1^T\left(y-\Gamma\Theta^*\right)=0\\ \vdots\\ \Gamma_1^T\left(y-\Gamma\Theta^*\right)=0\end{cases}\Leftrightarrow\begin{cases}\begin{bmatrix}- & \Gamma_1^T & -\\ & \vdots & \\ - & \Gamma_1^T & -\end{bmatrix}\left(y-\Gamma\Theta^*\right)=0\end{cases}\Leftrightarrow\Gamma^T\left(y-\Gamma\Theta^*\right)=0$$

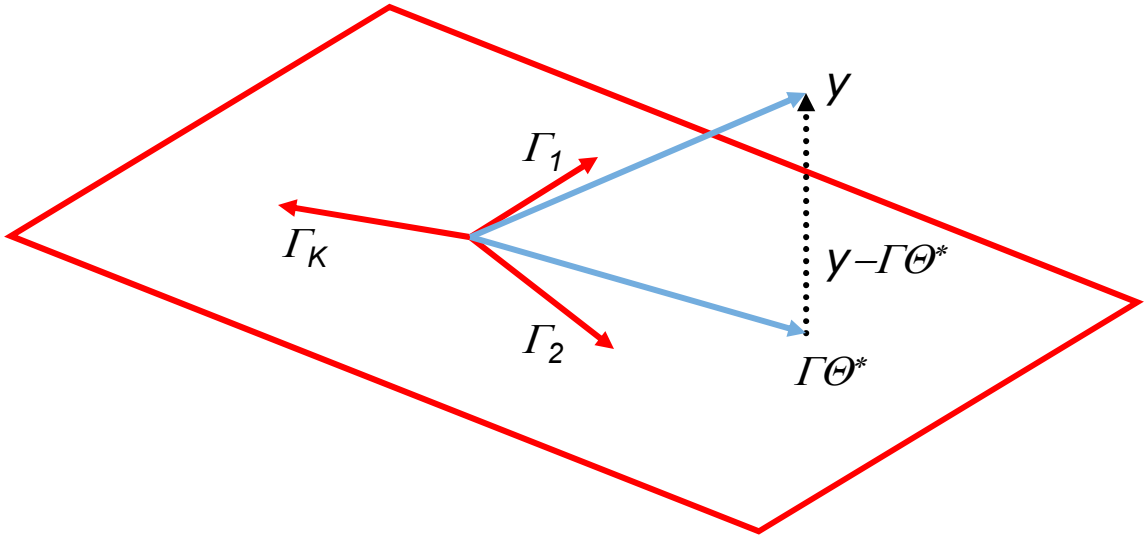
Geometric interpretation

- from which

$$\Gamma^T\left(y-\Gamma\Theta^*\right)=0 \Leftrightarrow \Gamma^T y=\Gamma^T\Gamma\Theta^*$$

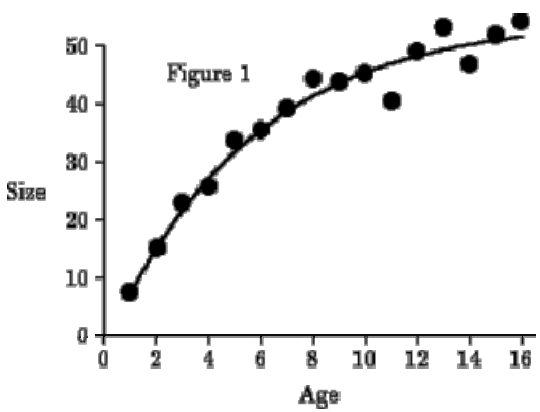
and we get our well known equation

$$\Theta^*=\left[\Gamma(x)^T\Gamma(x)\right]^{-1}\Gamma(x)^T y$$



Regression

- This is the usual definition of a regression problem
  - two random variables X and Y
  - a dataset of examples $\mathcal{D} = \{(x_1,y_1), \ldots (x_n,y_n)\}$
  - a parametric model of the form



$$y = f(x;\Theta) + \varepsilon$$

  - where Θ is a parameter vector, and ε a random variable that accounts for noise

- the pdf of the noise determines the loss in the other formulation

Regression

- error
  - Gaussian

$$P_\varepsilon(\varepsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\varepsilon^2}{2\sigma^2}}$$

  - Laplacian

$$P_\varepsilon(\varepsilon) = \frac{1}{2\sigma} e^{-\frac{|\varepsilon|}{\sigma}}$$

  - Rayleigh

$$P_\varepsilon(\varepsilon) = \frac{\varepsilon}{\sigma^2} e^{-\frac{\varepsilon^2}{2\sigma^2}}$$

- Loss
  - L_2 distance

$$L(x, y) = (y - x)^2$$

  - L_1 distance

$$L(x, y) = |y - x|$$

  - Rayleigh distance

$$L(x, y) = (y - x)^2 - \log(y - x)$$

Regression

- we know how to solve the problem with losses
  - why do we care about the added complexity of introducing probabilities?
- the main reason is that this allows a data driven definition of the loss
- one good way to see this is the problem of weighted least squares
  - suppose that you know that not all measurements (x_i,y_i) have the same importance
  - this can be encoded in the loss
  - remember that

$$L = \left\| y - \Gamma(x)\Theta \right\|^2 = \sum_i \left(y_i - \left[\Gamma(x)\Theta \right]_i \right)^2$$

Regression

- to weigh different points differently we use

$$L = \sum_i w_i \left(y_i - \left[\Gamma(x)\Theta \right]_i \right)^2$$

or even the more generic form

$$L = \left(y - \Gamma(x)\Theta \right)^T W \left(y - \Gamma(x)\Theta \right)$$

- in this case the solution is

$$\Theta^* = \left[\Gamma(x)^T W \, \Gamma(x) \right]^{-1} \Gamma(x)^T W \, y$$

- the question is "how do I set up these weights"?
- without a probabilistic model I have little guidance on this

Regression

- the probabilistic equivalent

$$\Theta^* = \arg\max_{\Theta} e^{-L[y, f(x;\Theta)]}$$

$$= \arg\max_{\Theta} \exp\left\{-\frac{1}{2}(y - \Gamma(x)\Theta)^T W (y - \Gamma(x)\Theta)\right\}$$

is the Gaussian of covariance

$$\Sigma = W^{-1}$$

- in the case where W is diagonal we have

$$L = \sum_i \frac{1}{\sigma_i^2}\left(y_i - [\Gamma(x)\Theta]_i\right)^2$$

- each point is weighted by the inverse variance

Regression

- this makes a lot of sense
  - under the probabilistic formulation the variance σ_i is the variance of the error associated with the i^{th} observation

$$y_i = f(x_i; \Theta) + \varepsilon_i$$

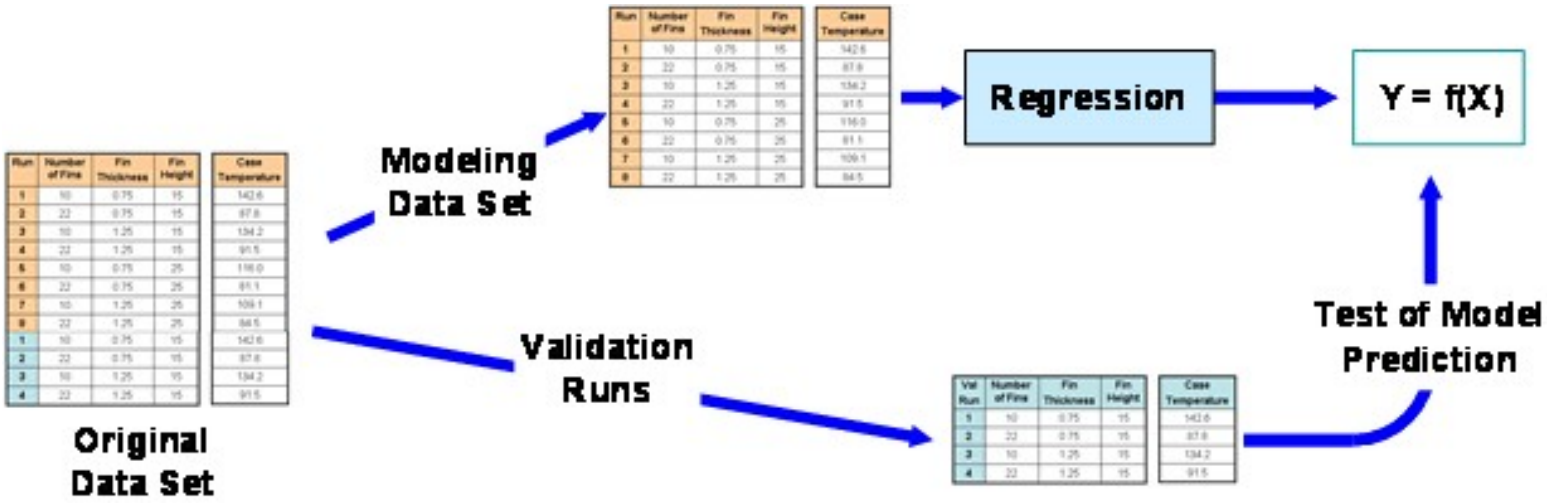
  - this means that it is a measure of the uncertainty of the observation
- when

$$W = \Sigma^{-1}$$

  we are weighting each point by the inverse of its uncertainty (variance)
- we can also check the goodness of this weighting matrix by plotting the histogram of the errors
- if *W* is right, the errors should be Gaussian

Model validation

- in fact, by analyzing the errors of the fit we can do a lot
- this is called model validation



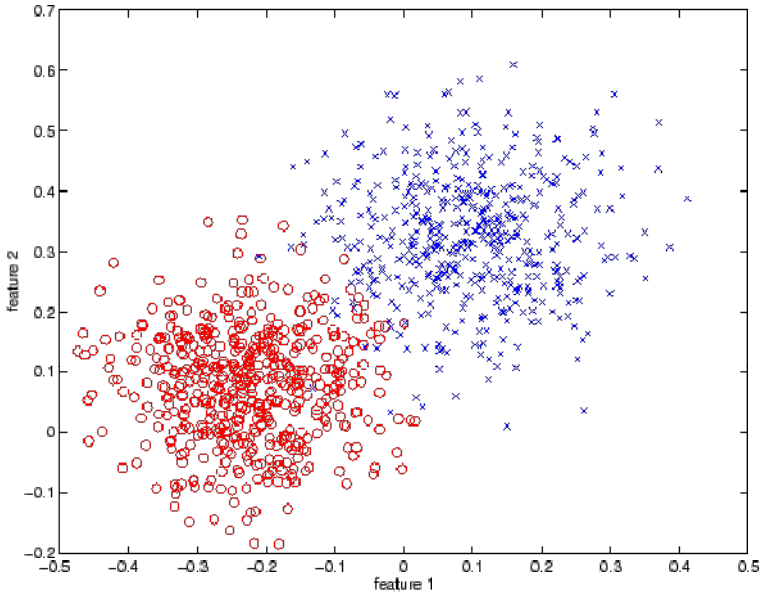
- leave some data on the side, and run it through the predictor
- analyze the errors to see if there is deviation from the assumed model (Gaussianity for least squares)

Clustering

- why need to learn without supervision?
  - in many problems labels are not available or impossible to get
  - in the digit example, someone sat in front of the computer for hours to label all those examples
  - for other problems the classes depend on the application
  - a good example is image segmentation
    - if you want to know if this is an image of the wild or of a big city, there is probably no need to segment
    - if you want to know if there is animal, then you do
    - the segmentation mask usually not available

Clustering

- in a clustering problem we do not have labels in the training set
- however, we can try to estimate these as well
- here is a strategy
  - we start with random class distributions
  - we then iterate between two steps
  - 1) apply the optimal decision rule for these distributions
    - this assigns each point to one of the clusters
  - 2) update the distributions by doing parameter estimation within each class



Clustering

- a natural question is: what model should we assume?
  - well, let's start as simple as possible
  - assume Gaussian with identity covariances and equal $P_Y(i)$
  - each class has a prototype μ_i
- the clustering algorithm becomes
  - start with some initial estimate of the μ_i (e.g. random)
  - then, iterate between
  - 1) classification rule:

$$i^*(x) = \arg\min_i \left\| x - \mu_i \right\|^2$$

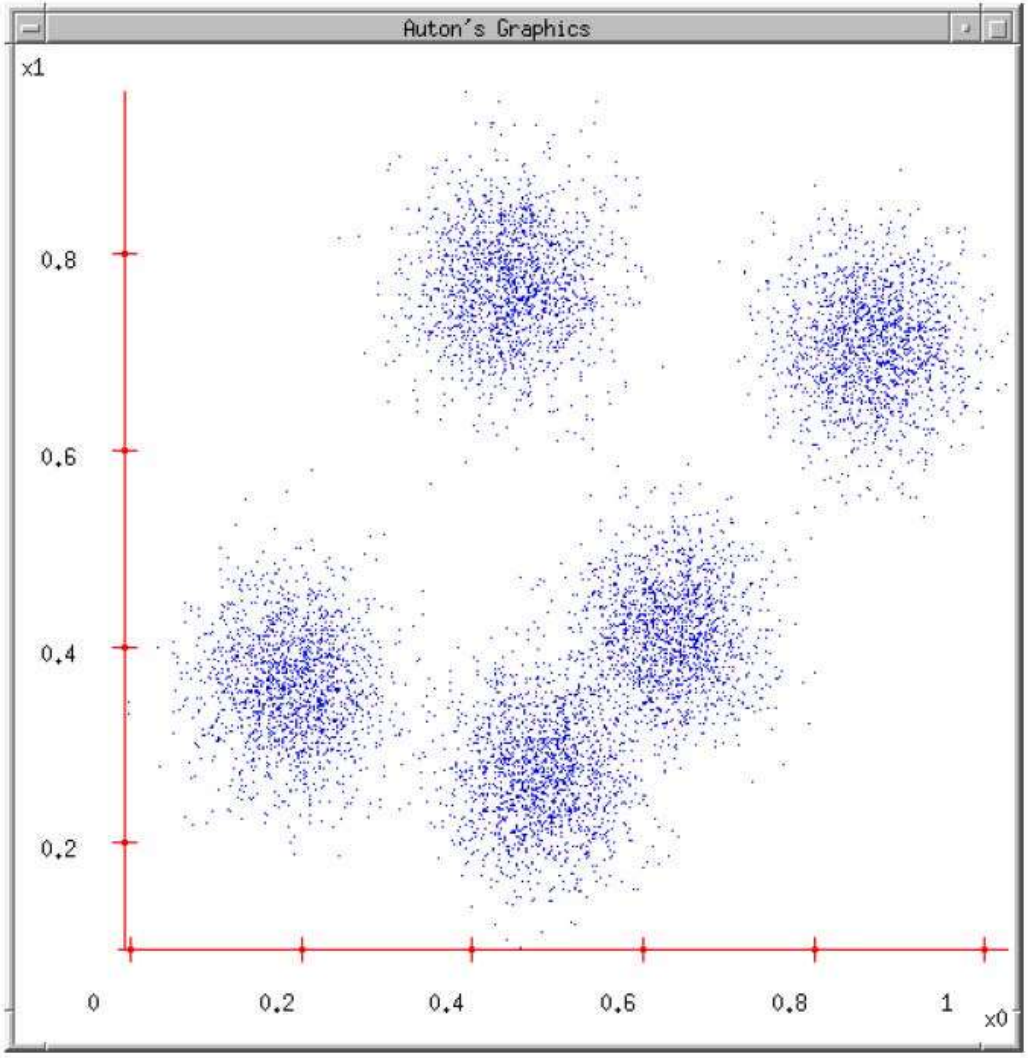
  - 2) re-estimation:

