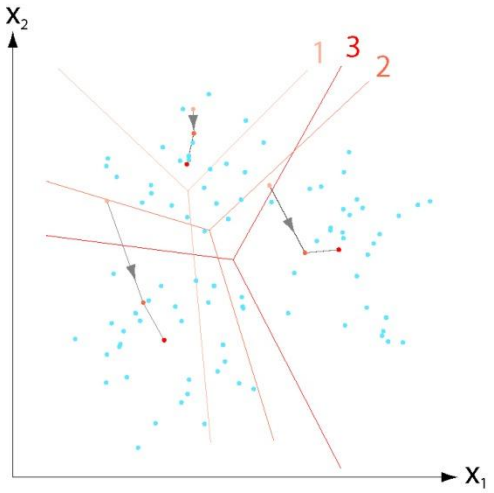
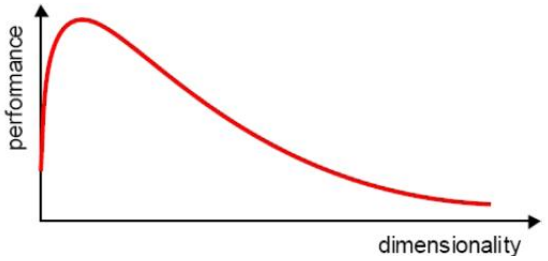
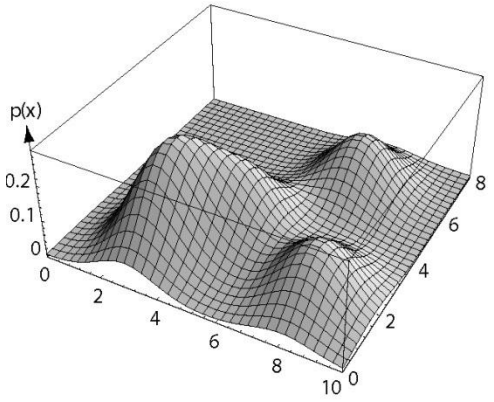






INF4300
Making good use of data

Asbjørn Berge 27-10-2010

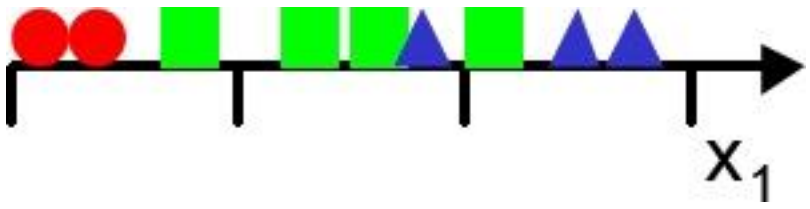
Today's plan

prtools

- Motivation for exploring data and reducing dimensionality
- Dimensionality reduction
  - Feature extraction
  - Feature selection
- Classifier performance and errors
  - Estimating error
  - Confusion matrix
  - Training and test dataset, and how to efficiently use your data.
  - Comment on complexity ⇔ generalization performance tradeoff
  - Outliers / rejection and doubt

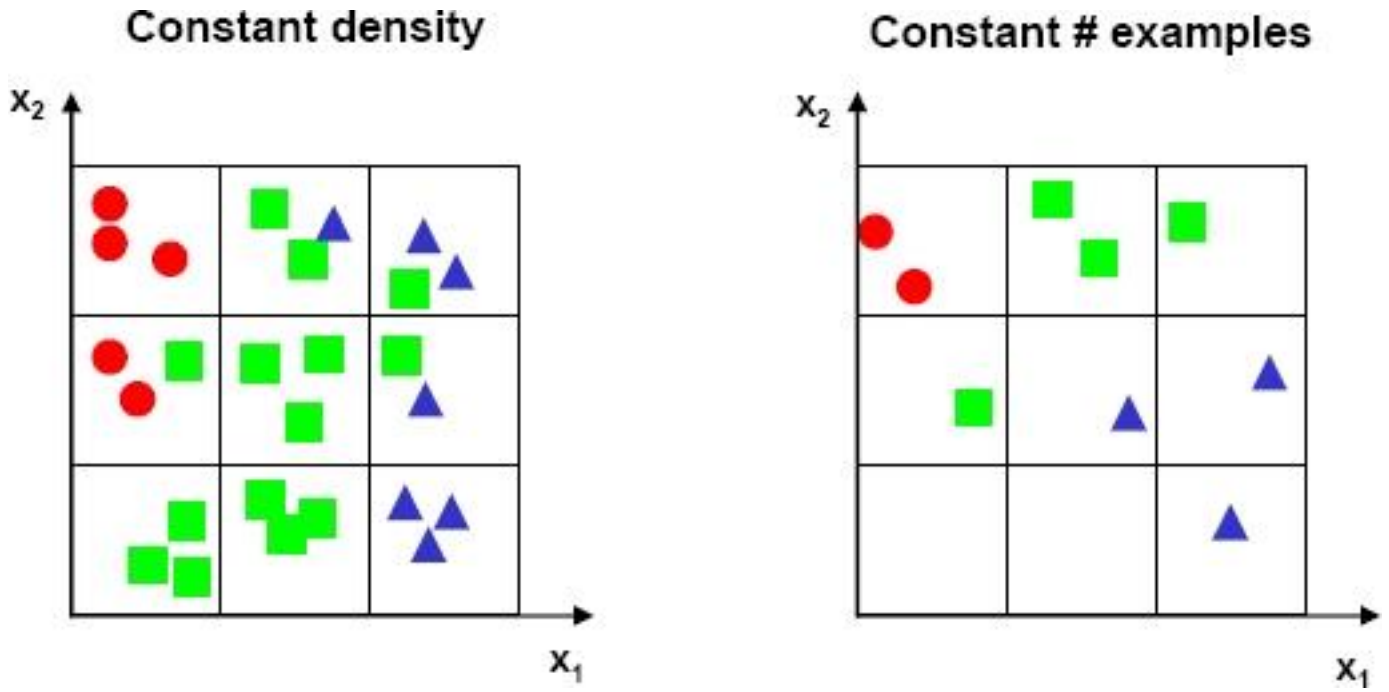
The ”curse” of dimensionality

- Very simple example, three class classification problem, 9 samples
  - Divide the space into bins and classify to majority



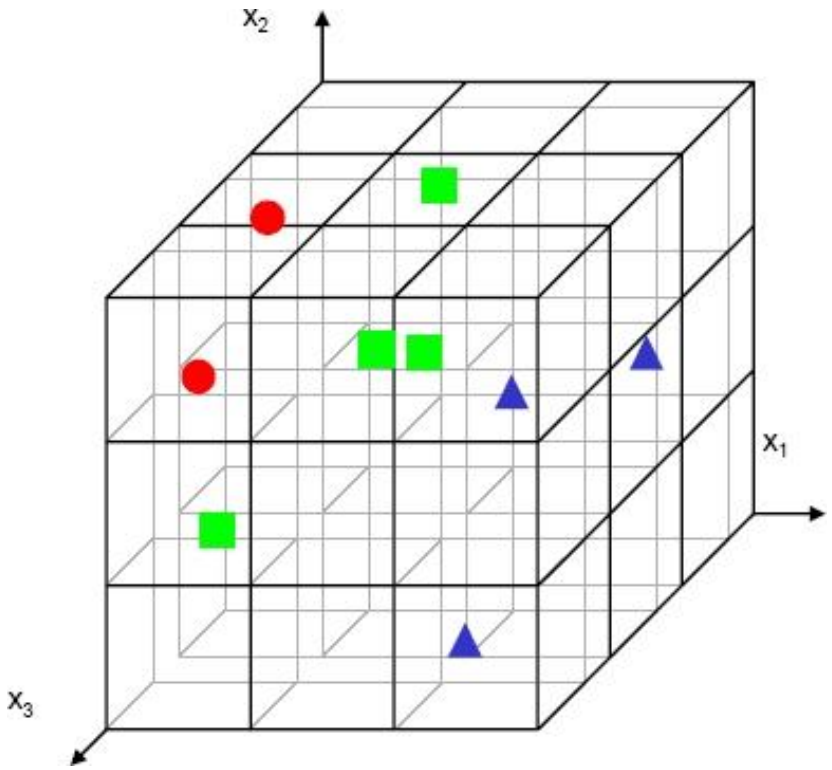
The ”curse” of dimensionality

- Keep the bin resolution, and increase dimensionality
  - *3* bins in 1D increases to 3^2 *bins in 2D*
  - Roughly 3 examples per bin in 1D, if we want to preserve the density of examples we now need 27 samples!



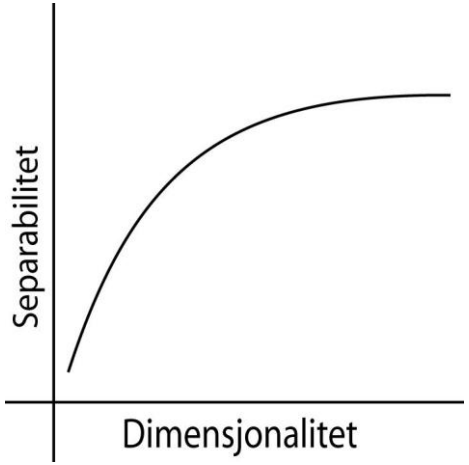
The ”curse” of dimensionality

- The problem escalates rapidly!
  - *81* examples needed to preserve density
  - If we use the original amount of samples (*9*), *2/3* of the feature space is *empty!*

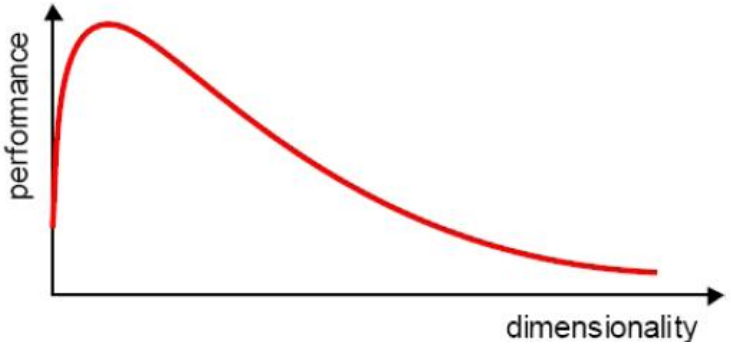


The ”curse” of dimensionality

- In practice, the curse means that, for a given sample size, there is a maximum number of features one can add before *any* classifier starts to degrade.



The big **but** resulting from space being empty



How do we beat the ”curse of dimensionality”?

- Generate few, but informative features
  - Careful feature design given the application
- Solve a simpler problem
  - Reducing the dimensionality
    - Feature selection
    - Feature transforms
  - Regularization
- Faced with model choices. Need performance metrics (error measures) and methods to evaluate these.