$$\mu_i^{new} = \frac{1}{n} \sum_j x_j^{(i)}$$

K-means

K-means

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

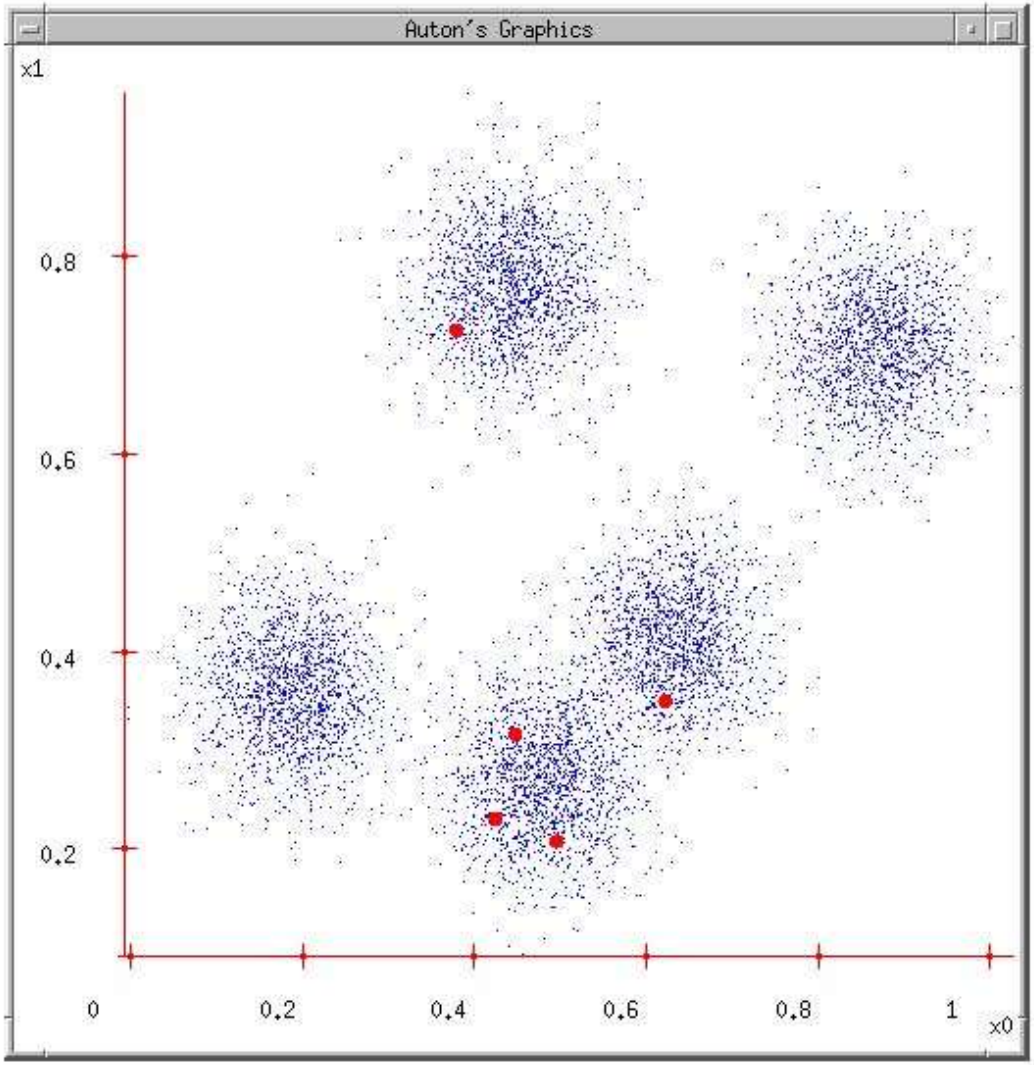


K-means and Hierarchical Clustering: Slide 6

K-means

K-means

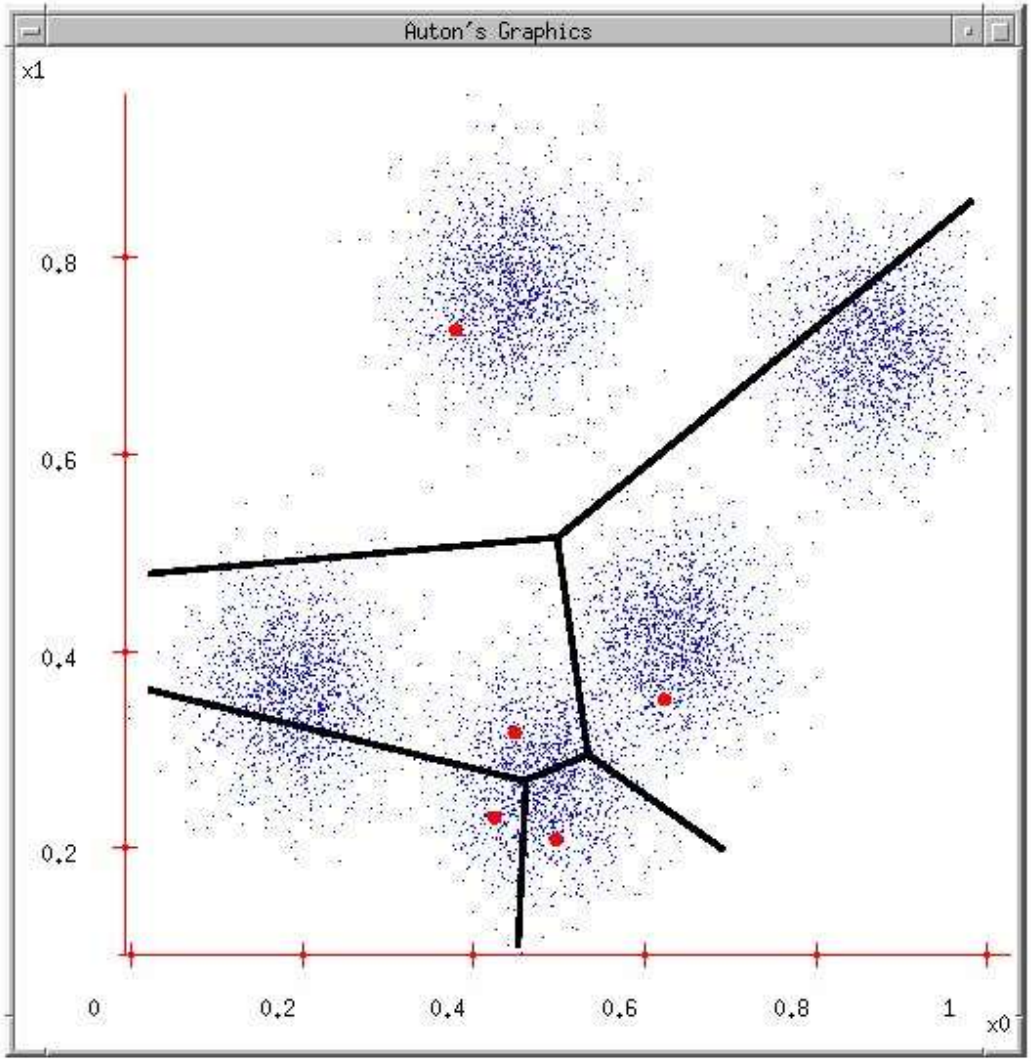
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



K-means

K-means

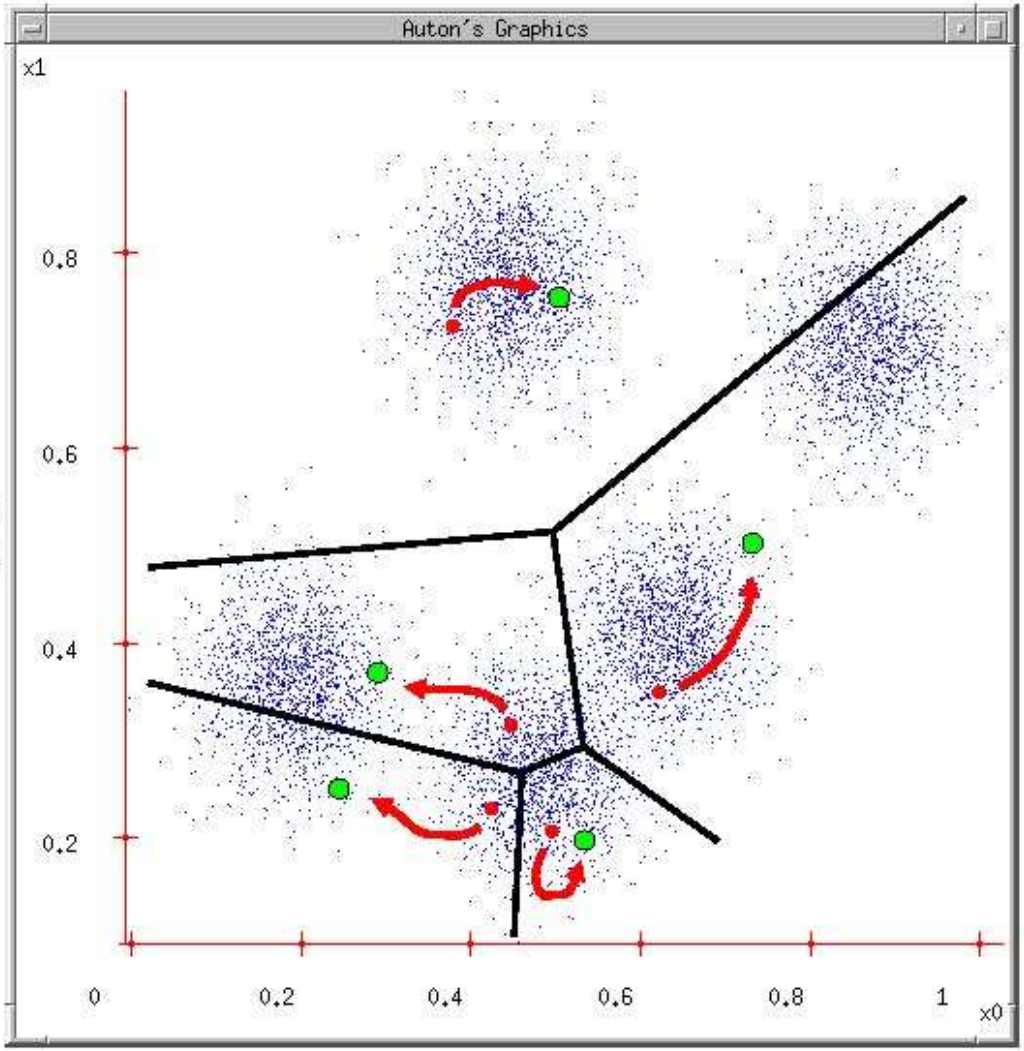
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



K-means

K-means

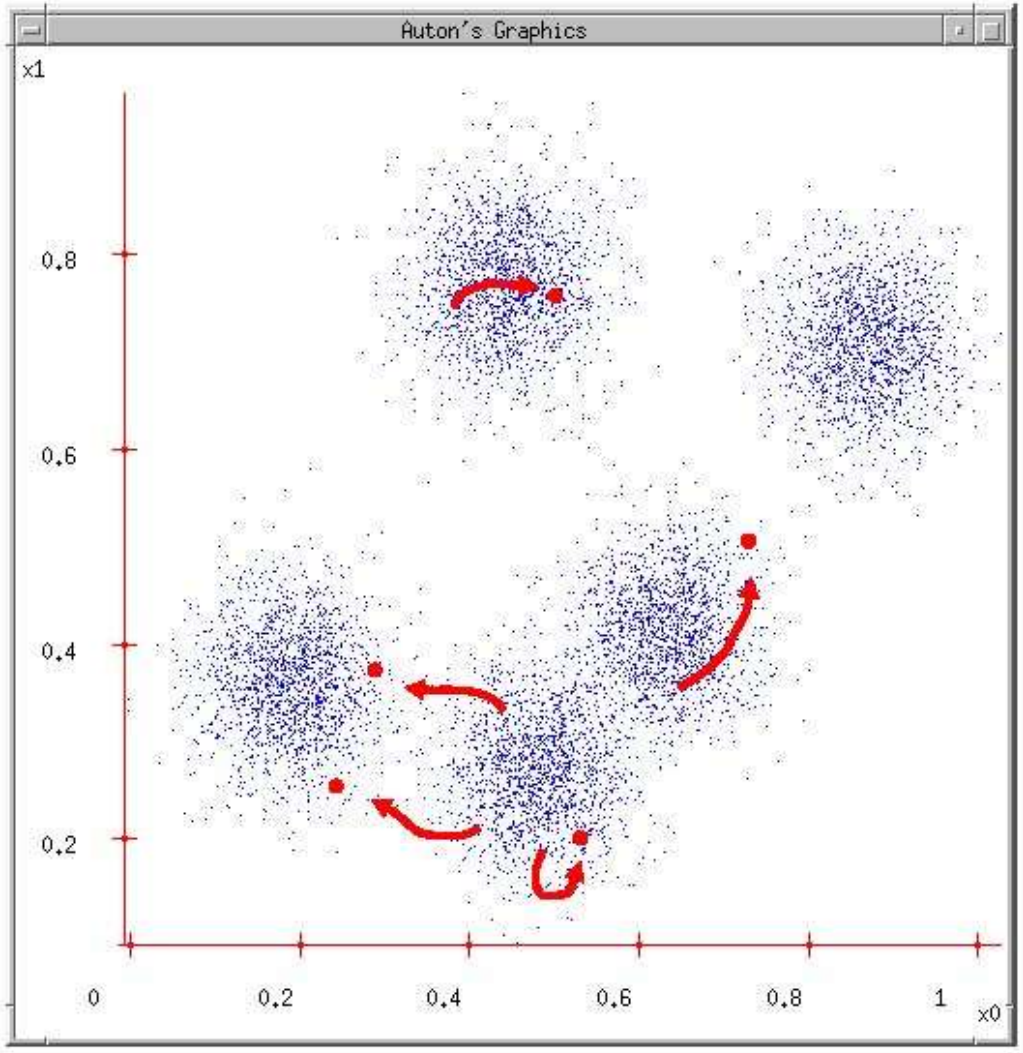
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



K-means

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!



K-means and Hierarchical Clustering: Slide 10

K means

- the name comes from the fact that we are trying to learn “k” means
- it is optimal if you want to minimize the expected value of the squared error between vector x and template to which x is assigned
- problems:
  - how many clusters?
    - various methods available, Bayesian information criterion, Akaike information criterion, minimum description length
    - guessing can work pretty well
  - local minimum only
  - how do I initialize?
    - random can be pretty bad
    - mean splitting can be significantly better

Mean splitting initialization

- for $K = 1$ we just need the mean of all points (μ^1)
- to initialize means for $K = 2$ perturb the mean randomly
  - $\mu_1^2 = \mu^1$
  - $\mu_2^2 = (1+\varepsilon)\ \mu^1$ $\qquad \varepsilon << 1$
- then run K means with $K = 2$
- initial means for $K = 4$
  - $\mu_1^4 = \mu_1^2$
  - $\mu_2^4 = (1+\varepsilon)\ \mu_1^2$
  - $\mu_3^4 = \mu_2^2$
  - $\mu_4^4 = (1+\varepsilon)\ \mu_2^2$
- then run K means with $K = 4$
- etc ….

Empty clusters

- can be a source of headaches
- at the end of each iteration of K means
  - check the number of elements in each cluster
  - if too low, throw the cluster away
  - reinitialize the mean with a perturbed version of that of the most populated cluster
- note that there are alternative names:
  - in the compression literature this is known as the generalized Loyd algorithm
  - this is actually the right name, since Loyd was the first to invent it
  - it is used in the design of vector quantizers

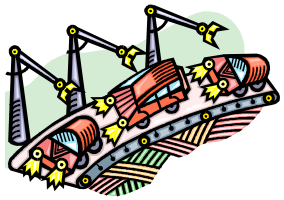
Extensions

- there are many extensions to the basic k-means algorithm
  - it turns out that one of the most important applications is supervised learning
  - remember that the optimal decision rule

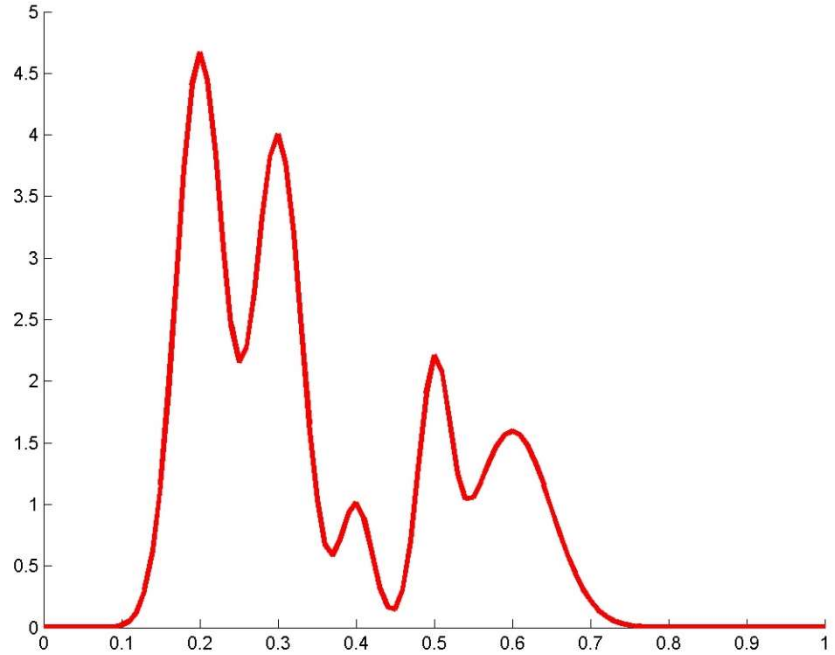
$$i^*(x) = \arg\max_i \left[\log P_{X|Y}(x \mid i) + \log P_Y(i)\right]$$

    is optimal insofar the probabilities $P_{X|Y}(x|i)$ are correctly estimated
  - this usually turns out to be impossible, by definition, when we use parametric models like the Gaussian
  - although these models are good local approximations, there are usually multiple clusters when we take a global view
  - this can be captured with recourse to mixture distributions

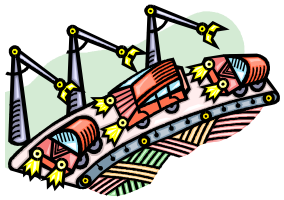
Mixture distributions



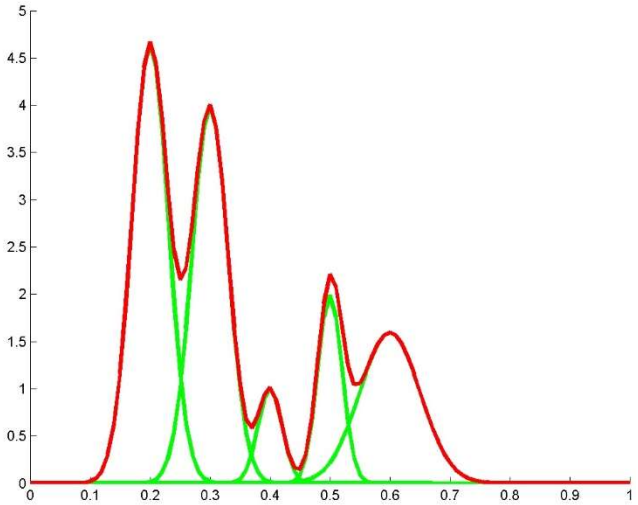
- consider the following problem
  - certain types of traffic banned from a bridge
  - we need a classifier to see if the ban is holding
  - the sensor measures vehicle weight
  - need to classify each car in OK vs banned class
  - we know that in each class there are multiple clusters
  - e.g. OK = {compact, station wagon, SUV}
  - banned = {truck, bus}
  - each of these is close to Gaussian, but for the whole class we get this



Mixture distributions



- this distribution is a mixture
  - the overall shape is determined by a number of sub-classes
  - we introduce a random variable Z to account for this
  - given the value of Z (the cluster) we have a parametric mixture component
  - e.g. a Gaussian



$$P_{\mathbf{X}}(\mathbf{x}) = \sum_{c=1}^{C} P_{\mathbf{X}|Z}(\mathbf{x}|c) P_Z(c)$$

$$= \sum_{c=1}^{C} P_{\mathbf{X}|Z}(\mathbf{x}|c)\pi_c$$

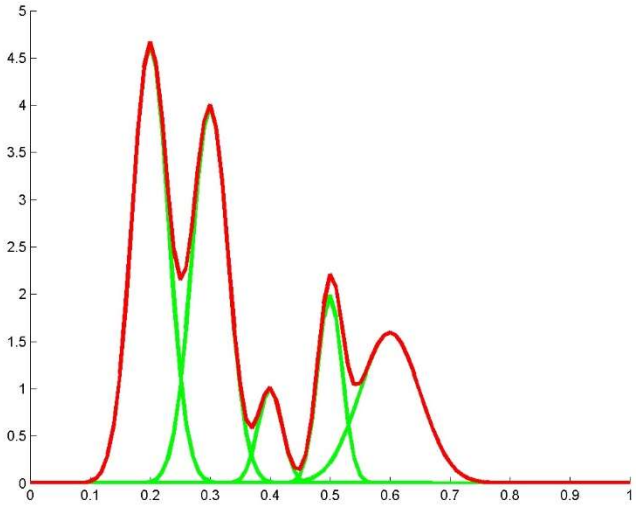
# of mixture components

component "weight"

c^{th} "mixture component"

Mixture distributions

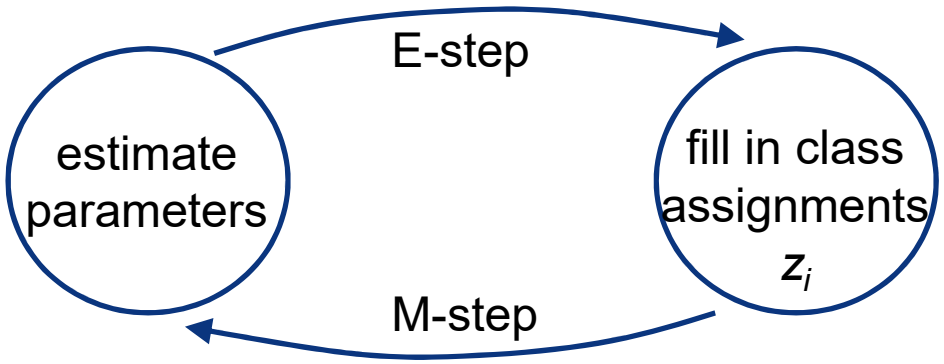
- learning a mixture density is itself a clustering problem
  - for each training point we need to figure out from which component it was drawn
  - once we know the point assignments we need to estimate the parameters of the component



- this could be done with k-means
- a more generic algorithm is expectation-maximization
  - the only difference is that we never assign the points
  - in expectation step we compute the posterior class probabilities
  - but we do not pick the max
  - in the maximization step, the point contributes to all classes
  - weighted by is posterior class probability

Expectation-maximization

- in summary
  1. start with an initial parameter estimate $\Psi^{(0)}$
  2. **E-step:** given current parameters $\Theta^{(i)}$ and observations in D, "guess" what the values of the z_i are
  3. **M-step:** with the new z_i, we have a complete data problem, solve this problem for the parameters, i.e. compute $\Theta^{(i+1)}$
  4. go to 2.
- in a graphical form



Expectation-maximization

- mathematically we have
- expectation:

$$\begin{aligned} h_{ij} &= P_{\mathbf{Z}|\mathbf{X}}(\mathbf{e}_j|\mathbf{x}_i;\boldsymbol{\Psi}^{(i)}) \\ &= \frac{\mathcal{G}\left(\mathbf{x}_i,\mu_j^{(i)},\sigma_j^{(i)}\right)\pi_j^{(i)}}{\sum_{k=1}^{C}\mathcal{G}\left(\mathbf{x}_i,\mu_k^{(i)},\sigma_k^{(i)}\right)\pi_k^{(i)}} \end{aligned}$$

- maximization:

$$\mu_j^{(i+1)} = \frac{\sum_i h_{ij}\mathbf{x}_i}{\sum_i h_{ij}} \qquad \pi_j^{(i+1)} = \frac{1}{n}\sum_i h_{ij}$$

$$\sigma_j^{2(i+1)} = \frac{\sum_i h_{ij}(\mathbf{x}_i-\mu_j)^2}{\sum_i h_{ij}}$$

Expectation-maximization

- note that the difference to k-means is that
  - in E-step h_{ij} would be thresholded to 0 or 1
  - this would make the M-step exactly the same
  - plus we get the estimate of the covariances and class probabilities automatically
  - k-means can be seen as a greedy version of EM
  - at each point we make the optimal decision
  - but this does not take into account other points or future iterations
  - if the hard assignment is best, EM can choose it too

Curse of dimensionality

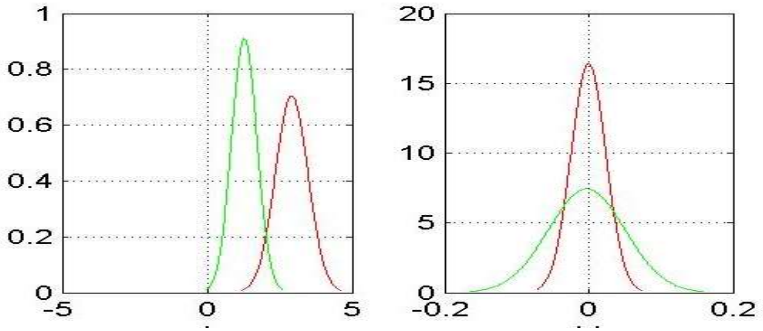
- typical observation in Bayes decision theory:
  - error increases when number of features is large
- problem: even for simple models (e.g. Gaussian) we need large # of examples *n* to have good estimates
- Q: what does "large" mean? This depends on the dimension of the space
- the best way to see this is to think of an histogram
  - suppose you have 100 points and you need at least 10 bins per axis in order to get a reasonable quantization
- for uniform data you get, on average,
  - decent in 1D, bad in 2D, terrible in 3D (9 out of each 10 bins empty)
- this is the curse of dimensionality

dimension	1	2	3
points/bin	10	1	0.1

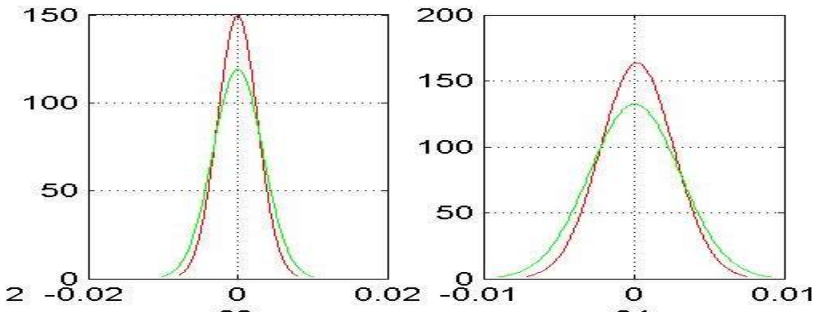
Dimensionality reduction

- what do we do about this? we avoid unnecessary dimensions
- unnecessary can be measured in two ways:
  1. features are not discriminant
  2. features are not independent
- non-discriminant means that they do not separate the classes well

discriminant

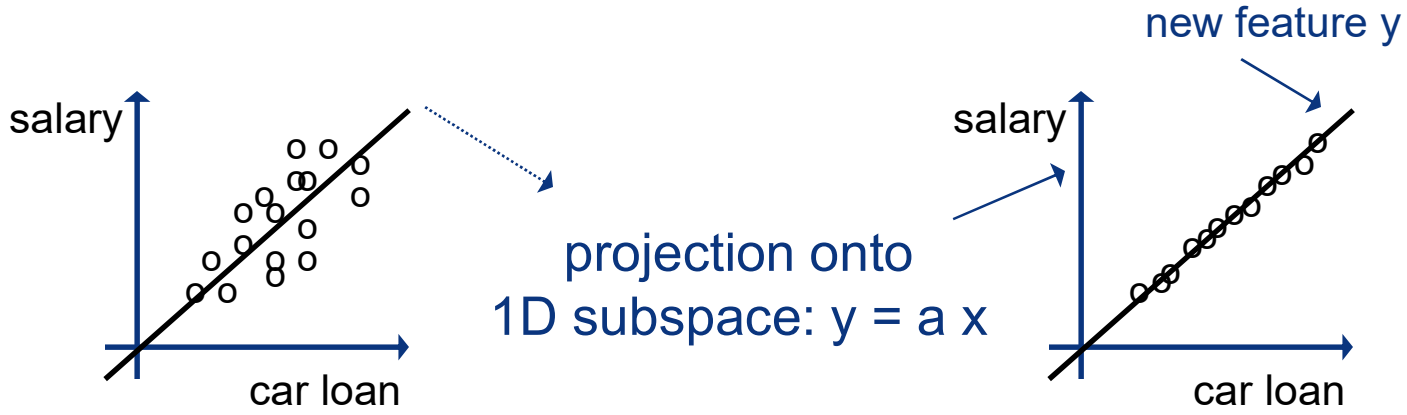


non-discriminant



Dimensionality reduction

- Q: how do we detect the presence of feature correlations?
- A: the data "lives" in a low dimensional subspace (up to some amounts of noise), e.g.