Feature evaluation and selection

- Why reduce the number of features we use to describe the data?
  - Countering overfitting
    - More room for samples to reside in, when dataset dimensionality increases
  - Reducing variance in parameter estimates
    - Want to use as many samples as possible to estimate each parameter in the model
  - In the extreme case – make estimates numerically stable
    - Samples ~ dimensionality of data means that e.g. covariance matrix at risk of being singular and impossible to invert
  - Common rule of thumb
    - To get reasonable estimates we need a number of samples *5-10* times the dimensionality

Evaluating classification performance

- To choose the best classifier for a task, we need to define some metrics.
- There is no superior classifier for all kinds of problems, so we're stuck with using heuristics to make a choice.
- Some classification approaches have parameters we can tune using these heuristics
- We would like to know what kind of performance we can expect on new (unseen) data

Overall error

testc

- One common way of defining the quality is overall error rate
  - Usually, it is a weighted average of errors from each class weighted by class prior

$$\varepsilon = \sum_j \frac{(\#\text{incorrect samples from class } \omega_j) * P(\omega_j)}{(\#\text{total samples})}$$

  - What would happen if?
    - We define error as number of correct samples ratioed by the total number of samples?
    - Our prior estimates are ”wrong”?
    - E.g. “my classifier gives right answers 80% of the time” Is it good? Why (not)? What happens if 80% of the data has the “N” label and my classifier always say “N”?

Confusion matrix

confmat

- A convenient way of evaluating classifiers – avoiding such pitfalls - is the confusion matrix
- Plot the true class labels versus the class labels assigned by the classificator
- From this we can read the distribution of incorrectly classified samples

Confusion matrix

confmat

True class labels

Classification

	Class ω_1	Class ω_2	Class ω_3	Total
Class ω_1	80	15	5	100
Class ω_2	5	140	5	150
Class ω_3	25	50	125	200
Total	110	205	135	450

Confusion matrix – derived measures of classification accuracy

confmat

- Overall accuracy
  - Makes sense to evaluate normalized by (true) class size
  - !! Many researchers do not, however!
- Precision
  - How accurate (precise) is the classification on *each* class?
  - #"correct label ω"/#"total classified label ω"
  - Look at rows
- Recall
  - What is the chance of choosing correctly within each class?
  - #"correct label ω"/#"total true label ω"
  - Calculate from the columns
- Kappa
  - How much better is the classifier than random guessing?
  - Compare diagonal of your confusion matrix with one due to random chance

True class labels

Classification

	Class ω_1	Class ω_2	Class ω_3	Total
Class ω_1	80	15	5	100
Class ω_2	5	140	5	150
Class ω_3	25	50	125	200
Total	110	205	135	450

$$\text{Accuracy} = \frac{\sum_i \omega_{ii}}{\sum_{i,j} \omega_{ij}}$$

$$\text{Precision}(\omega_i) = \frac{\omega_{ii}}{\sum \omega_{i\bullet}}$$

$$\text{Recall}(\omega_i) = \frac{\omega_{ii}}{\sum \omega_{\bullet i}}$$

$$\kappa = \frac{P(c) - P(r)}{1 - P(r)} = \frac{\sum_i \omega_{ii} - \sum_{i,j} \omega_{i\bullet} * \omega_{\bullet j} / \omega_{ij}}{\sum_{i,j} \omega_{ij} - \sum_i \omega_{ii}}$$

Outliers and doubt

rejectc

- Two rather vague errors in a classification problem is outliers and doubt samples
- We might want an ideal classifier to report
  - 'this sample is from class I' (usual case)
  - 'this sample is not from any of the classes' (outlier)
  - 'this sample is too hard for me' (doubt/reject)
- The two last cases should lead to an rejection of the sample

Outliers

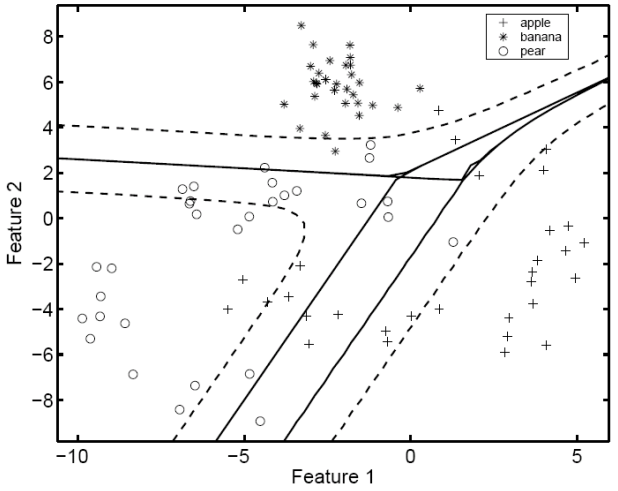
rejectc

- Outliers are heuristically defined as ”..samples which did not (or are thought not to have) come from the assumed population of samples”
- The outliers can result from some breakdown in preprocessing (or even before we aquire an image)
- One way to deal with outliers is to model them as an own class, for example a gaussian with a very large variance, and estimate prior probability from the training data
- Another approach is to decide on some threshold on the aposteriori – and if a sample falls below this threshold for all classes, then declare it an outlier.

Doubt samples

rejectc

- Doubt samples are samples for which the class with the highest probability is not significantly more probable than some of the other classes (e.g. two classes have essentially equal probability).
- Classify as doubt if $p(x|\omega_i)P(\omega_i) < 1-c$, where c is given by the user.
- c must be in the range [0, K-1/K] if we have K classes.
- Some classification software can allow the user to specify thresholds for doubt
- Other software choose the simpler solution of just *guessing*



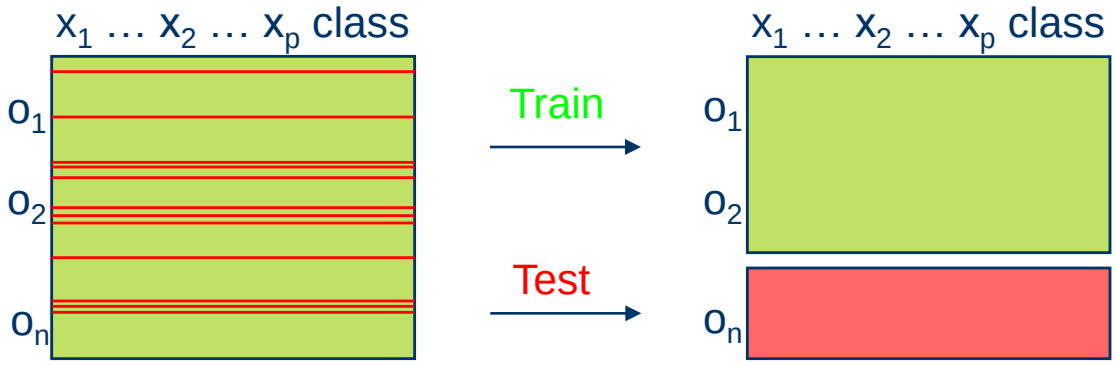
Training and test dataset, and how to efficiently use your data.

- In the ideal case we want to maximize the size of the training and test dataset
- Obviously there is a fixed amount of available data with known labels
- A very simple approach is to separate the dataset in two random subsets, but we can do better!
- The number of features for each object is an important factor with regards to the amount of available data

Back to good use of training data

gendat

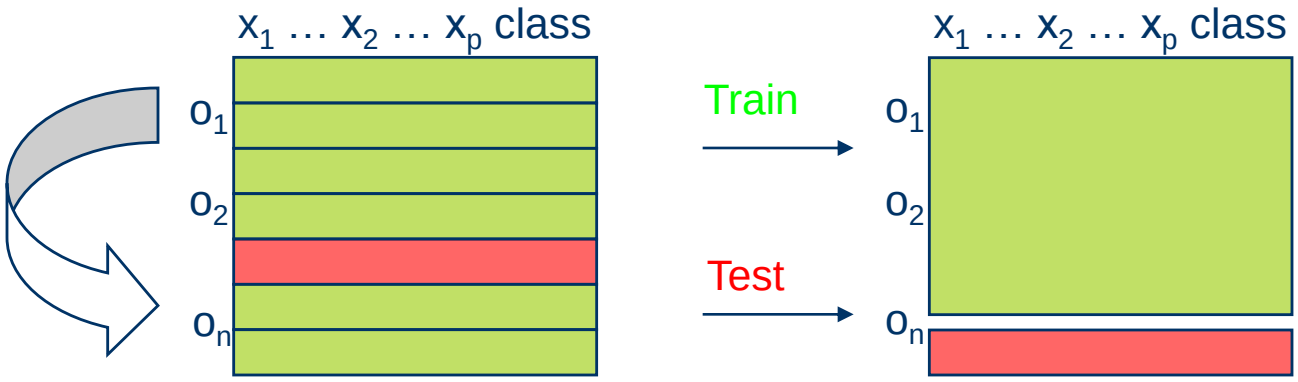
- “Hold out”, ok for large (>1000 objects) datasets
- Simply put away a part of the training data, say 1/3 of the samples chosen randomly – train on the 2/3 remaining, and evaluate classifier performance (error and so on) on the 1/3.
- Can repeat this a couple of times, and report the average of repetitions.
- Problem: repeated draws overlap



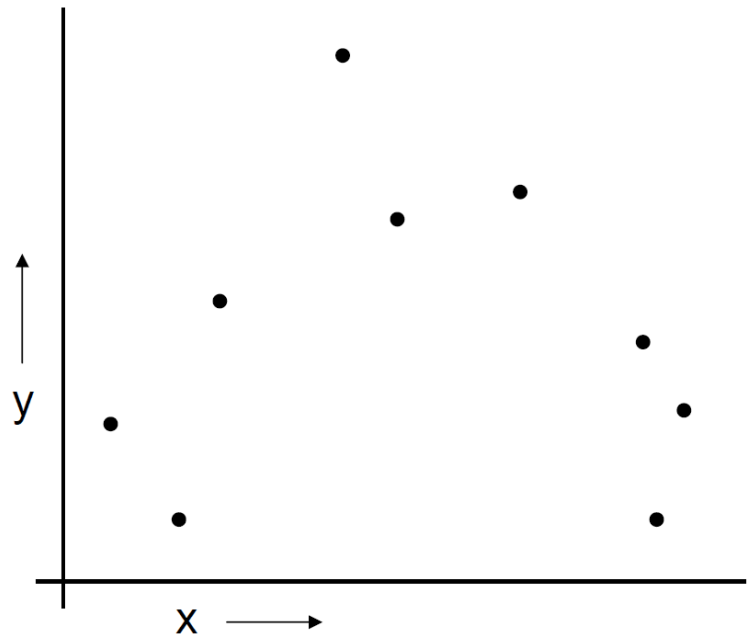
Crossvalidation / Leave – n - Out

crossval

- A very simple (but computationally complex) improvement on the hold-out
- Train the classifier on a set of *N-n* samples
  - Divide the dataset into blocks of *n samples*
  - Test the classifier on the *n* remaining samples
  - Repeat n/N times (dependent on subsampling) rotating through data
  - Report average performance on the repeated experiments