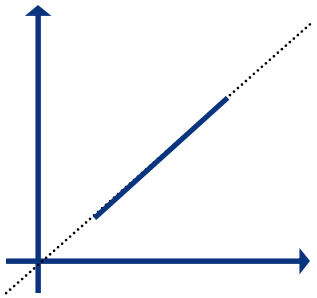


- if we can find this hyper-plane we can
  - project the data onto it
  - get rid of unnecessary dimensions without introducing significant error

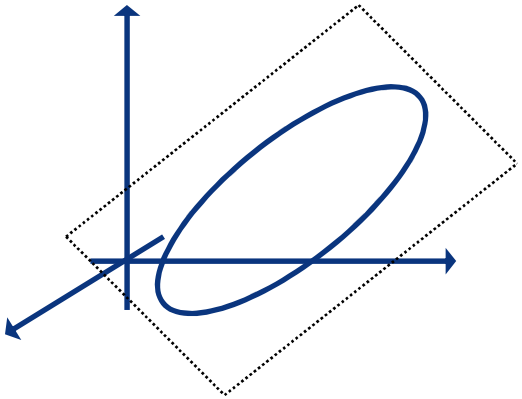
Principal component analysis

- basic idea:
  - if the data lives in a subspace, it is going to look very flat when viewed from the full space, e.g.

1D subspace in 2D



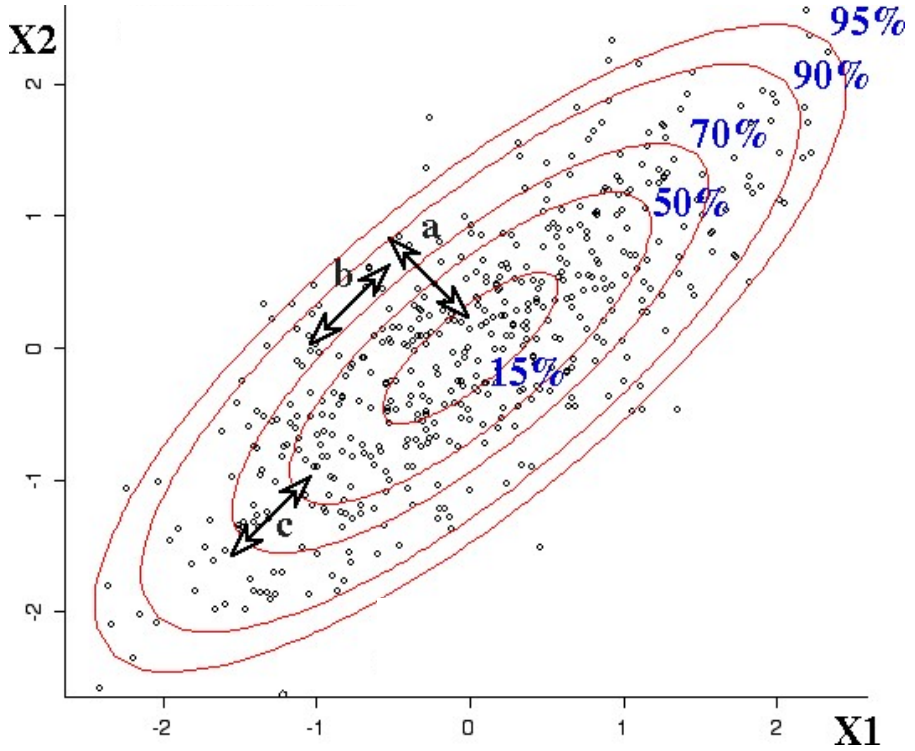
2D subspace in 3D



- this means that
  - if we fit a Gaussian to the data
  - the iso-probability contours are going to be highly skewed ellipsoids

Principal component analysis

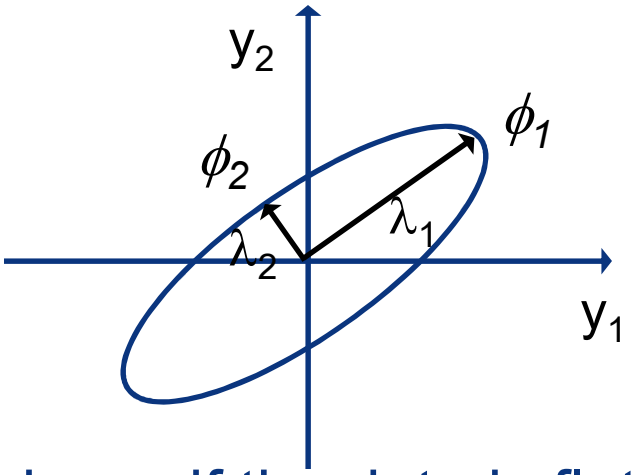
- how do we find these ellipsoids?
- when we talked about metrics we said that
  - the Mahalanobis distance
  - measures the "natural" units for the problem
  - because it is "adapted" to the covariance of the data
- we also know that
  - what is special about it
  - is that it uses Σ^{-1}
- hence, the information must be in Σ



$$d(x,y) = (x-y)^T \Sigma^{-1} (x-y)$$

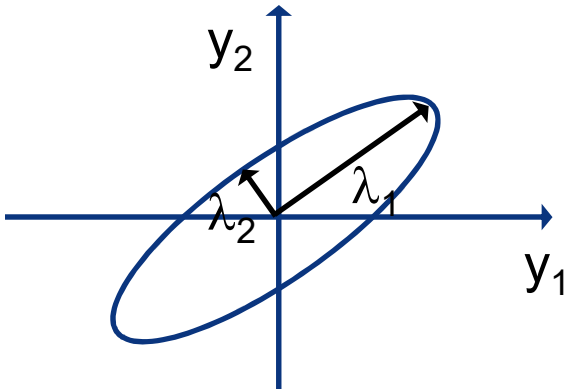
Principal component analysis

- If y is Gaussian with covariance Σ, the equiprobability contours are the ellipses whose
  - principal components ϕ_i are the eigenvectors of Σ
  - principal lengths λ_i are the eigenvalues of Σ

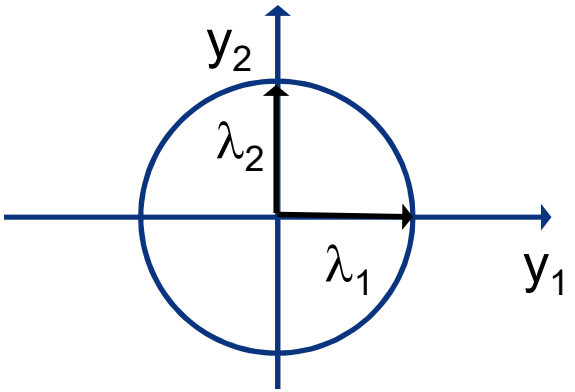


- by computing the eigenvalues we know if the data is flat

$\lambda_1 >> \lambda_2$: flat



$\lambda_1 = \lambda_2$: not flat



Principal component analysis (learning)

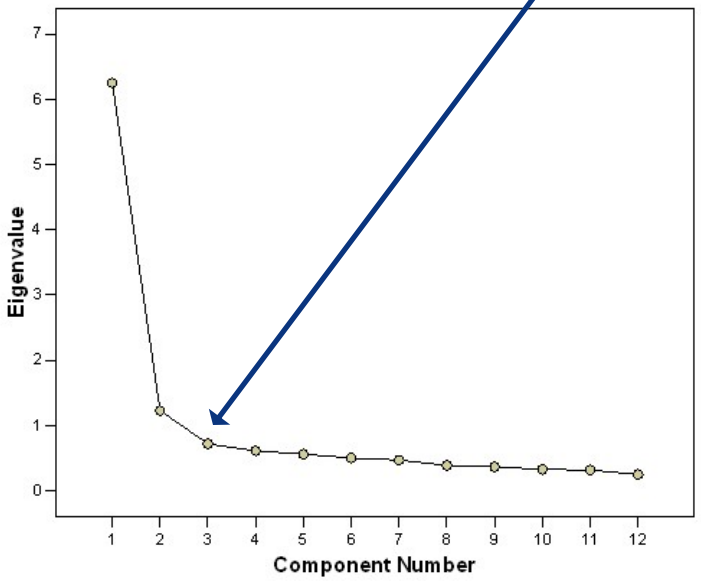
- Given sample $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, $x_i \in \mathcal{R}^d$
  - compute sample mean: $\hat{\mu} = \frac{1}{n}\sum_i(\mathbf{x}_i)$
  - compute sample covariance: $\hat{\Sigma} = \frac{1}{n}\sum_i(\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T$
  - compute eigenvalues and eigenvectors of $\hat{\Sigma}$

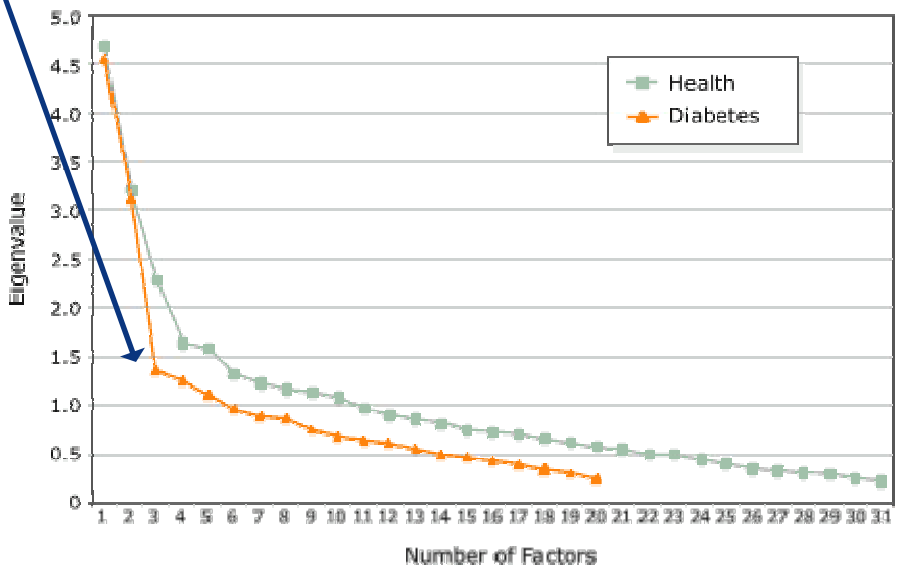
$$\hat{\Sigma} = \Phi^T \Lambda \Phi, \;\; \Lambda = diag(\sigma_1^2, \ldots, \sigma_n^2) \;\; \Phi^T\Phi = I$$

  - order eigenvalues $\sigma_1^2 > ... > \sigma_n^2$
  - if, for a certain k, $\sigma_k << \sigma_1$ eliminate the eigenvalues and eigenvectors above k.

Principal component analysis

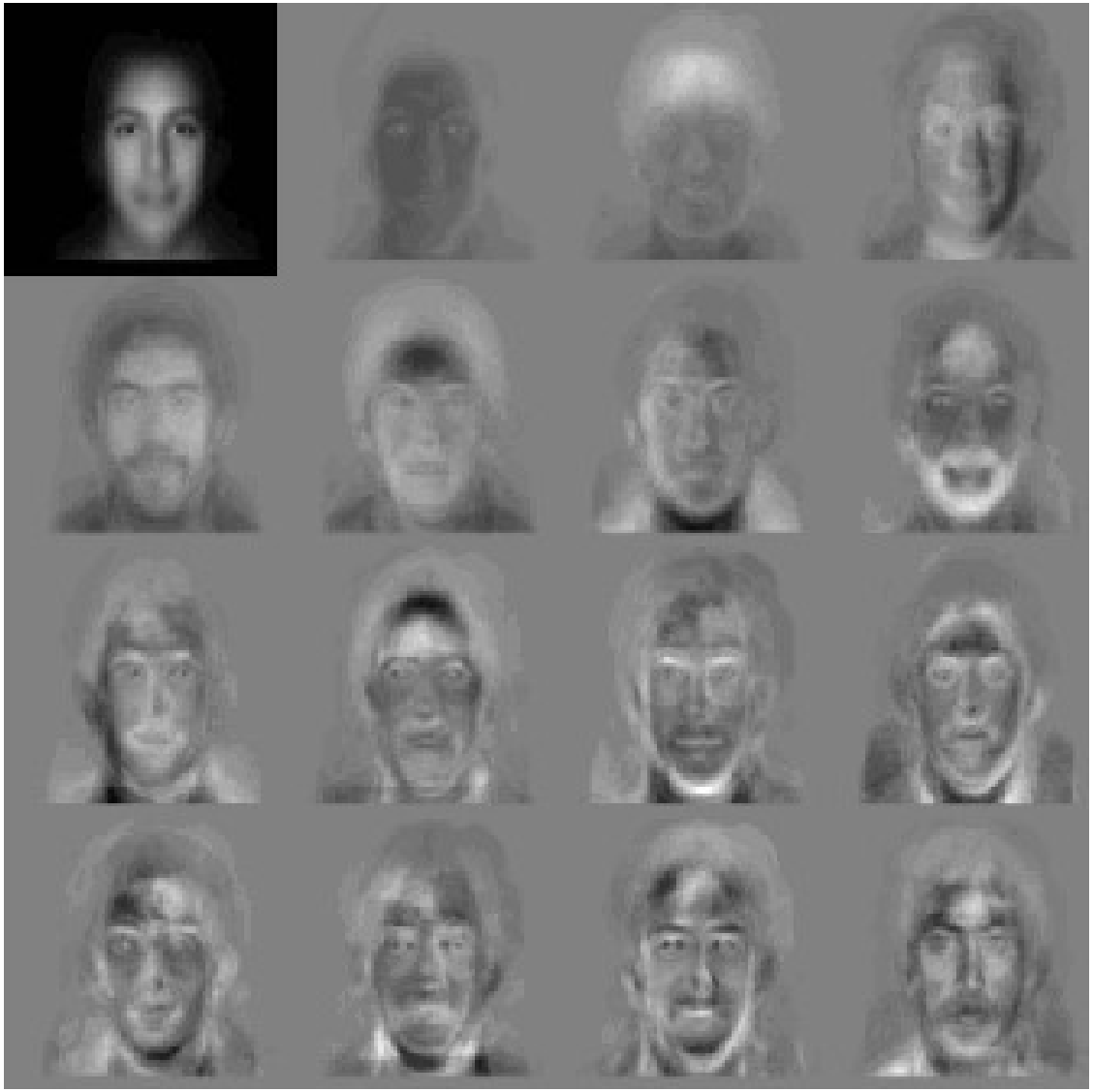
- how do I determine the number of eigenvectors to keep?
  - one possibility is to plot eigenvalue magnitudes
  - this is a scree plot
  - usually there is a fast decrease in the eigenvalue magnitude
  - followed by a flat area
  - one good choice is the knee of this curve