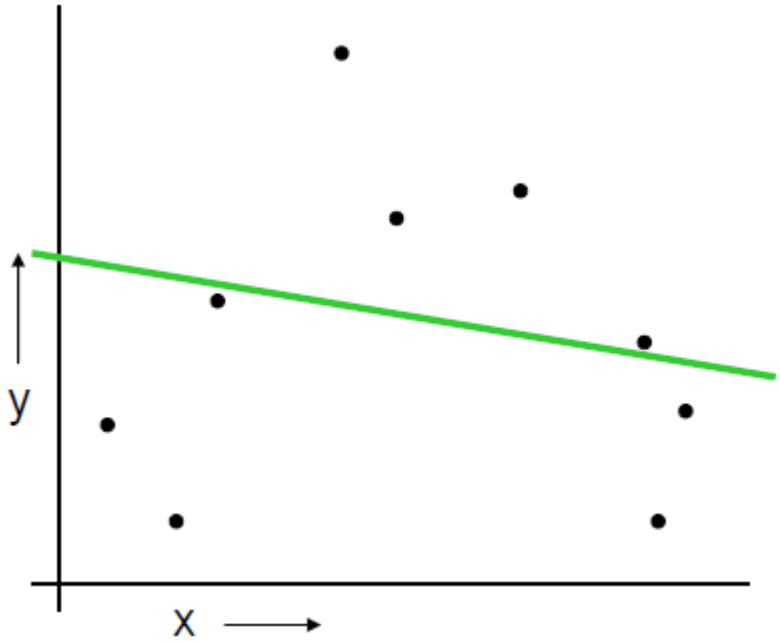


CV example: A regression problem

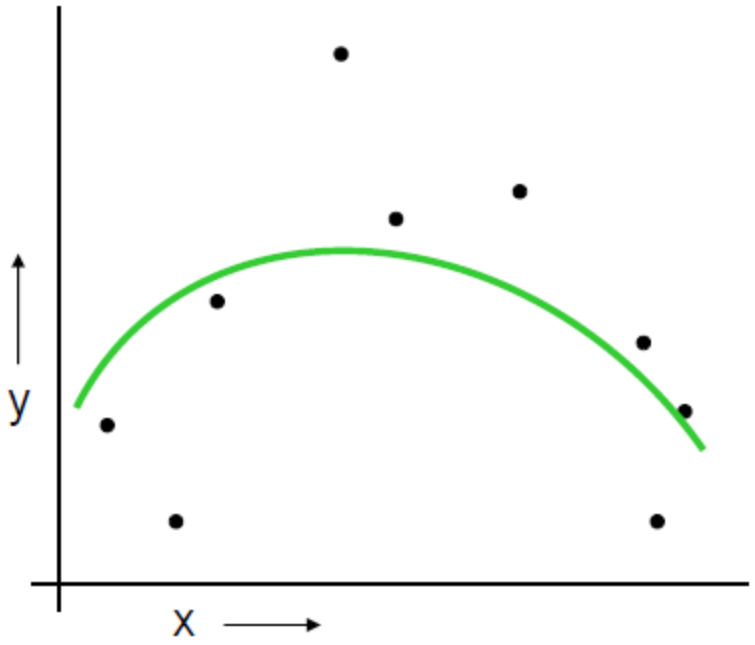


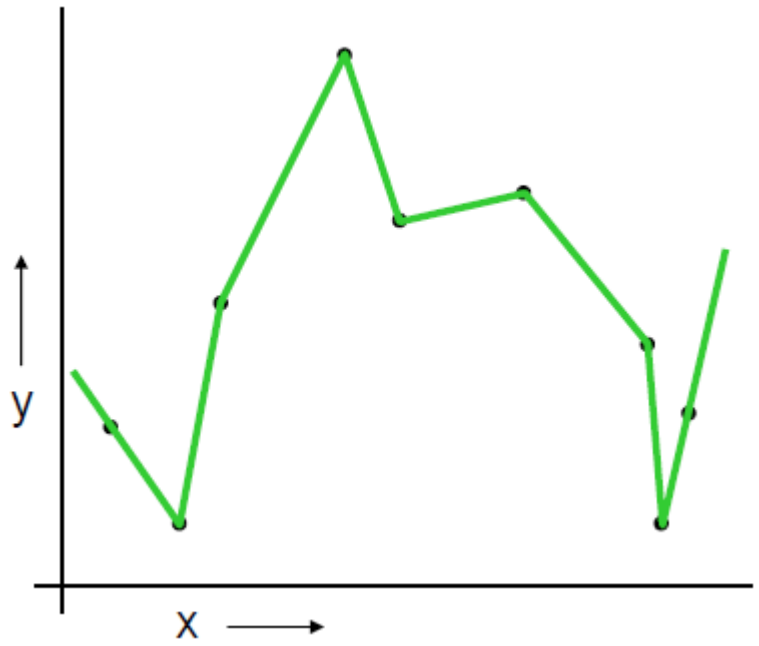
- $y = f(x) + noise$
- Can we learn f from the data?
- Consider three models
  - Linear
  - Cubic
  - «Connect-the-dots» (piecewise linear)

Linear regression

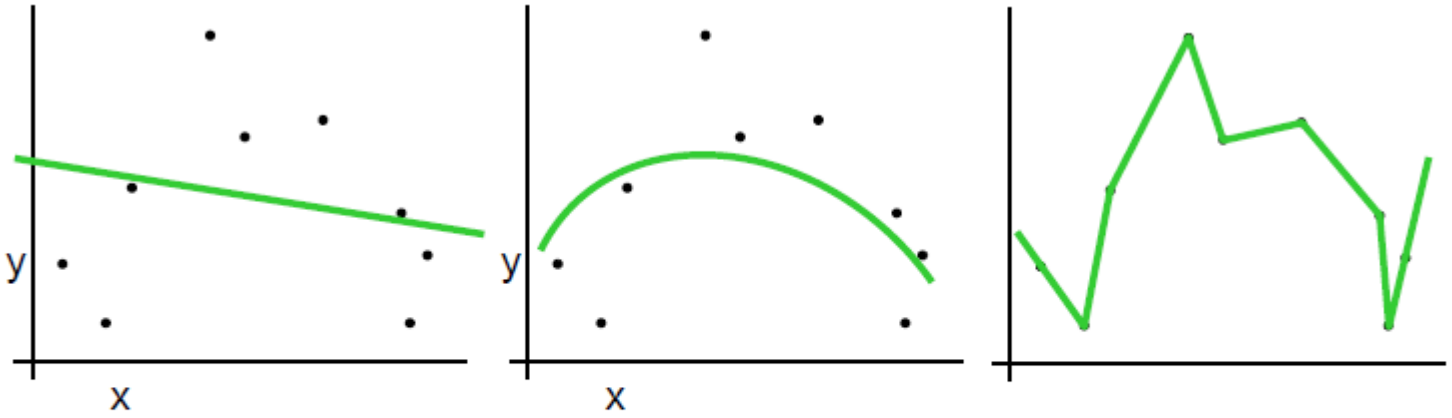


Cubic model



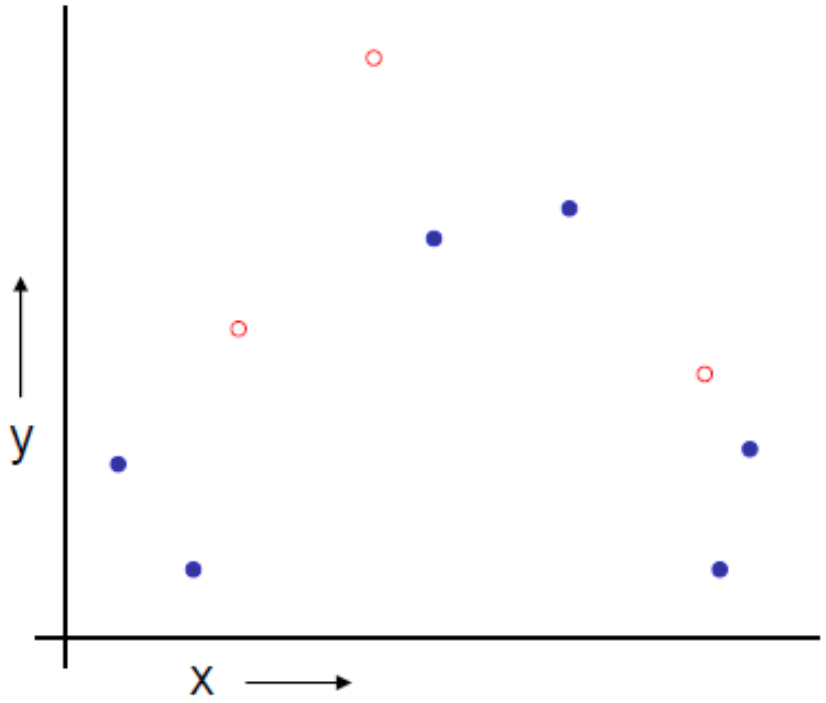

y
x

Which model is best



Why not choose the method with the best fit to the data?

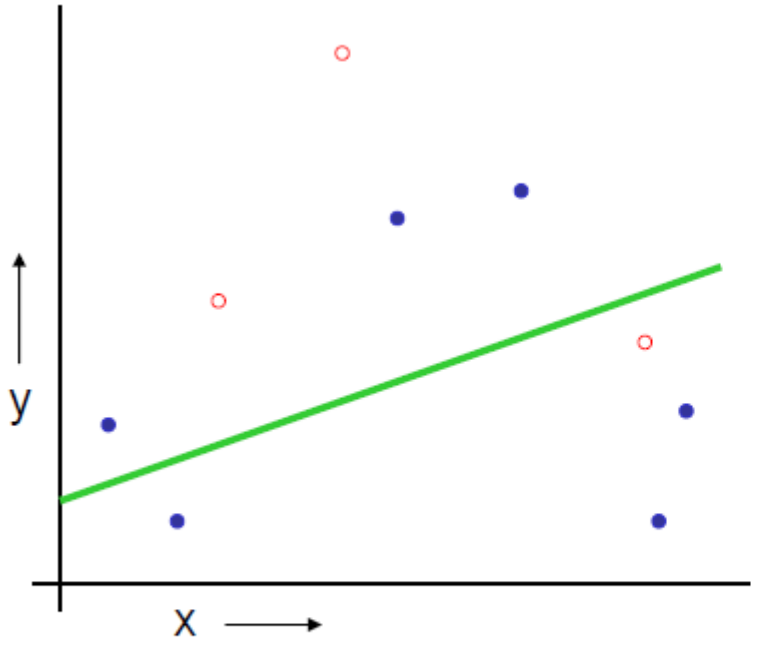
The test set method



1. Randomly choose 30% of the data to be in a test set

2. The remainder is a training set

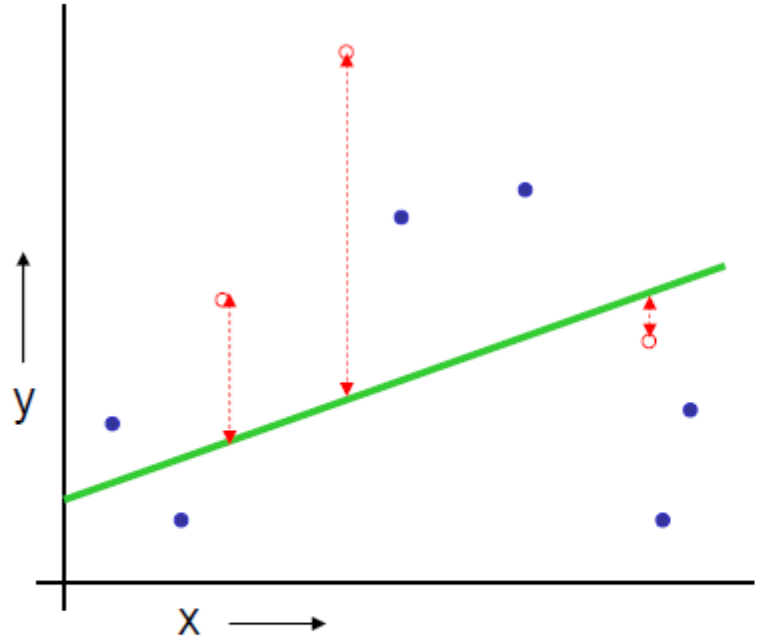
The test set method



(Linear regression example)