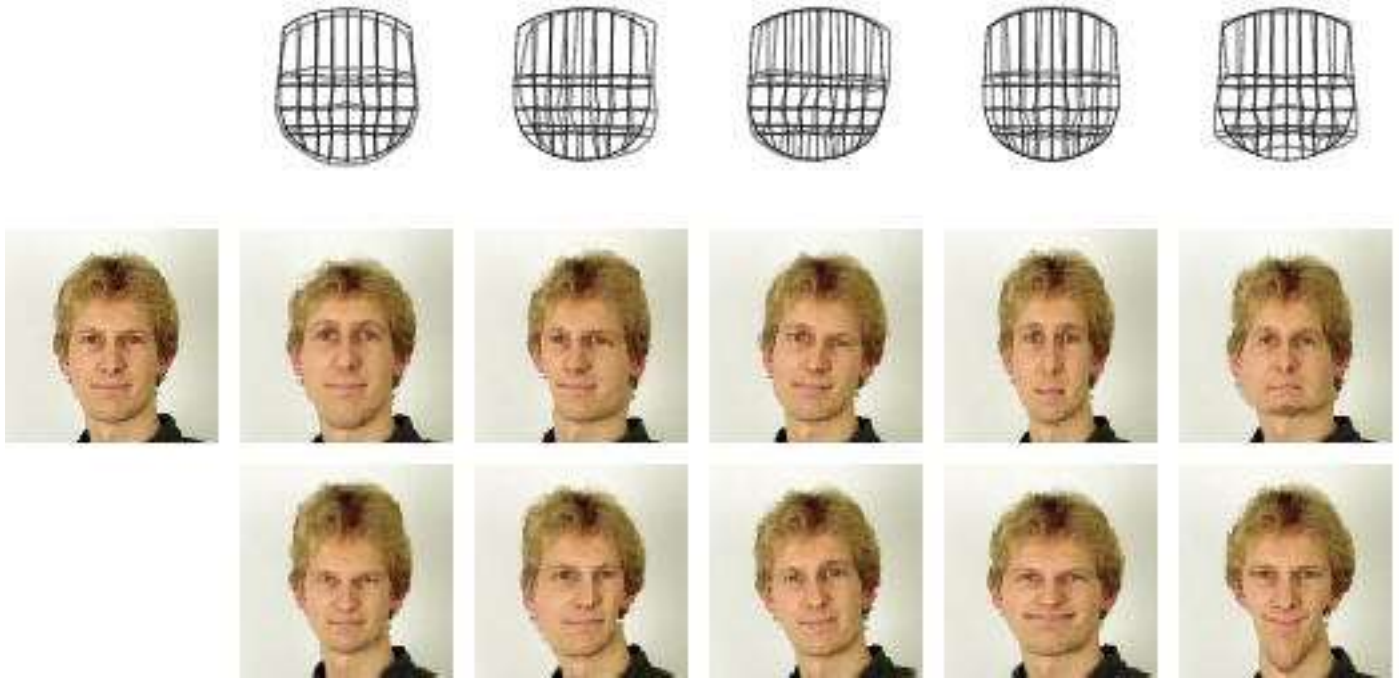
Principal component analysis

- principal components are usually quite informative about the structure of the data
- example
  - the principal components for the space of images of faces
  - the figure only show the first 16 eigenvectors
  - note lighting, structure, etc



Principal components analysis

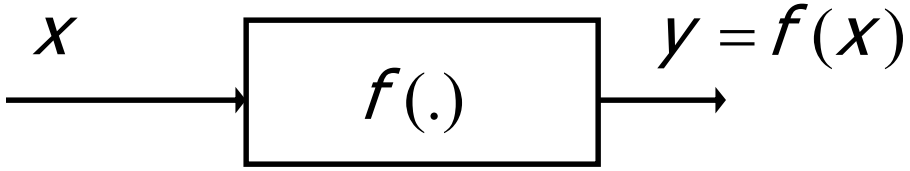
- PCA has been applied to virtually all learning problems
- e.g. eigenshapes for face morphing

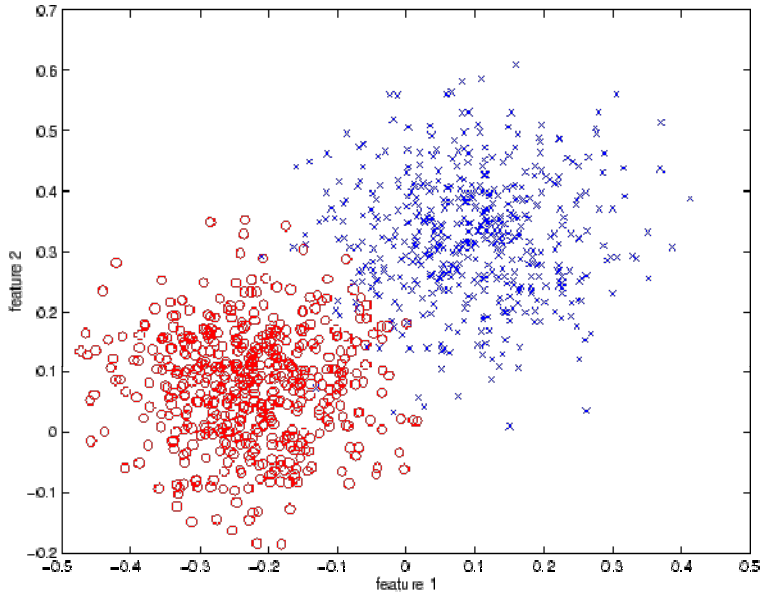


morphed faces

Classification

- a classification problem has two types of variables
  - X - vector of observations (features) in the world
  - Y - state (class) of the world
- e.g.
  - $x \in \mathcal{X} \subset \mathcal{R}^2$ = (fever, blood pressure)
  - $y \in \mathcal{Y}$ = {disease, no disease}
- X, Y related by a (unknown) function





- goal: design a classifier $h: \mathcal{X} \rightarrow \mathcal{Y}$ such that $h(x) = f(x) \ \forall x$

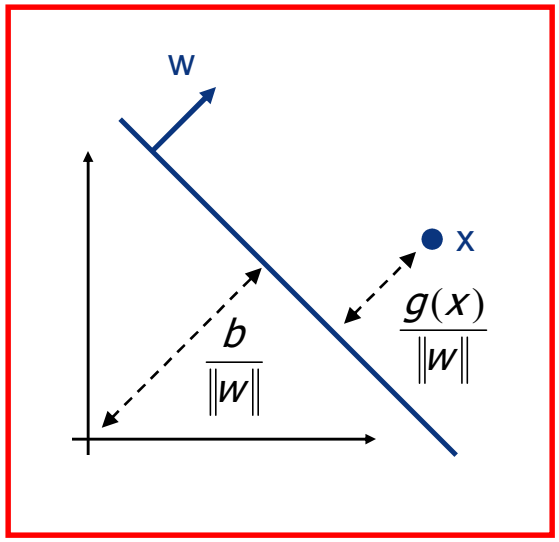
Linear classifier

- implements the decision rule

$$h^*(x) = \begin{cases} 1 & \text{if } g(x) > 0 \\ -1 & \text{if } g(x) < 0 \end{cases} = \text{sgn}[\text{g(x)}]$$

with $g(x) = w^T x + b$

- has the properties
  - it divides $\mathcal{X}$ into two "half-spaces"
  - boundary is the plane with:
    - normal *w*
    - distance to the origin *b*/||*w*||
  - g(x)/||w|| is the distance from point x to the boundary
    - g(x) = 0 for points on the plane
    - g(x) > 0 on the side *w* points to ("positive side")
    - g(x) < 0 on the "negative side"



Linear classifier

- we have a classification error if
  - $y = 1$ and $g(x) < 0$ or $y = -1$ and $g(x) > 0$
  - i.e $y.g(x) < 0$
- and a correct classification if
  - $y = 1$ and $g(x) > 0$ or $y = -1$ and $g(x) < 0$
  - i.e $y.g(x) > 0$
- note that, for a linearly separable training set

  $$D = \{(x_1,y_1), \ldots, (x_n,y_n)\}$$

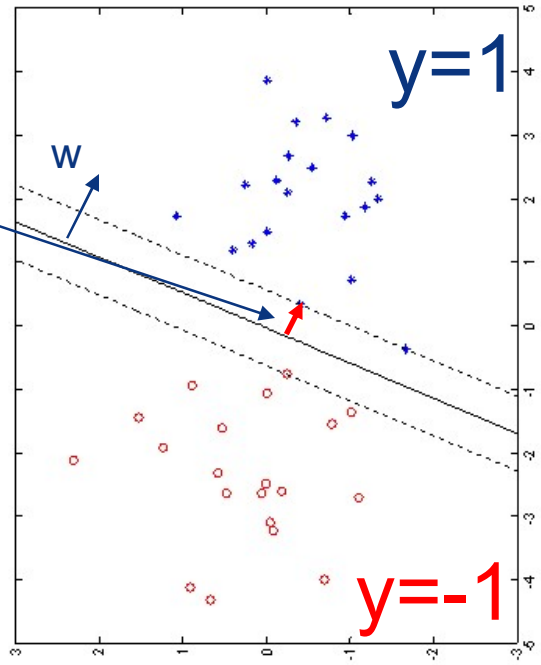
  we can have zero empirical risk
- the necessary and sufficient condition is that

$$y_i\left(w^T x_i + b\right) > 0, \quad \forall i$$

The margin

- is the distance from the boundary to the closest point

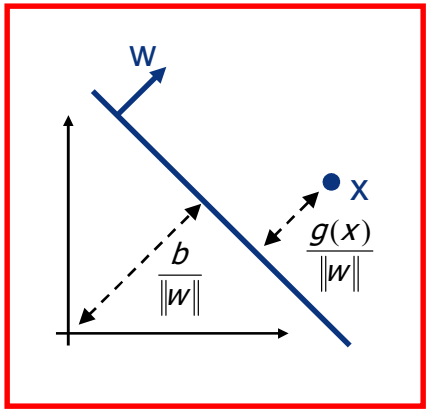
$$\gamma = \min_i \frac{\left| w^T x_i + b \right|}{\|w\|}$$



- there will be no error if it is strictly greater than zero

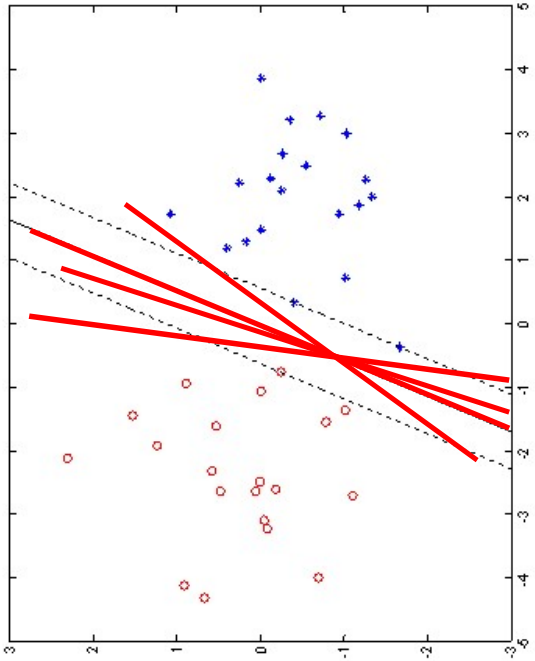
$$y_i \left(w^T x_i + b \right) > 0, \ \forall i \ \Leftrightarrow \ \boxed{\gamma > 0}$$

- note that this is ill-defined in the sense that γ does not change if both w and b are scaled by λ
- we need a normalization



Maximizing the margin

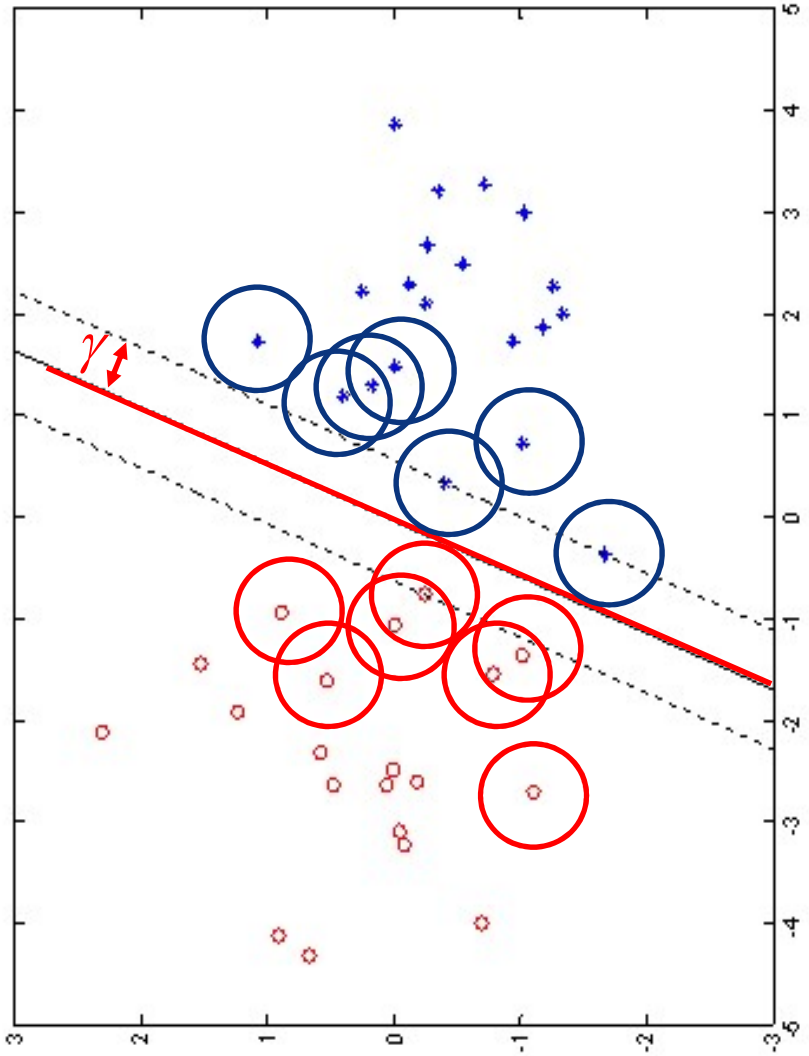
- this is similar to what we have seen for Fisher discriminants
- lets assume we have selected some normalization, e.g. $||w||=1$
- the next question is: what is the cost that we are going to optimize?
- there are several planes that separate the classes, which one is best?



- recall that in the case of the Perceptron, we have seen that the margin determines the complexity of the learning problem
  - the Perceptron converges in less than $(k/\gamma)^2$ iterations
- it sounds like maximizing the margin is a good idea.