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set

The test set method



(Linear regression example)

Mean Squared Error = 2.4

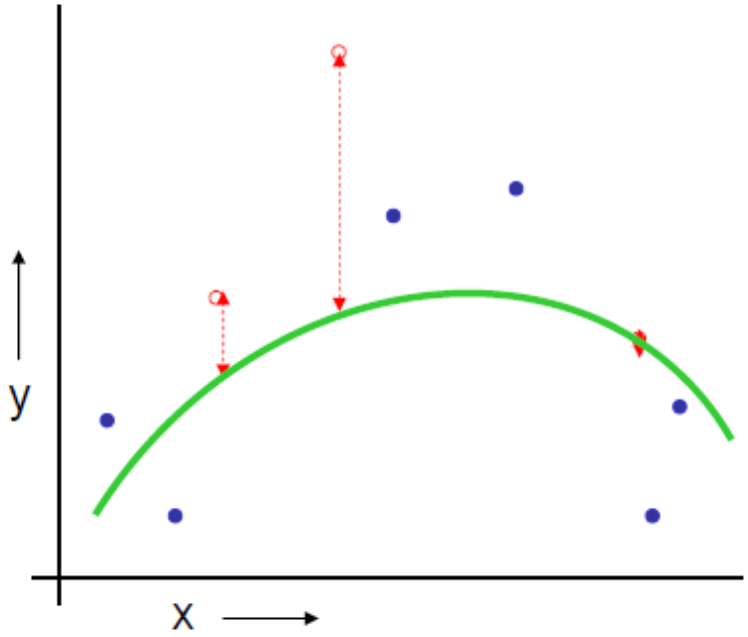
1. Randomly choose 30% of the data to be in a test set

2. The remainder is a training set

3. Perform your regression on the training set

4. Estimate your future performance with the test set

The test set method



(Quadratic regression example)

Mean Squared Error = 0.9

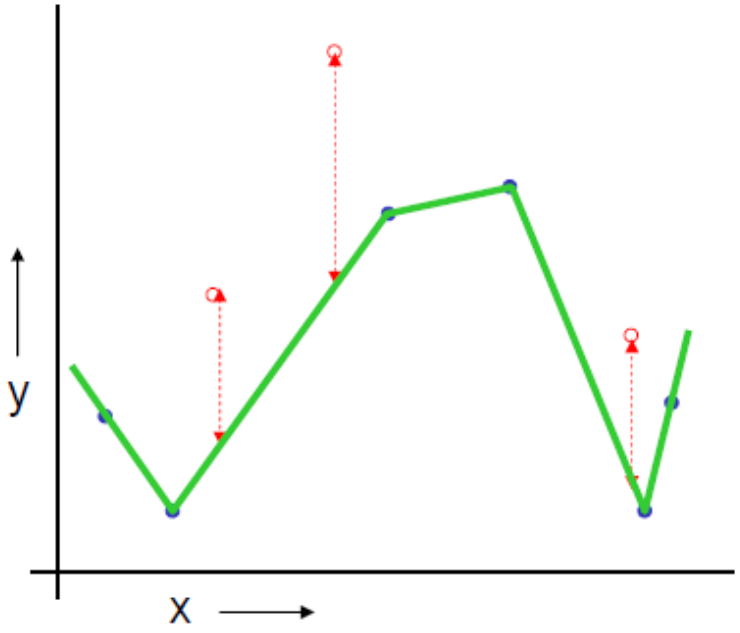
1. Randomly choose 30% of the data to be in a test set

2. The remainder is a training set

3. Perform your regression on the training set

4. Estimate your future performance with the test set

The test set method



(Join the dots example)

Mean Squared Error = 2.2

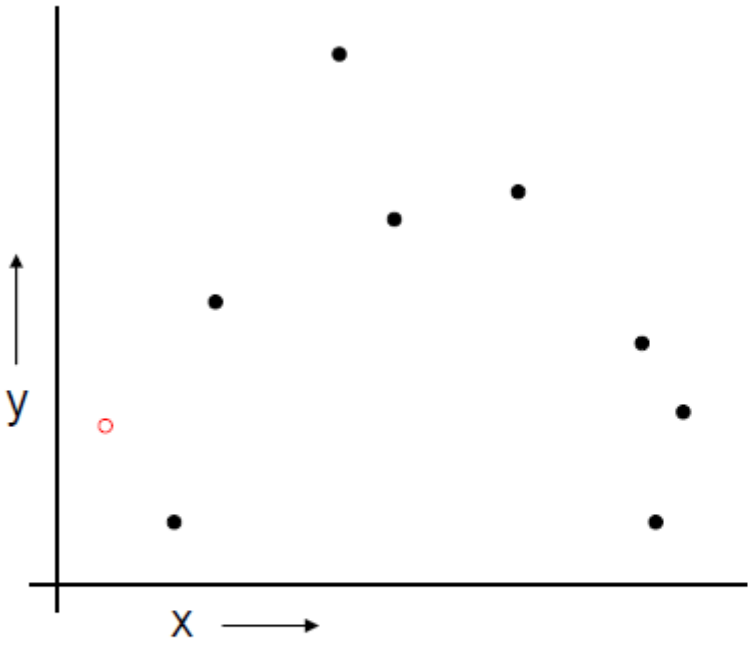
1. Randomly choose 30% of the data to be in a test set

2. The remainder is a training set

3. Perform your regression on the training set

4. Estimate your future performance with the test set

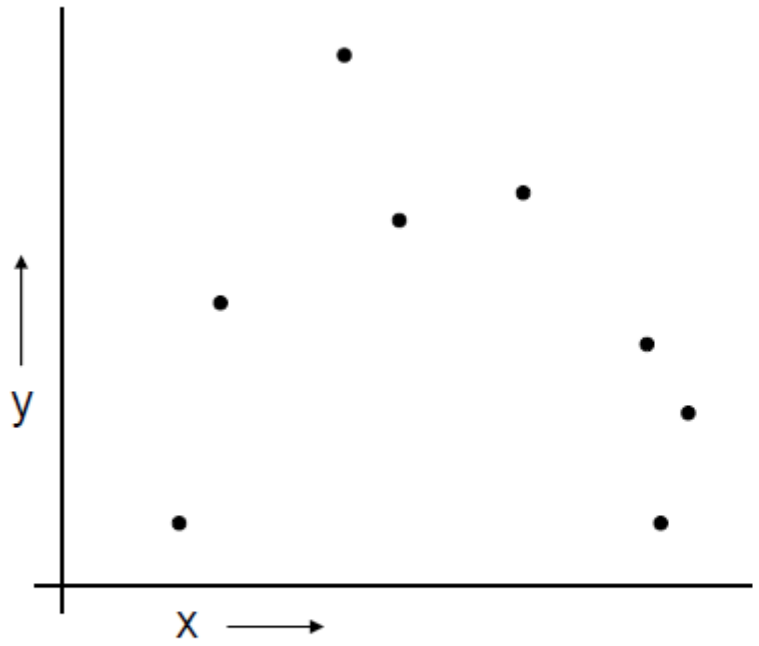
LOOCV (Leave-one-out-Crossvalidation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record

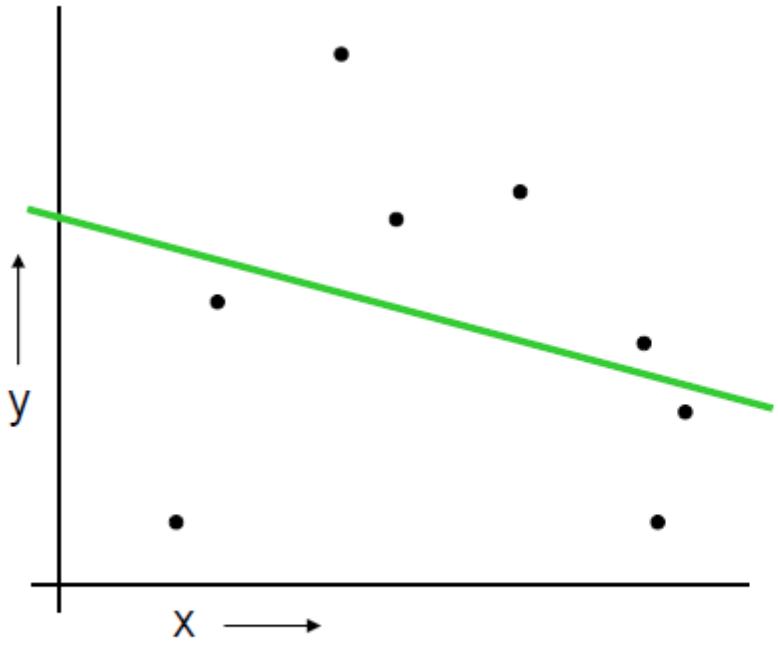
LOOCV (Leave-one-out-Crossvalidation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset