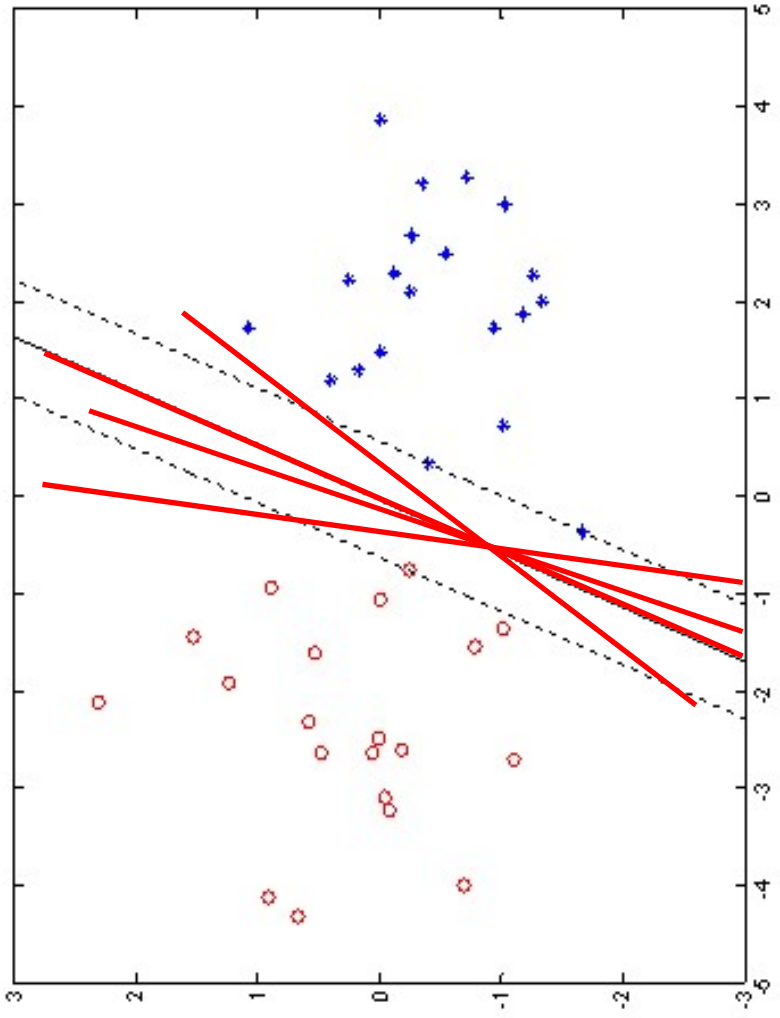
Maximizing the margin

- intuition 1:
  - think of each point in the training set as a sample from a probability density centered on it
  - if we draw another sample, we will not get the same points
  - each point is really a pdf with a certain variance
  - this is a kernel density estimate
  - if we leave a margin of γ on the training set, we are safe against this uncertainty
  - (as long as the radius of support of the pdf is smaller than γ)
  - the larger γ, the more robust the classifier!



Maximizing the margin

- intuition 2:
  - think of the plane as an uncertain estimate because it is learned from a sample drawn at random
  - since the sample changes from draw to draw, the plane parameters are random variables of non-zero variance
  - instead of a single plane we have a probability distribution over planes
  - the larger the margin, the larger the number of planes that will not originate errors
  - the larger γ, the larger the variance allowed on the plane parameter estimates!



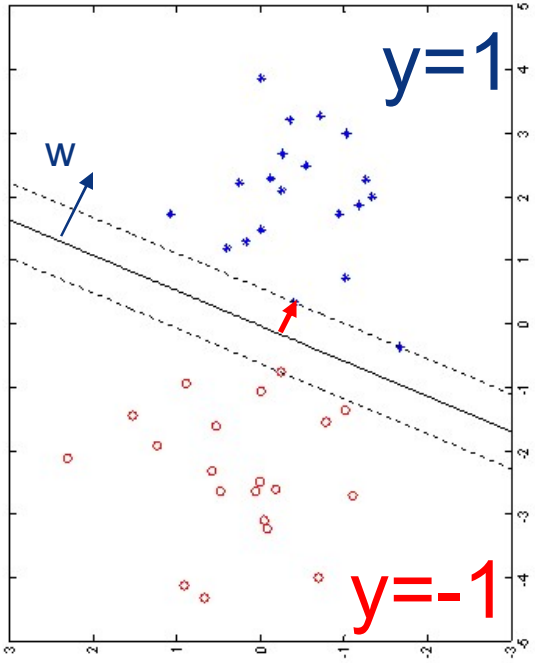
Normalization

- a convenient normalization is to make $|g(x)| = 1$ for the closest point, i.e.

$$\min_i \left| w^T x_i + b \right| \equiv 1$$

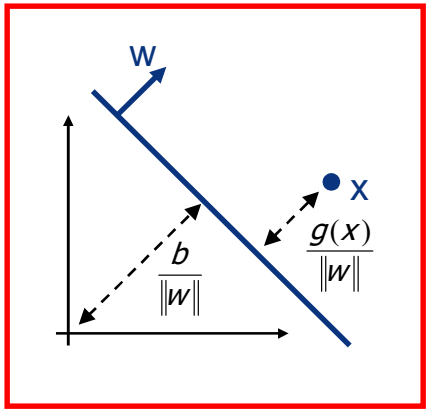
under which

$$\gamma = \frac{1}{\|w\|}$$



- the SVM is the classifier that maximizes the margin under these constraints

$$\min_{w,b} \|w\|^2 \quad \text{subject to} \quad y_i\left(w^T x_i + b\right) \geq 1 \quad \forall i$$



Duality

- this is an optimization problem with constraints
- there is a rich theory on how to solve such problems
  - (we will not get into it here!)
  - the main result is that we can formulate a dual problem which is easier to solve
  - in the dual formulation we introduce a vector of Lagrange multipliers α_i, one associated with each constraint, and solve

$$\max_{\alpha \geq 0} q(\alpha) = \max_{\alpha \geq 0} \left\{ \min_{w} L(w, b, \alpha) \right\}$$

  - where

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i} \alpha_i \left[y_i \left(w^T x_i + b \right) - 1 \right]$$

    is the Lagrangian

The dual problem

- for the SVM, the dual problem can be simplified into

$$\max_{\alpha \geq 0} \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{\mathrm{i}} \alpha_i \right\}$$

$$\text{subject to } \sum_{\mathrm{i}} y_i \alpha_i = 0$$

- once this is solved, the vector

$$w^* = \sum_i \alpha_i y_i x_i$$

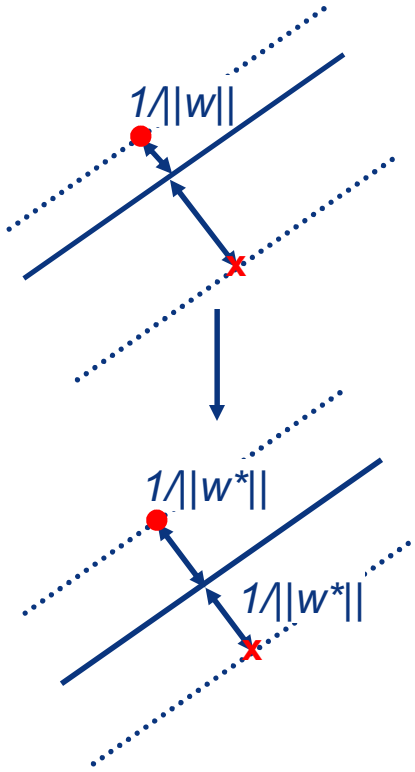
  is the normal to the maximum margin plane

- note: the dual solution does not determine the optimal b^*, since b drops off when we solve $\min_{\mathrm{w}} L(w, b, \alpha)$

The dual problem

- determining b*
  - various possibilities, for example
  - pick one point x^+ on the margin on the *y=1* side and one point x^- on the *y=-1* side
  - use the margin constraint

$$\left.\begin{aligned} w^T x^+ + b &= 1 \\ w^T x^- + b &= -1 \end{aligned}\right\} \Leftrightarrow \boxed{b^* = -\frac{w^T (x^+ + x^-)}{2}}$$



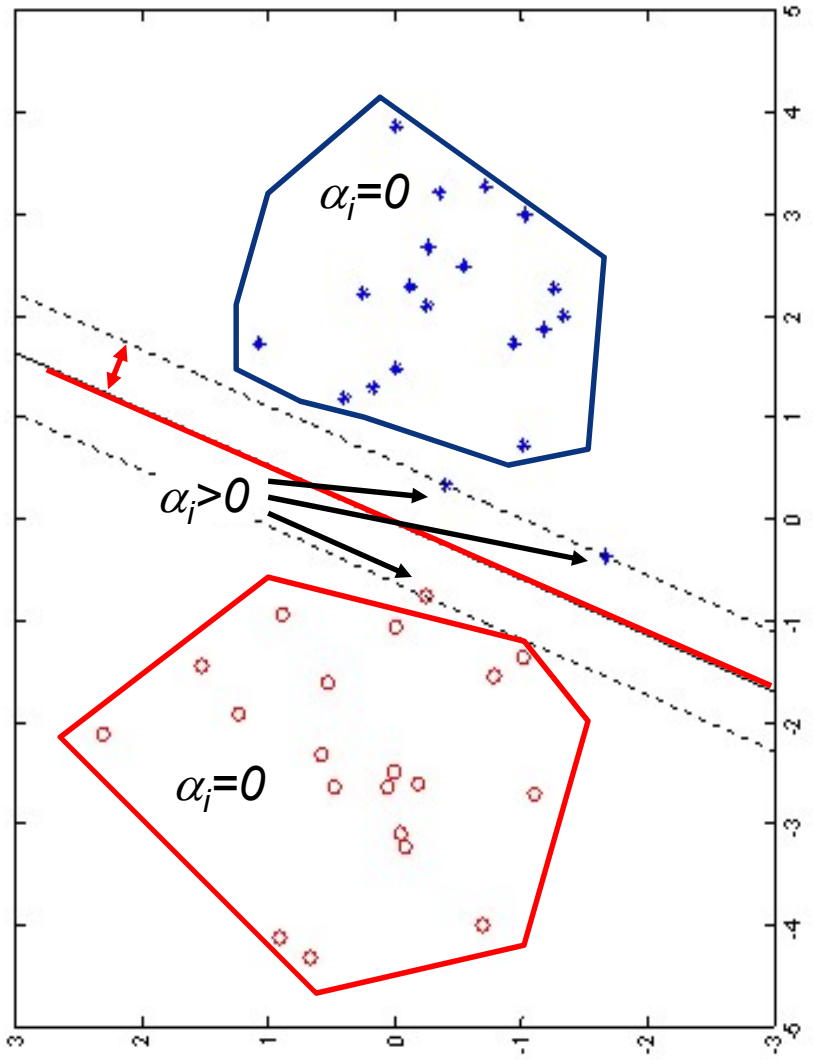
- note:
  - the maximum margin solution guarantees that there is always at least one point "on the margin" on each side
  - if not, we could move the plane and get a larger margin

Support vectors

- it turns out that
- an inactive constraint always has zero Lagrange multiplier α_i.
- that is,
  - i) $\alpha_i > 0$ and $y_i(w^{*T}x_i + b^*) = 1$ or
  - ii) $\alpha_i = 0$ and $y_i(w^{*T}x_i + b^*) \geq 1$
- hence $\alpha_i > 0$ only for points

$$|w^{*T}x_i + b^*| = 1$$

which are those that lie at a distance equal to the margin



Support vectors

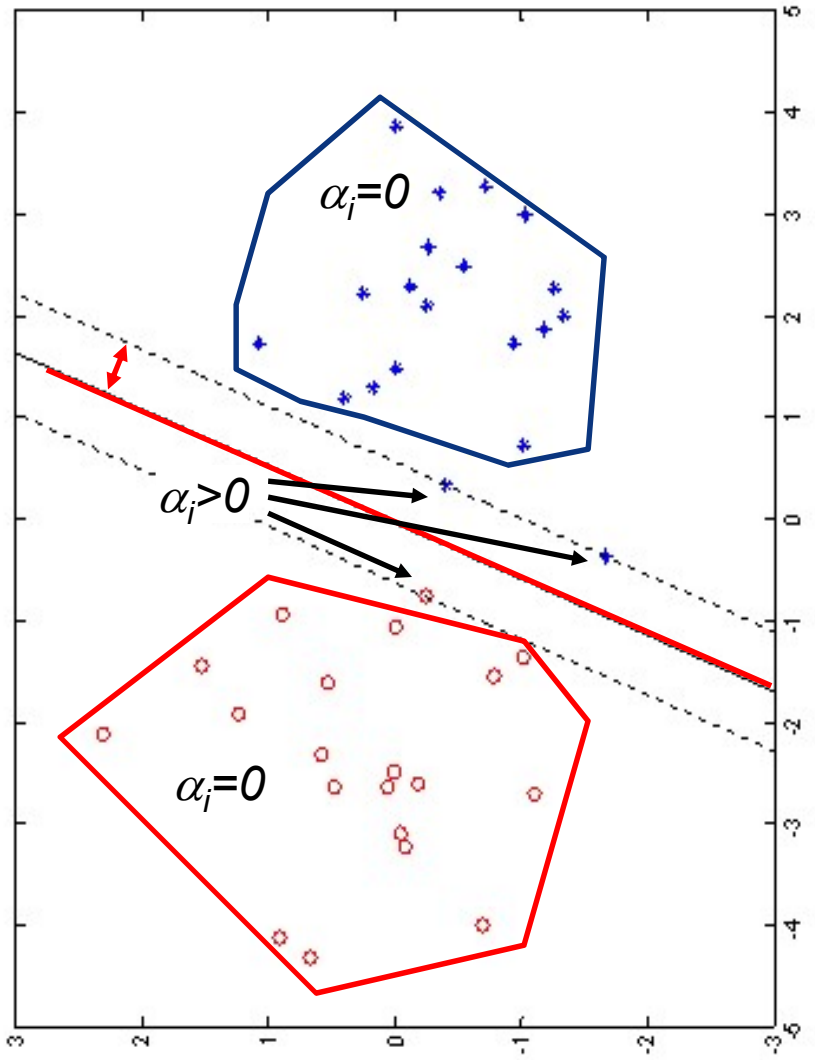
- points with $\alpha_i > 0$ support the optimal plane (w*,b*).
- this why they are called "support vectors"
- note that the decision rule is

$$f(x) = \mathrm{sgn}\left[w^{*T} x + b^*\right]$$

$$= \mathrm{sgn}\left[\sum_i y_i \alpha_i^* x_i^T x + b^*\right]$$

$$= \mathrm{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^*\right]$$

where $SV = \{i \mid \alpha^*_i > 0\}$ is the set of support vectors



Support vectors

- since the decision rule is

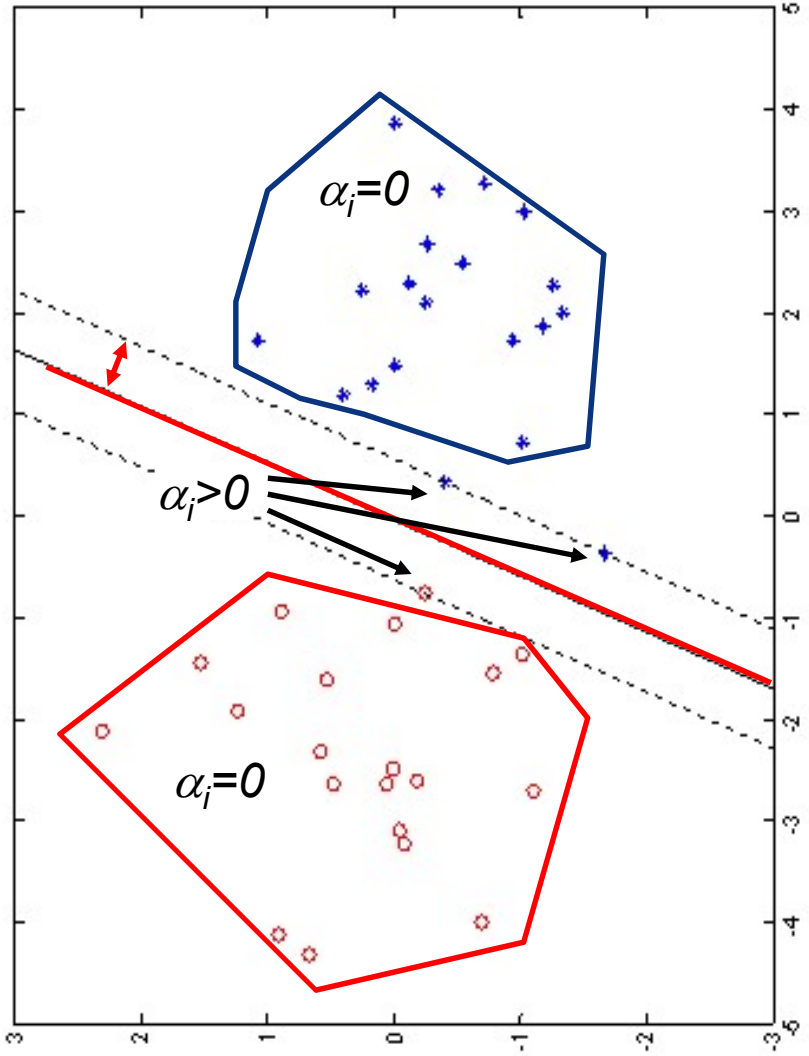
$$f(x) = \text{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^*\right]$$

  we only need the support vectors to completely define the classifier

- we can literally throw away all other points!