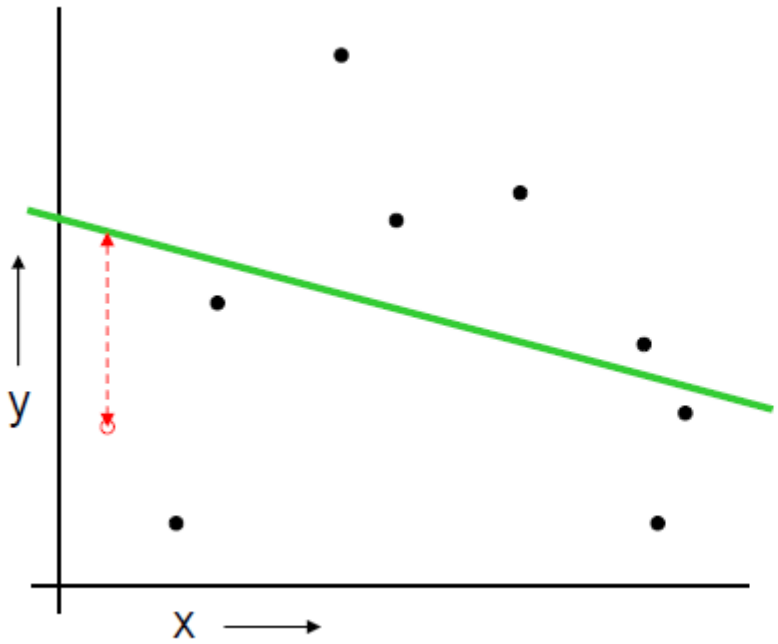
LOOCV (Leave-one-out-Crossvalidation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints

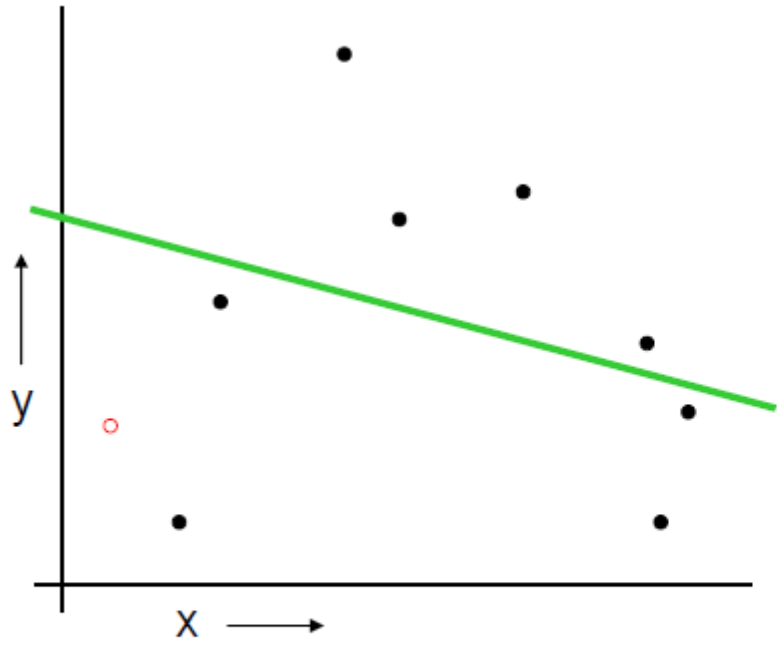
LOOCV (Leave-one-out-Crossvalidation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

LOOCV (Leave-one-out-Crossvalidation)

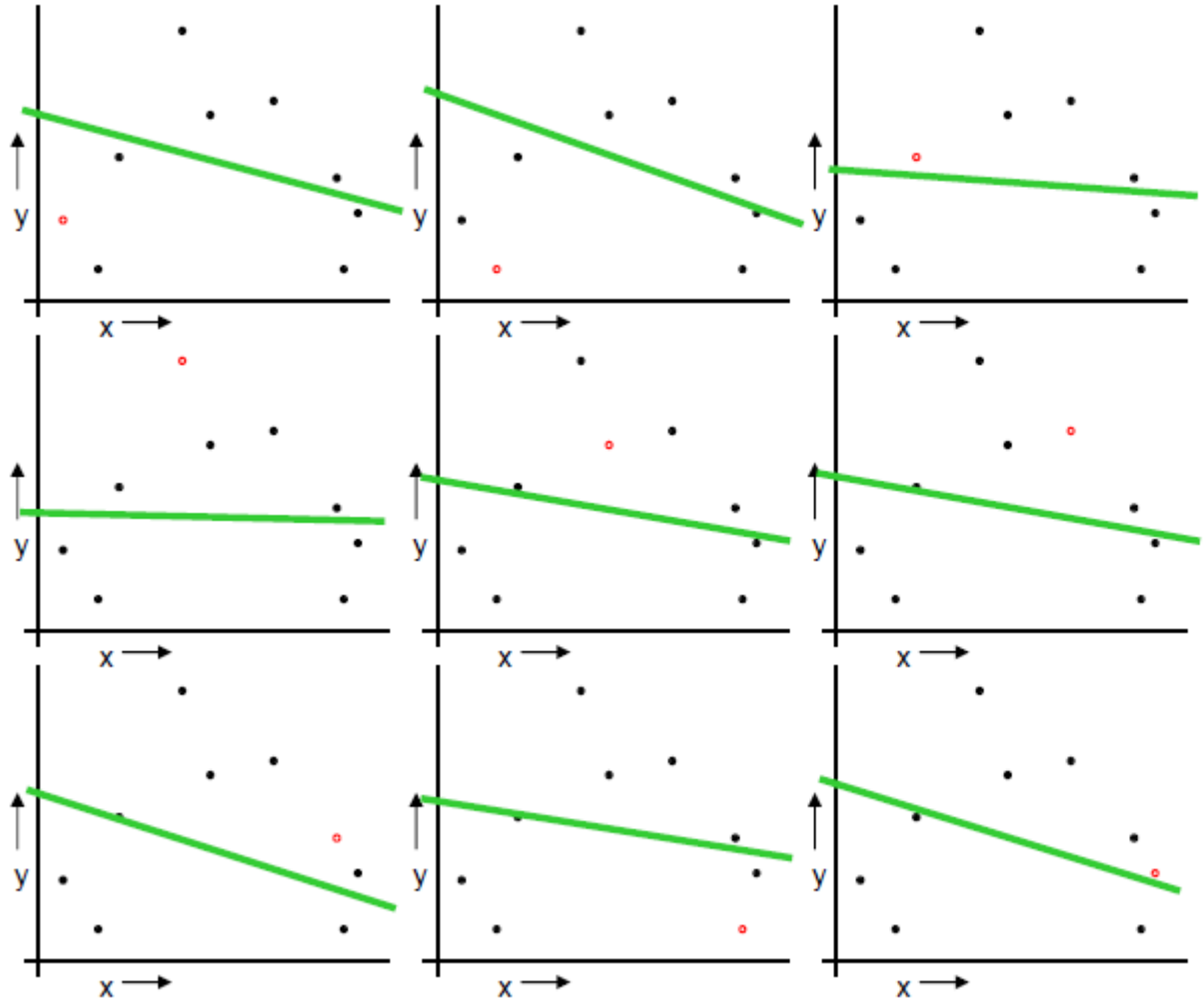


For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

LOOCV (Leave-one-out-Crossvalidation)



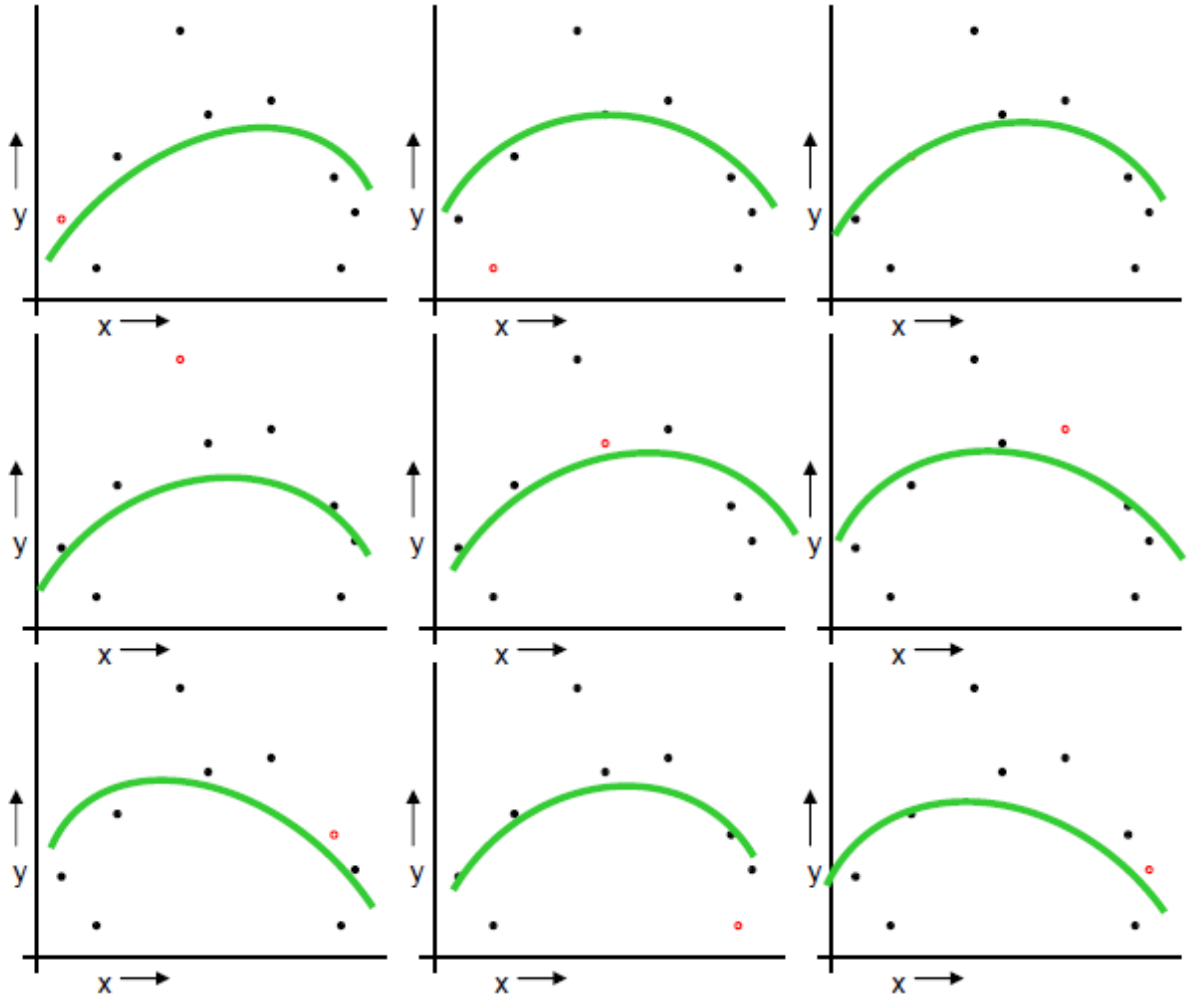
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 2.12$$

LOOCV (Leave-one-out-Crossvalidation)



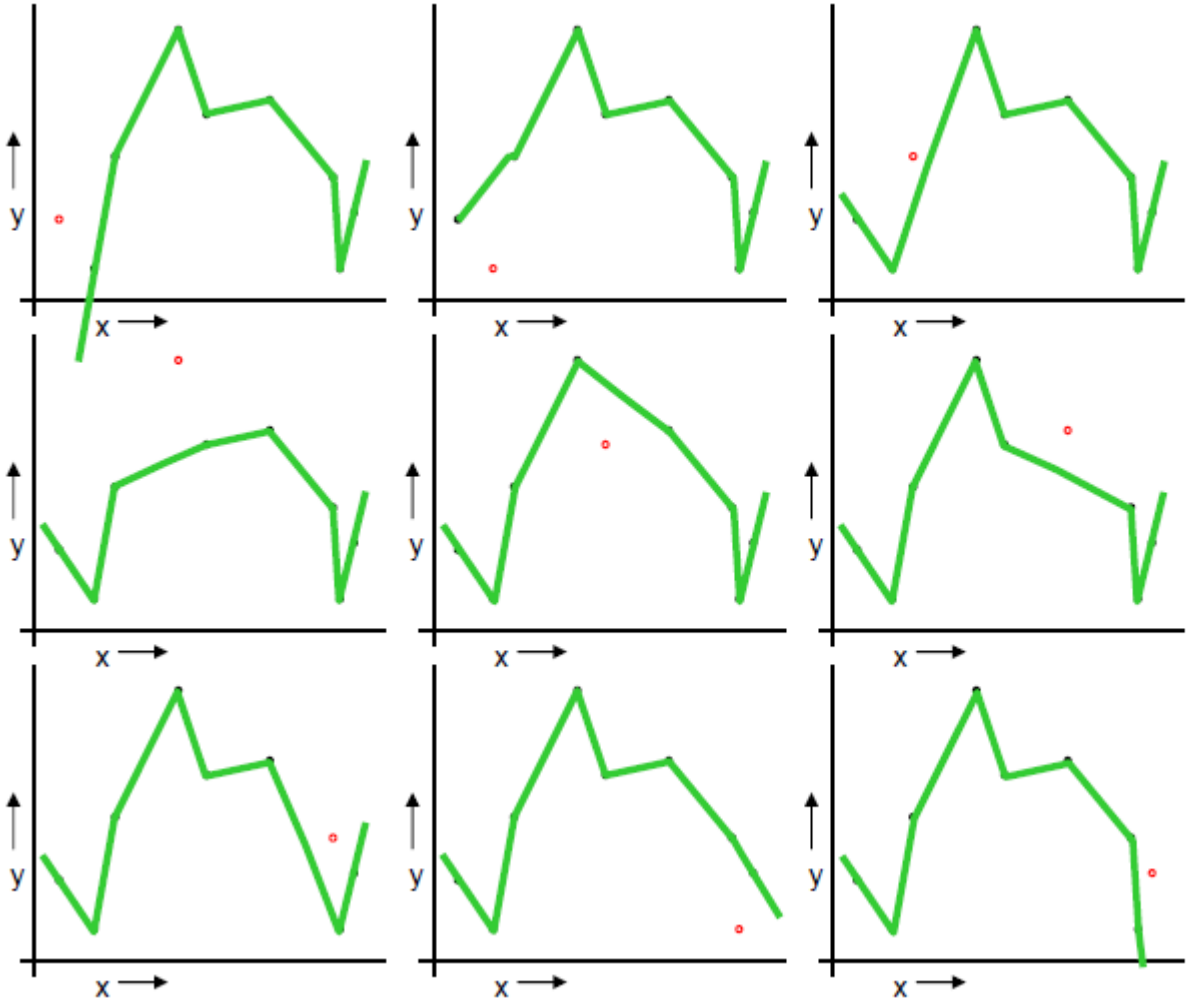
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$MSE_{LOOCV} = 0.962$

LOOCV (Leave-one-out-Crossvalidation)



For k=1 to R

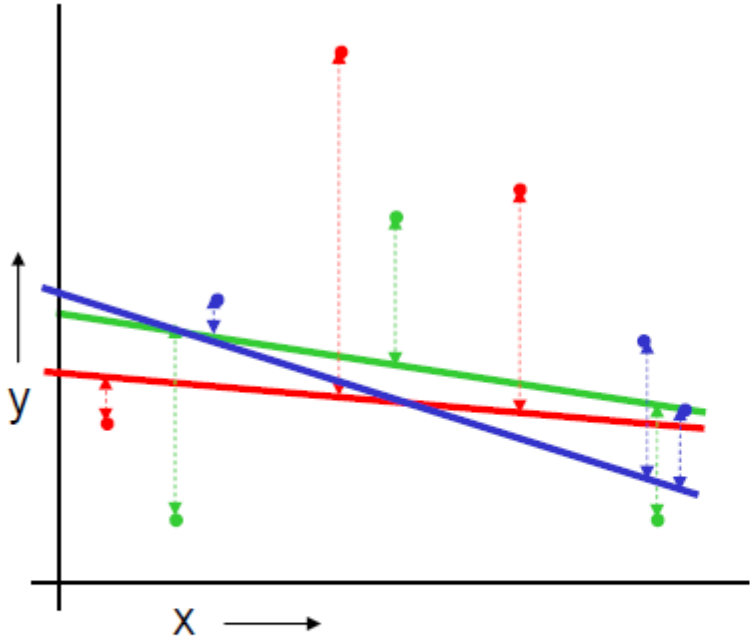
1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$MSE_{LOOCV} = 3.33$

k-fold Crossvalidation

- Randomly break the dataset into k partitions (in our example we'll have $k = 3$ partitions colored red,green and blue)



Linear Regression
MSE_{3FOLD}=2.05

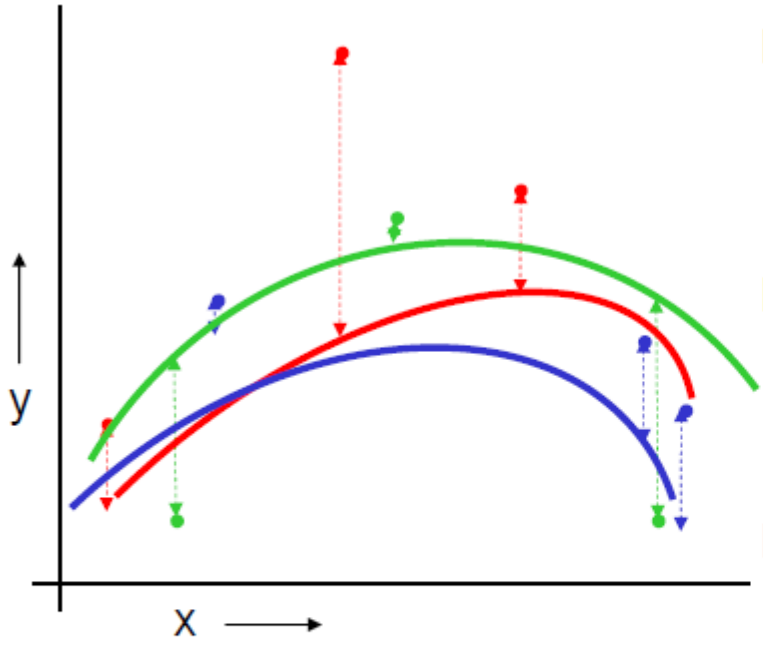
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

k-fold Crossvalidation



Quadratic Regression

$MSE_{3FOLD}=1.11$

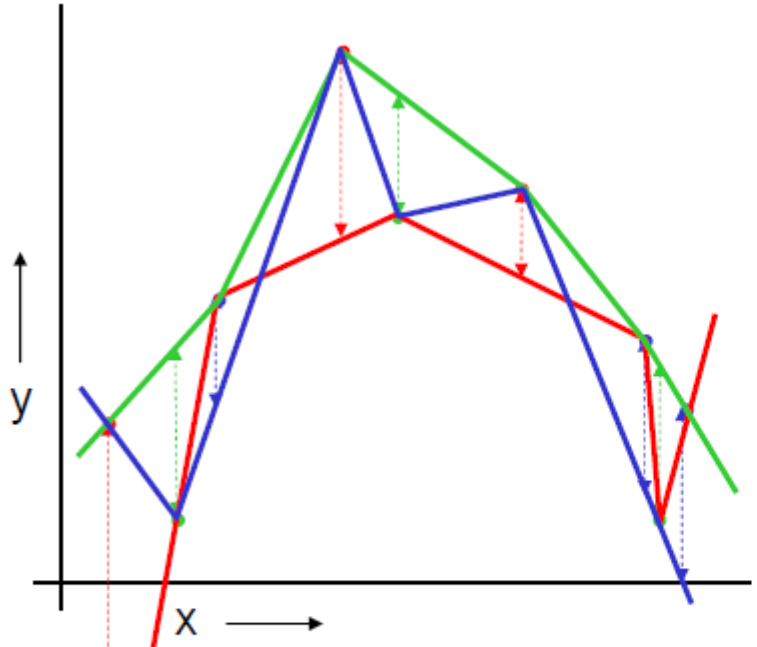
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

k-fold Crossvalidation



Joint-the-dots

$MSE_{3FOLD}=2.93$

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

Crossvalidation / Leave – n - Out

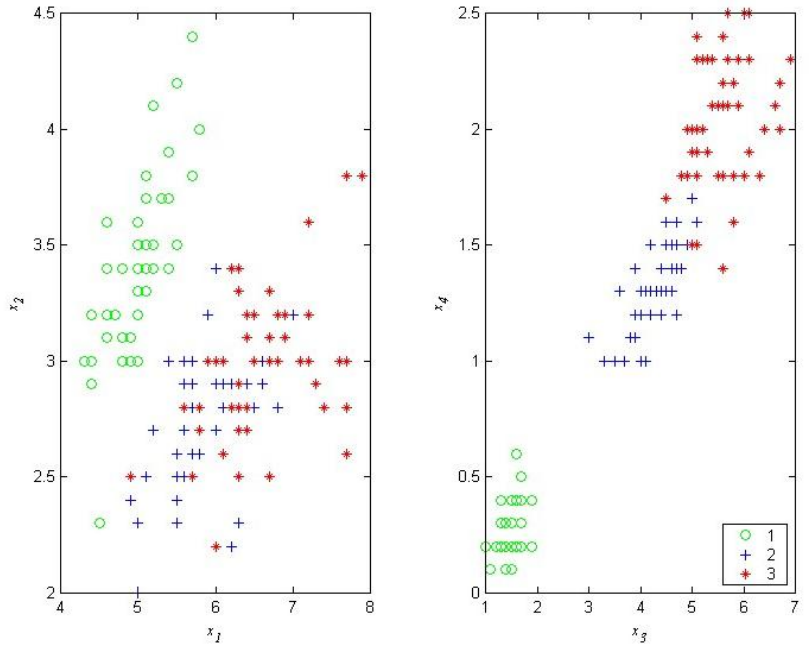
crossval

- How many blocks to divide the data in?
  - More is usually better, but trade-off with computational complexity
  - Usually five or ten blocks is used, often denoted 5-CV, 10-CV
  - Average, and spread, of classification error can be reported
  - Can be designed to guarantee samples from each class. (Stratification)
- The logical extreme of crossvalidation is to leave only *one* sample out each repetition
  - Extremely time consuming
  - Since all samples are visited once, no bias from random subsampling
  - Stratification impossible

Exploratory data analysis

scatterdui

- For a small number of features, manual data analysis to study the features is recommended.
- Choose intelligent features.
- Evaluate e.g.
  - Error rates for single-feature classification
  - Scatter plots



Scatter plots of feature combinations

What are *good* features?