- the Lagrange multipliers can also be seen as a measure of importance of each point

- points with $\alpha_i=0$ have no influence, small perturbation does not change solution

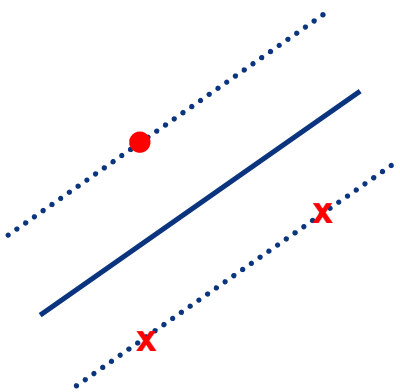


The robustness of SVMs

- we talked a lot about the "curse of dimensionality"
  - number of examples required to achieve certain precision is exponential in the number of dimensions
- it turns out that SVMs are remarkably robust to dimensionality
  - not uncommon to see successful applications on 1,000D+ spaces
- two main reasons for this:
  - 1) all that the SVM does is to learn a plane.

    Although the number of dimensions may be large, the number of parameters is relatively small and there is no much room for overfitting

    In fact, *d+1* points are enough to specify the decision rule in R^d!



SVMs as feature selectors

- the second reason is that the space is not really that large
  - 2) the SVM is a feature selector

To see this let's look at the decision function

$$f(x) = \text{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^*\right].$$

This is a thresholding of the quantity

$$\boxed{\sum_{i \in SV} y_i \alpha_i^* x_i^T x}$$

note that each of the terms $x_i^T x$ is the projection of the vector to classify (x) into the training vector x_i

SVMs as feature selectors

- defining z as the vector of the projection onto all support vectors

$$z(x) = \left(x^T x_{i_1}, \cdots, x^T x_{i_k}\right)^T$$

- the decision function is a plane in the z-space

$$f(x) = \operatorname{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^*\right] = \operatorname{sgn}\left[\sum_k w_k^* z_k(x) + b^*\right]$$

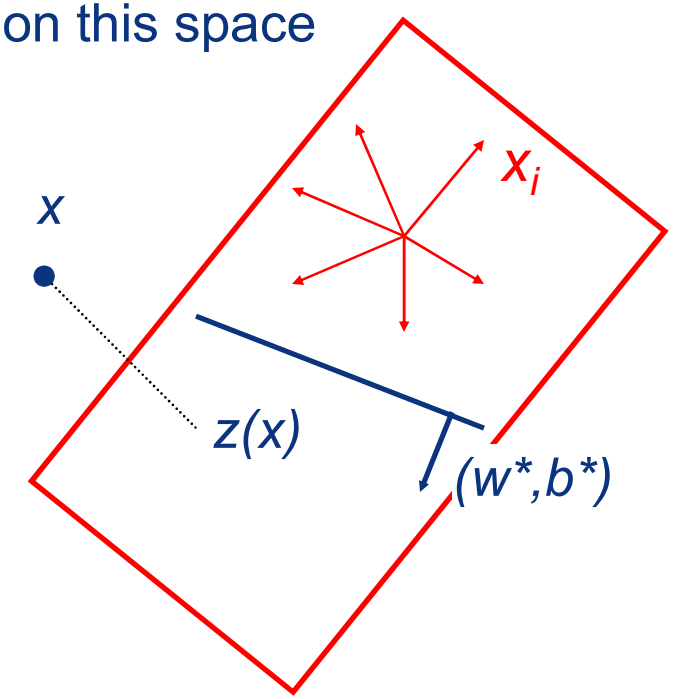
with

$$w^* = \left(\alpha_{i_1}^* y_{i_1}, \cdots, \alpha_{i_k}^* y_{i_k}\right)^T$$

- this means that
  - the classifier operates on the span of the support vectors!
  - the SVM performs feature selection automatically

SVMs as feature selectors

- geometrically, we have:
  - 1) projection on the span of the support vectors
  - 2) classifier on this space



$$w^* = \left(\alpha^*_{i_1} y_{i_1}, \cdots, \alpha^*_{i_k} y_{i_k}\right)^T$$

  - the effective dimension is |SV| and, typically, |SV| << n

In summary

- SVM training:
  - 1) solve

$$\max_{\alpha \geq 0} \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{\mathrm{i}} \alpha_i \right\}$$

$$\text{subject to } \sum_{\mathrm{i}} y_i \alpha_i = 0$$

  - 2) then compute

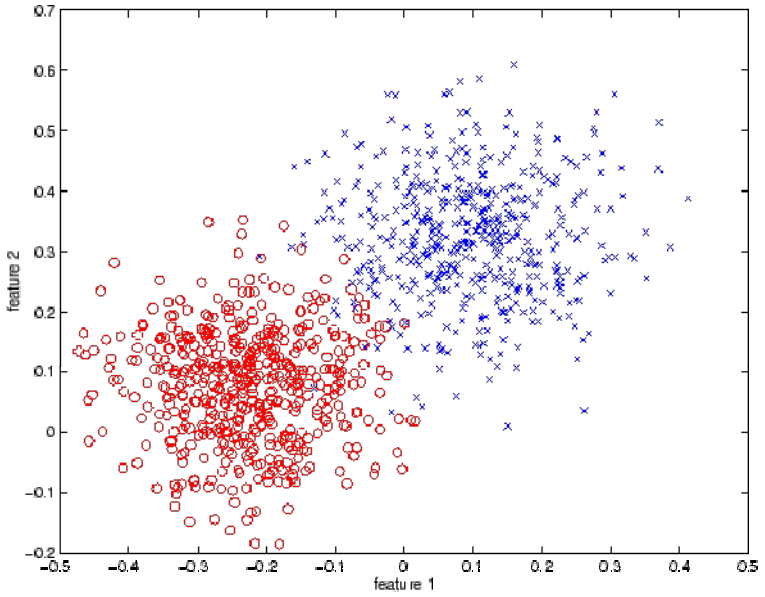
$$w^* = \sum_{i \in SV} \alpha_i^* y_i x_i \qquad b^* = -\frac{1}{2} \sum_{i \in SV} y_i \alpha_i^* \left(x_i^T x^+ + x_i^T x^- \right)$$

- decision function:

$$f(x) = \operatorname{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^* \right]$$

Non-separable problems

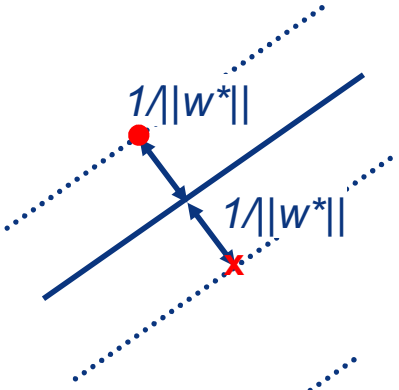
- so far we have assumed linearly separable classes
- this is rarely the case in practice
- a separable problem is “easy” most classifiers will do well
- we need to be able to extend the SVM to the non-separable case
- basic idea:
  - with class overlap we cannot enforce a margin
  - but we can enforce a soft margin
  - for most points there is a margin, but then there are a few outliers that cross-over, or are closer to the boundary than the margin



Soft margin optimization

- mathematically this can be done by introducing slack variables
- instead of solving

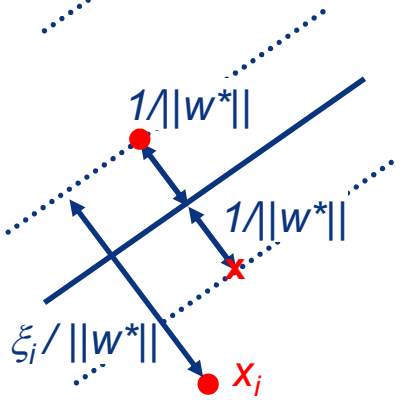
$$\min_{w,b} \|w\|^2 \quad \text{subjectto } y_i\left(w^T x_i + b\right) \geq 1 \quad \forall i$$



- we solve the problem

$$\min_{w,\xi,b} \|w\|^2 \quad \text{subject to } y_i\left(w^T x_i + b\right) \geq 1 - \xi_i \quad \forall i$$

$$\xi_i \geq 0, \forall i$$



- the ξ_i are called slacks
- basically, the same as before but points with $\xi_i > 0$ are allowed to violate the margin

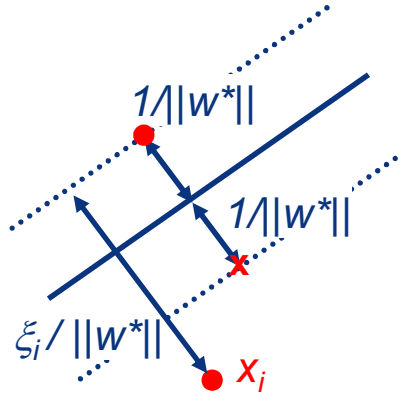
Soft margin optimization

- note that the problem is not really well defined
- by making ξ_i arbitrarily large, any w will do
- we need to penalize large ξ_i
- this is done by solving instead

$$\min_{w,\xi,b} \|w\|^2 + C\sum_i \xi_i$$

$$\text{subject to } y_i\left(w^T x_i + b\right) \geq 1 - \xi_i \quad \forall i$$

$$\xi_i \geq 0, \forall i$$



- the term $C\sum_i \xi_i$ is the penalty, C controls how harsh it is

The dual problem

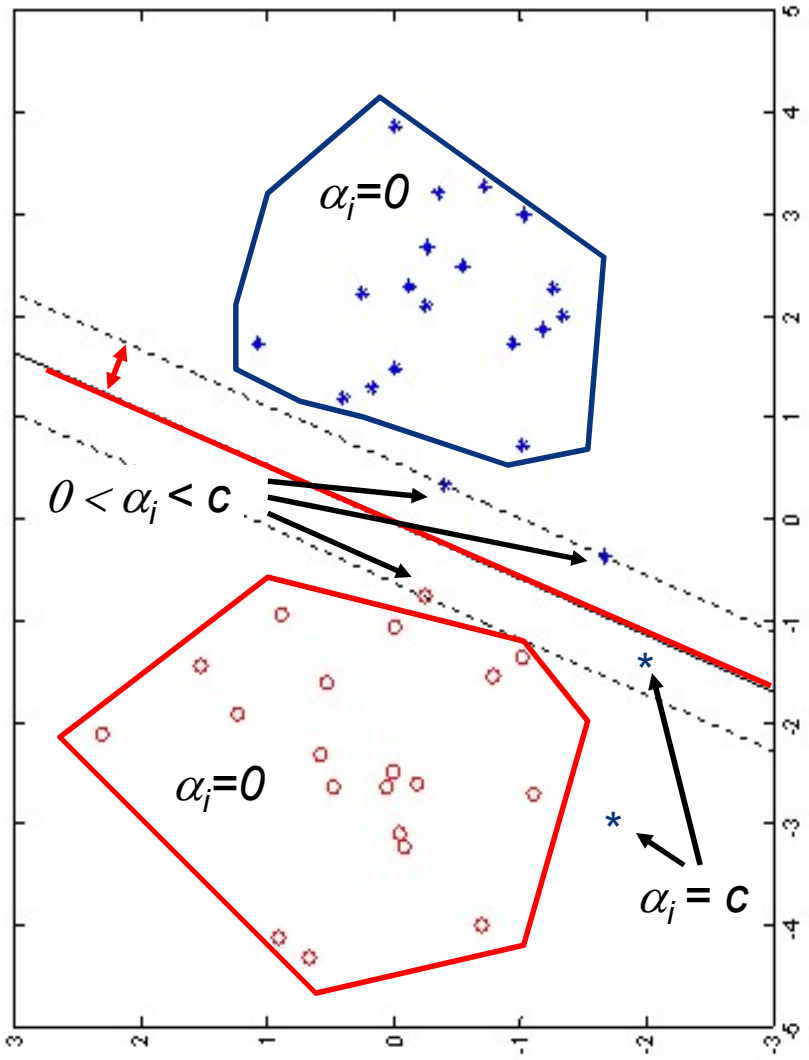
- the dual problem is

$$\max_{\alpha \geq 0} \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i \right\}$$

$$\text{subject to} \quad \sum_i y_i \alpha_i = 0,$$

$$0 \leq \alpha_i \leq C$$

- the only difference with respect to the hard margin case is the box constraint on the α_i.
- geometrically we have this



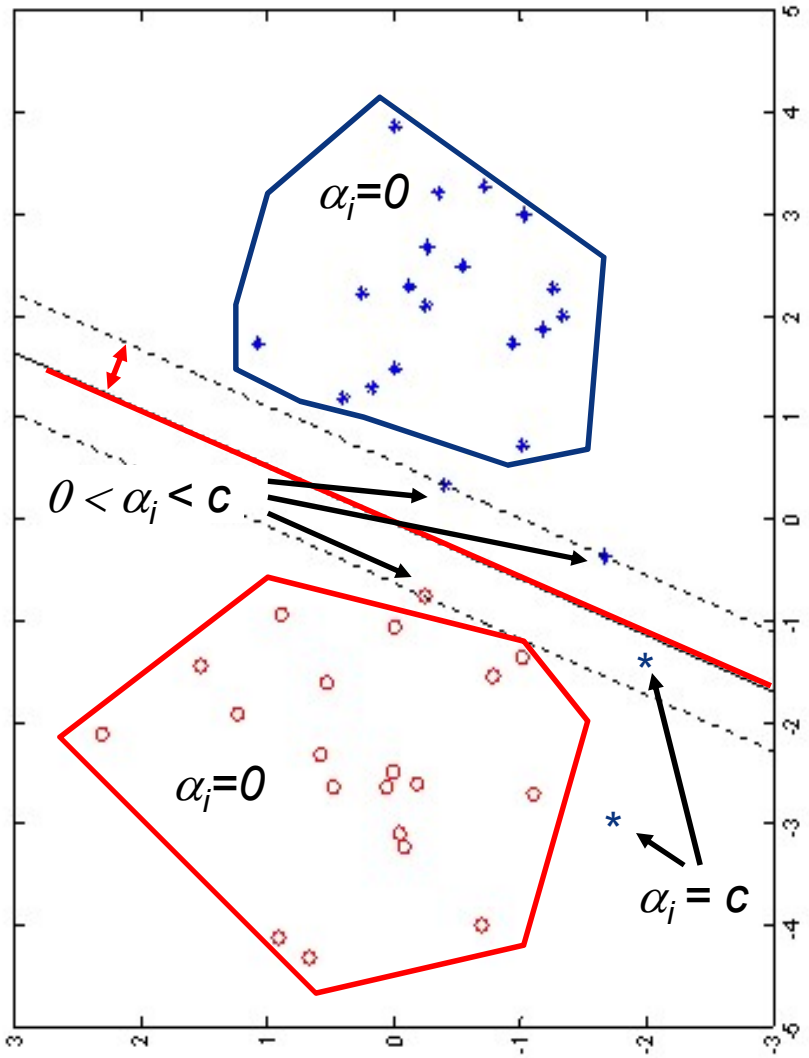
Support vectors

- are the points with $\alpha_i > 0$
- as before, the decision rule is

$$f(x) = \text{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^*\right]$$

where $SV = \{i \mid \alpha^*_i > 0\}$

- *and b* chosen s.t.*
  - $y_i g(x_i) = 1$, for all x_i s.t. $0 < \alpha_i < C$
- the box constraint on Lagrange multipliers
  - makes intuitive sense
  - prevents a single outlier from having large impact