- Clearly, we need to choose *good* features
- How do we quantify feature quality?
  - A good feature is simple to ”learn”
  - This is often related to class separation

Class separation

- Measure distance between all points or just class means?
- Many distance measures are ”pairwise”
  - Use average or minimum?
- All these distance measures can be represented as a scalar *J*, also called an ”objective function”

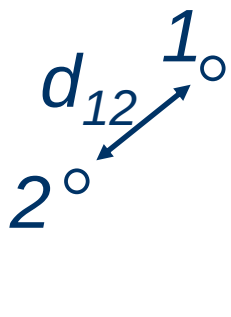
Typical class separation measures

- Euclidean distance
  - distance between pair of means
- Mahalanobis distance
  - sometimes called statistical distance
  - distance between pair of classes weighed by probability density
- Inter/intra class distance
  - Measure ratio of distance between class means and class ”size”
- Classifier accuracy
  - How good does a classifier perform on the dataset?
  - Evaluate with hold-out or cross-validation

Distance matrices

feateval

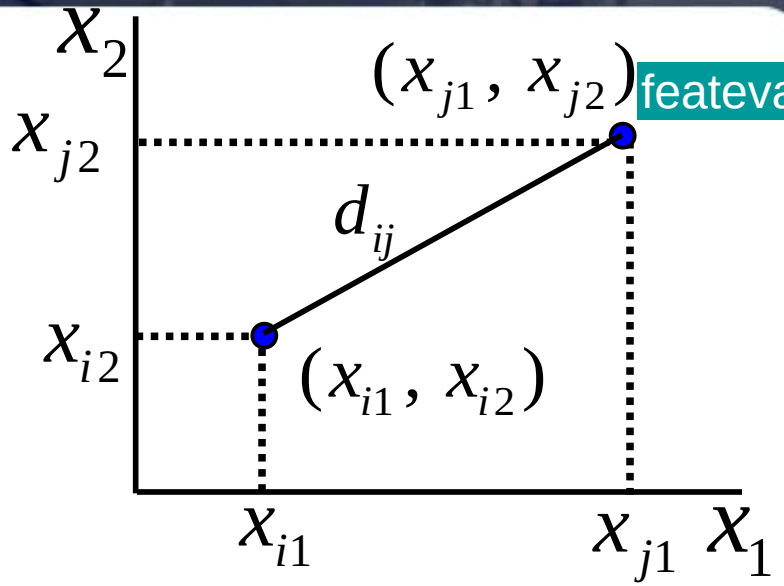
- Once a distance measure is defined, we can calculate the distance between objects. These objects could be individual observations, groups of observations (classes)
- For *N* objects, we then have a symmetric distance matrix D whose elements are distances between objects *i* and *j*.



$$\mathbf{D} = \begin{bmatrix} 0 & d_{12} & d_{13} \\ d_{21} & 0 & d_{23} \\ d_{31} & d_{32} & 0 \end{bmatrix}$$

Euclidean distance

- A possible distance measure for spaces equipped with a Euclidean metric
- For two dimensions (variables), this is just the hypotenuse of a right-angle triangle…
- …while for *p* dimensions, it is the hypotenuse of a hyper-triangle.

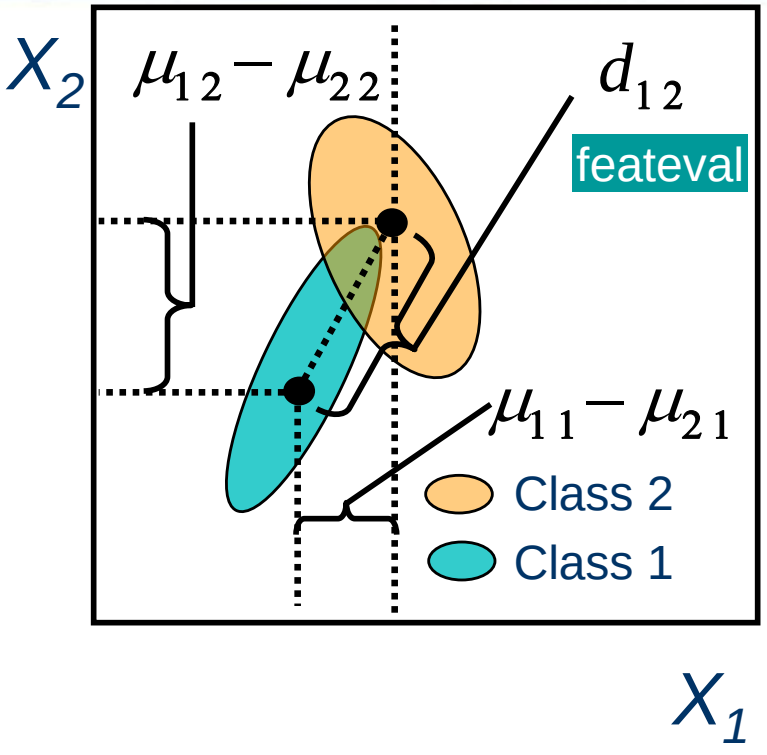


$$d_{ij} = \sqrt{\sum_{k=1}^{2} \left(x_{ik} - x_{jk} \right)^2}$$

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \left(x_{ik} - x_{jk} \right)^2}$$

Multivariate distances between classes: the Euclidean distance

- Calculates the Euclidean distance between two "points" defined by the multivariate means of two classes of *p* variables.
- Does not take into account differences among classes in within-class variability nor correlations among variables.



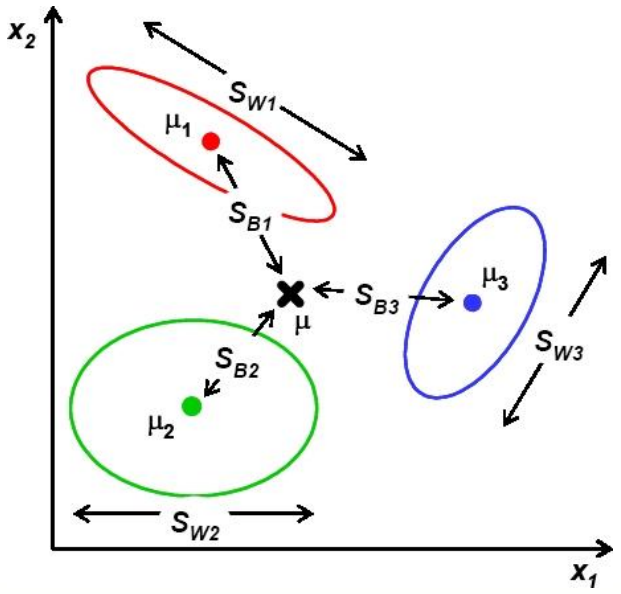
$$J = d_{ij} = \sqrt{\sum_{k=1}^{p} \left(\mu_{ik} - \mu_{jk}\right)^2}$$

Inter/intra class distance

featureval

- A simple measure of class separation is inter/intra class distance
  - Assumptions
    - discriminative information in mean differences
    - class scatter distribution similar for all classes

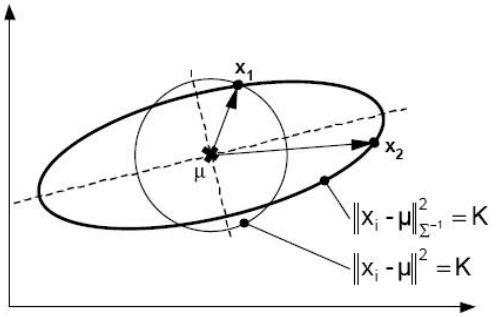
$$J_{\text{inter/intra}} = tr\{S_b S_w^{-1}\}$$



Mahalanobis distance

featureval

- Similar to inter/intra is a distance measure based on the Gaussian distribution
  - Assumptions
    - weigh mean distance by covariance estimate
    - pooled covariance estimate
    - Natural extension allowing different covariances (Bhattacharyya distance)



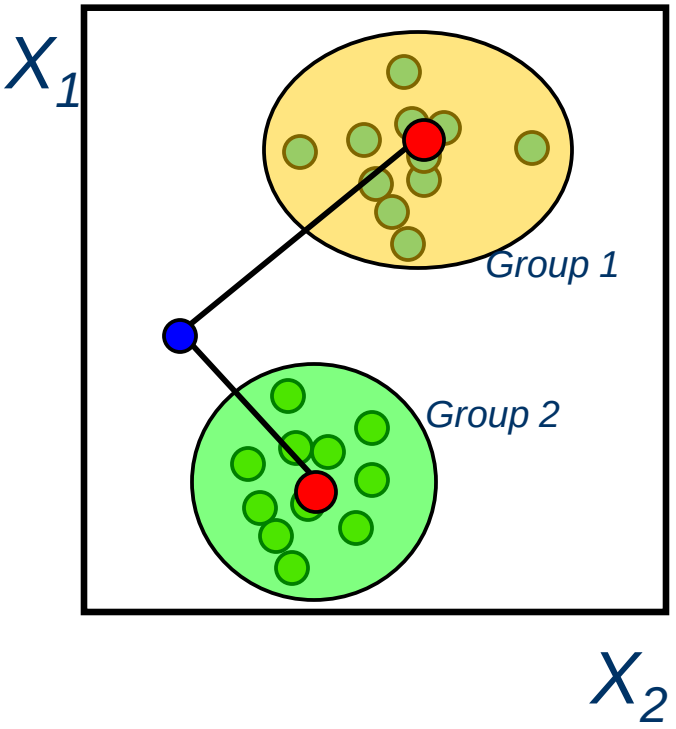
$$J_{\text{Mahalanobis}} = (\mu_1 - \mu_2)^T \Sigma^{-1} (\mu_1 - \mu_2)$$

$$\Sigma = N_1 \Sigma_1 + N_2 \Sigma_2$$

Distances between observations and objects

feateval

- We can also calculate a distance between an individual observation and some object, where the object may be another observation or a group mean.
- The distance between an observation and a group can be used to define the probability that the observation belongs to the group (f.ex. when using the Mahalanobis distance)