Kernelization

- note that all equations depend only on $x_i^T x_j$
- the kernel trick is trivial: replace by $K(x_i, x_j)$
  - 1) training

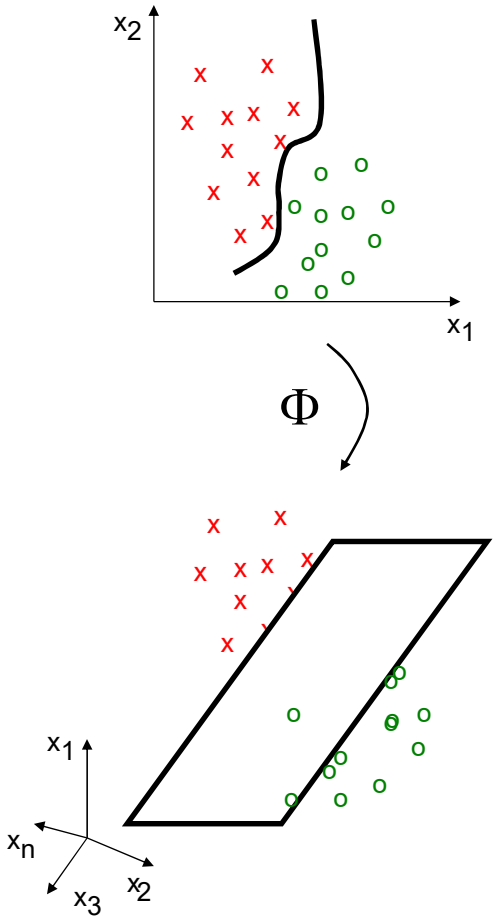
$$\max_{\alpha \geq 0} \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j k(x_i, x_j) + \sum_i \alpha_i \right\}$$

$$\text{subject to } \sum_i y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C$$

$$b^* = -\frac{1}{2} \sum_{i \in SV} y_i \alpha_i^* \left(K(x_i, x^+) + K(x_i, x^-) \right)$$

  - 2) decision function:

$$f(x) = \operatorname{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* K(x_i, x) + b^* \right]$$



Kernelization

- notes:
  - as usual this follows from the fact that nothing of what we did really requires us to be in R^d.
  - we could have simply used the notation $\langle x_i, x_j \rangle$ for the dot product and all the equations would still hold
  - the only difference is that we can no longer recover w* explicitly without determining the feature transformation Φ, since

$$w^* = \sum_{i \in SV} \alpha_i^* y_i \Phi(x_i)$$

  - this could have infinite dimension, e.g. we have seen that it is a sum of Gaussians when we use the Gaussian kernel
  - but, luckily, we don't really need w*, only the decision function

$$f(x) = \text{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* K(x_i, x) + b^* \right]$$

Input space interpretation

- when we introduce a kernel, what is the SVM doing in the input space?
- let's look again at the decision function

$$f(x) = \operatorname{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* K(x_i, x) + b^*\right]$$

with

$$b^* = -\frac{1}{2}\sum_{i \in SV} y_i \alpha_i^* \left(K(x_i, x^+) + K(x_i, x^-)\right)$$

- note that
  - x^+ and x^- are support vectors
  - assuming that the kernel as reduced support when compared to the distance between support vectors

Input space interpretation

- note that
  - assuming that the kernel as reduced support when compared to the distance between support vectors

$$b^* = -\frac{1}{2}\sum_{i \in SV} y_i \alpha_i^* \left(K\left(x_i, x^+\right) + K\left(x_i, x^-\right)\right)$$

$$\approx -\frac{1}{2}\left[\alpha_+^* K\left(x^+, x^+\right) - \alpha_-^* K\left(x^-, x^-\right)\right]$$

$$\approx 0$$

  - where we have also assumed that $\alpha_+ \sim \alpha_-$
  - these assumptions are not crucial, but simplify what follows
  - namely the decision function is

$$f(x) = \operatorname{sgn}\left[\sum_{i \in SV} y_i \alpha_i^* K\left(x_i, x\right)\right]$$

Input space interpretation

- or

$$f(x)=\begin{cases}1, & \text{if } \sum_{i\in SV} y_i\alpha_i^* K(x_i,x)\geq 0\\ -1, & \text{if } \sum_{i\in SV} y_i\alpha_i^* K(x_i,x)<0\end{cases}$$

- rewritting

$$\sum_{i\in SV} y_i\alpha_i^* K(x_i,x)=\sum_{i|y_i>0}\alpha_i^* K(x_i,x)-\sum_{i|y_i<0}\alpha_i^* K(x_i,x)$$

- this is

$$f(x)=\begin{cases}1, & \text{if } \sum_{i|y_i\geq 0}\alpha_i^* K(x_i,x)\geq \sum_{i|y_i<0}\alpha_i^* K(x_i,x)\\ -1, & \text{otherwise}\end{cases}$$

Input space interpretation

- or

$$f(x)=\begin{cases}1, & \text{if } \frac{1}{\sum_{i|y_i<0}\alpha_i^*}\sum_{i|y_i\geq 0}\pi_i^* K(x_i,x)\geq \frac{1}{\sum_{i|y_i\geq 0}\alpha_i^*}\sum_{i|y_i<0}\beta_i^* K(x_i,x)\\ -1, & \text{otherwise}\end{cases}$$

- with

$$\pi_i^*=\frac{\alpha_i^*}{\sum_{i|y_i\geq 0}\alpha_i^*}, i\,|\,y_i\geq 0 \qquad \beta_i^*=\frac{\alpha_i^*}{\sum_{i|y_i<0}\alpha_i^*}, i\,|\,y_i<0$$

- which is the same as

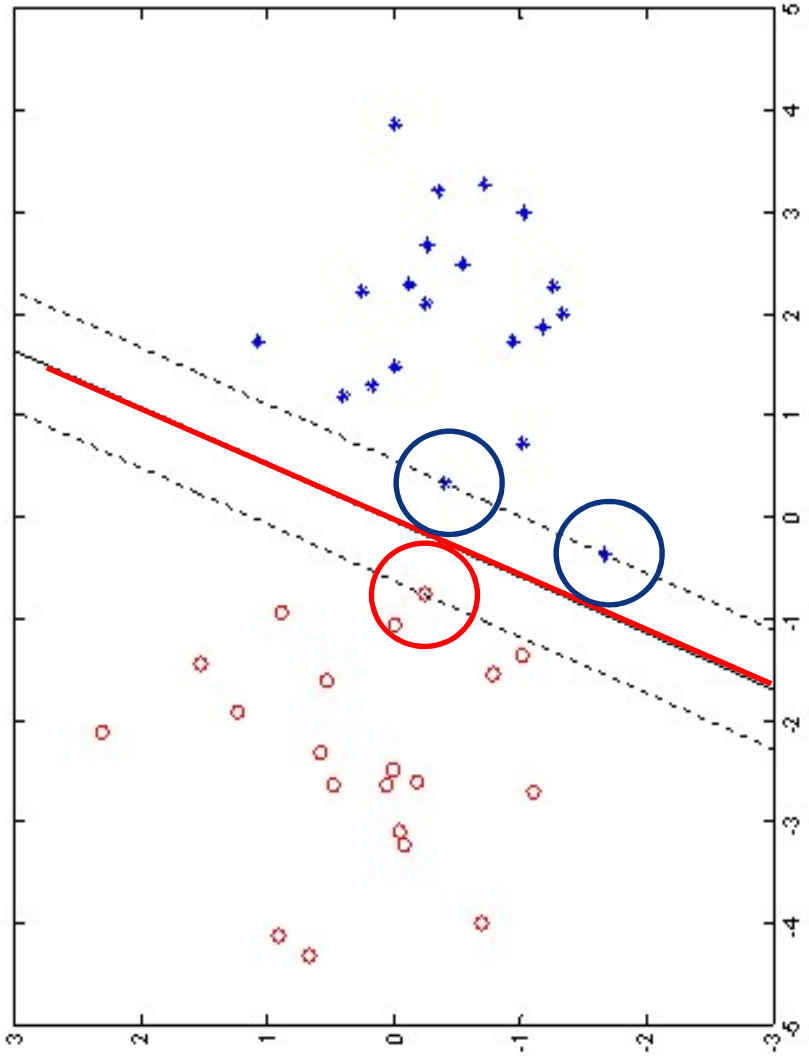
$$f(x)=\begin{cases}1, & \text{if } \frac{\sum_{i|y_i\geq 0}\pi_i^* K(x_i,x)}{\sum_{i|y_i<0}\beta_i^* K(x_i,x)}\geq \frac{\sum_{i|y_i<0}\alpha_i^*}{\sum_{i|y_i\geq 0}\alpha_i^*}\\ -1, & \text{otherwise}\end{cases}$$

Input space interpretation

- note that this is the Bayesian decision rule for
  - 1) class 1 with likelihood $\sum_{i|y_i \geq 0} \pi_i^* K(x_i, x)$ and prior $\sum_{i|y_i<0} \alpha_i^* / \sum_i \alpha_i^*$
  - 2) class 2 with likelihood $\sum_{i|y_i<0} \beta_i^* K(x_i, x)$ and prior $\sum_{i|y_i \geq 0} \alpha_i^* / \sum_i \alpha_i^*$
- these likelihood functions
  - can be seen as density estimates if $k(.,x_i)$ is a valid pdf
  - peculiar density estimate that only places kernels around the support vectors
  - all other points are ignored

Input space interpretation

- this is a discriminant form of density estimation
- concentrate modeling power where it matters the most, i.e. near classification boundary
- smart, since points away from the boundary are always well classified, even if density estimates in their region are poor
- the SVM can be seen as a highly efficient combination of the BDR with density estimation



Input space interpretation

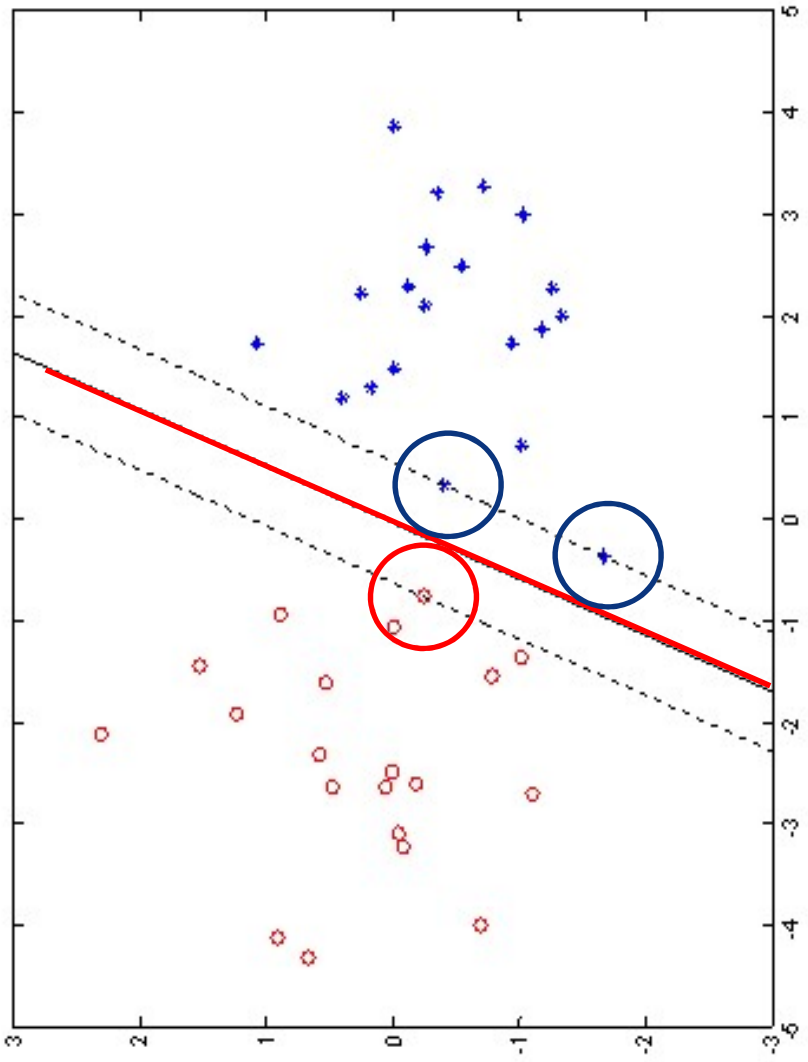
- note on the approximations made:
  - this result was derived assuming b~0
  - in practice, *b* is frequently left as a parameter which is used to trade-off false positives for misses
  - here, that can be done by controlling the BDR threshold

$$f(x) = \begin{cases} 1, & \text{if } \dfrac{\sum_{i|y_i \geq 0} \pi_i^* K(x_i, x)}{\sum_{i|y_i < 0} \beta_i^* K(x_i, x)} \geq T \\ -1, & \text{otherwise} \end{cases}$$

  - hence, there is really not much practical difference, even when the assumption of $b^*=0$ does not hold!

Limitations of the SVM

- appealing, but also points out the limitations of the SVM:
  - major problem is the selection of appropriate kernel
  - no generic "optimal" procedure to find the kernel or its parameters
    - usually we pick an arbitrary kernel, e.g. Gaussian
    - determine the parameters, e.g. variance, by trial and error
  - *C* controls the importance of outliers (larger *C* = less outliers)
  - not really intuitive how to set up *C*



- usually cross-validation, there is a need to cross-validate with respect to both *C* and kernel parameters

Practical implementations

- in practice we need an algorithm for solving the optimization problem of the training stage
  - this is still a complex problem
  - there has been a large amount of research in this area
  - coming up with “your own” algorithm is not going to be competitive
  - luckily there are various packages available:
    - libSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm/
    - SVM light: http://www.cs.cornell.edu/People/tj/svm_light/
    - SVM fu: http://five-percent-nation.mit.edu/SvmFu/
    - various others (see http://www.support-vector.net/software.html)
- SVMs were the neural networks of the 90’s ☺