Feature selection

featselm

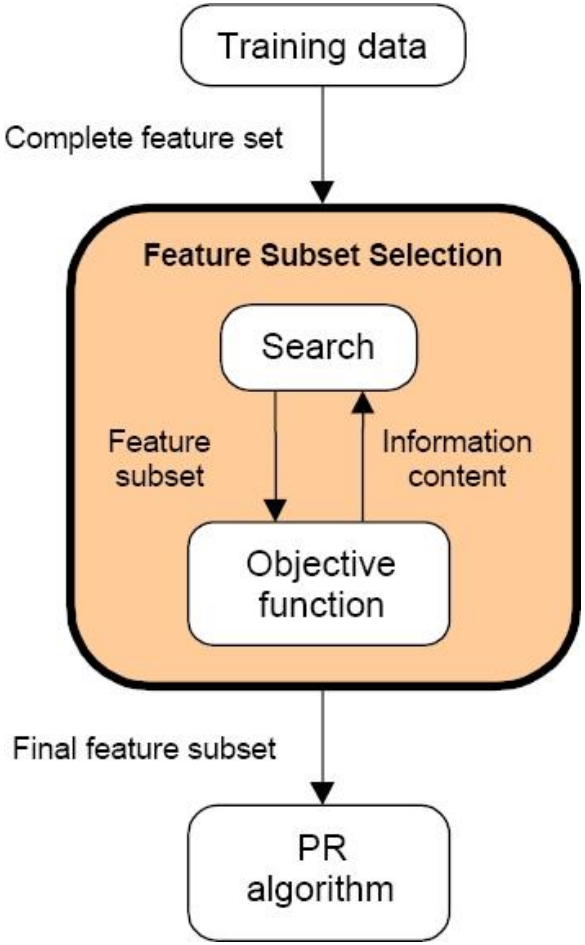
- Given a feature set $x=\{x_1, x_2,\ldots,x_n\}$ find a subset $y_m=\{xi_1,x_{i2},\ldots,x_{im}\}$ with $m<n$ which optimizes an objective function $J(Y)$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_m} \end{bmatrix}, x_{i1}, x_{i2}, \ldots, x_{im} = \underset{M, i_m}{\text{argmax}}[J(x_1, x_2, \ldots, x_n)]$$

Feature selection

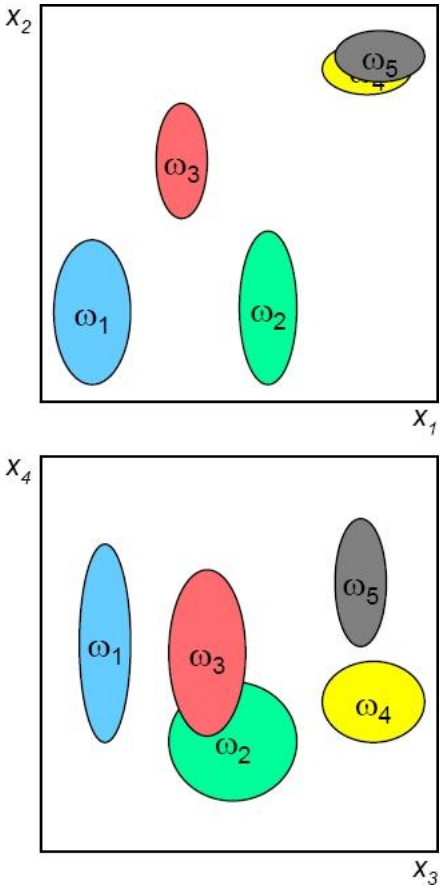
featselm

- Search strategy
  - Exhaustive search implies $\binom{n}{m}$ if we fix *m* and 2^n if we need to search all possible *m* as well.
  - Choosing *10* out of *100* will result in 10^{13} queries to *J*
  - Obviously we need to guide the search!
- Objective function (J)
  - ”Predict” classifier performance
  - ”Predicting” is faster than actual classification



Naïve feature search (individual selection)

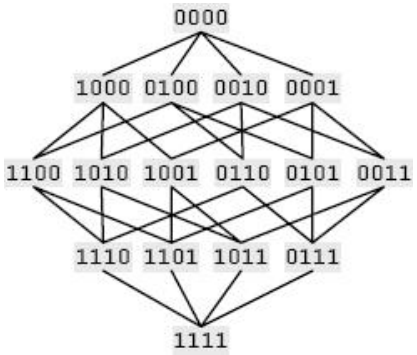
- Goal: select the two best features individually
- Easy to devise a breakdown case
  - Any reasonable objective *J* will rank the features $J(x_1)>J(x_2)\approx J(x_3)>J(x_4)$
  - Features chosen will be $[x_1,x_2]$ or $[x_1,x_2]$
  - However – the only feature that provides *complementary* information to x_1 is x_4
- Search is ”too greedy”
  - We need to compare choice with reference to *already chosen* features



Forward feature selection

featself

- Starting from the empty set, sequentially add the feature x^+ that results in the highest objective function $J(Y_k + x^+)$ when combined with the features Y_k that have already been selected



- Algorithm
  1. Start with the empty set $Y_0 = \emptyset$;
  2. Select the next best feature $x^+ = \arg\max_{x \notin Y_k} J(Y_k + x)$
  3. Update $Y_{k+1} = Y_k + x^+$; $k = k + 1$
  4. If k less than number of features wanted goto 2

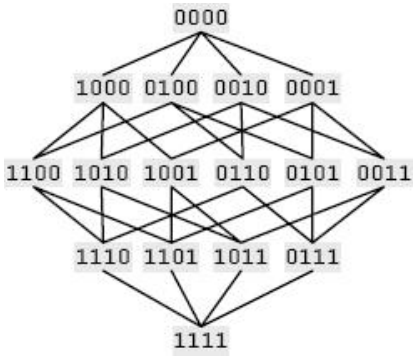
- Forward selection performs best when the optimal subset has a small number of features
- Forward selection cannot discard features that become obsolete when adding other features

Backward feature selection

featselb

- Starting from the full set, sequentially remove the feature x^- that results in the smallest decrease in objective function $J(Y_k - x^-)$ when combined with the features Y_k that are already in the set

- Algorithm
  1. Start with the full set $Y_k = \underline{X}$;
  2. Remove the worst feature $x^- = \arg\min_{x \in Y_k} J(Y_k - x)$
  3. Update $Y_{k-1} = Y_k + x^-$; $k = k - 1$
  4. If k more than number of features wanted goto 2

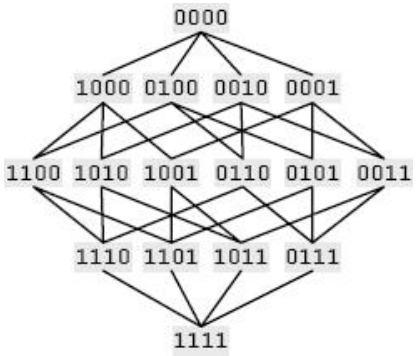


- Backward selection performs best when the optimal subset has a large number of features
- Backward selection cannot re-include features that become necessary when removing other features
- Note that the decrease can also be an increase

Floating search (Pudil's forward)

featselp

- Starting from the empty set, include features by forward search, then backtrack using backward search until criterion decreases

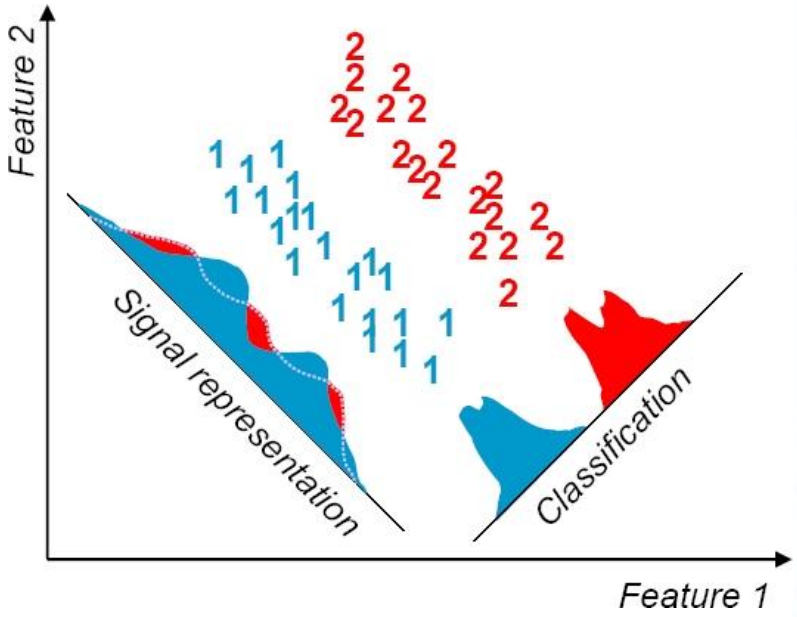


- Algorithm
  1. Start with the empty set $Y_0 = \underline{\varnothing}$;
  2. Do a forward step;
     $Y_{k+1} = Y_k + x^+;\ k = k + 1$
  3. While we can increase criterion *J;* do backward step
     $Y_{k-1} = Y_k + x^-;\ k = k - 1$
  3. If *k* less than number of features wanted goto 2

- Can be extremely time-consuming
- The improvement over other methods somewhat dependent on the feature set

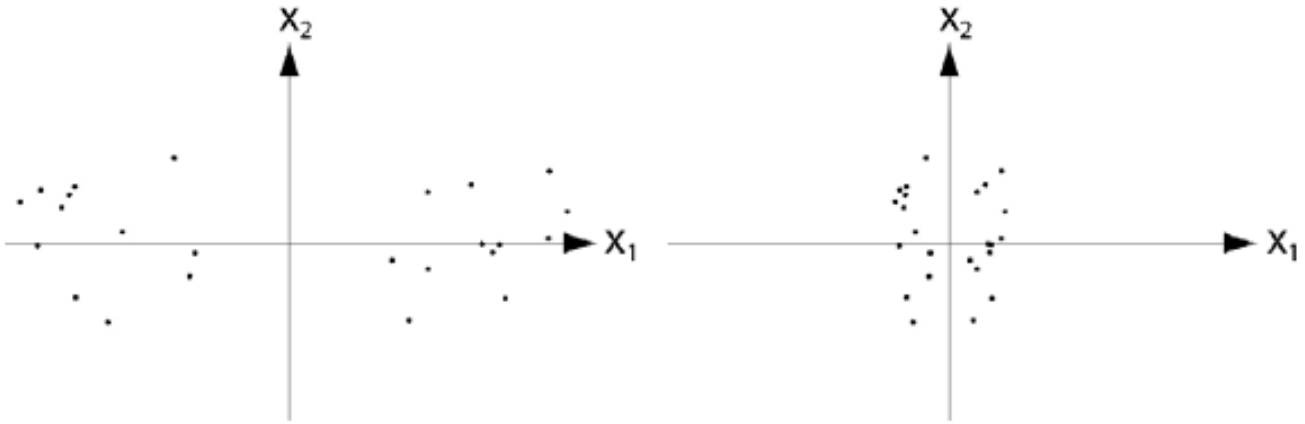
Feature selection as dimension reduction

- In some cases, a linear (or nonlinear combination) of features might be a better choice than using a subset of features
  - Consider however, that not all transforms are appropriate for dimension reduction for classification
- However, feature selection has one interesting property – we represent the data on a set of dimensions that retain their *meaning*



Using distance as a criterion

- It might be tempting to rescale the features
- Seems reasonable to make features scale-invariant?
- For example, scale the data cloud to zero mean and unit variance
- When using euclidean distance as a criterion this might change the clustering result, which one is the one we want?
  - Rescaling is not always a good idea, but should be considered if Euclidean distance is used

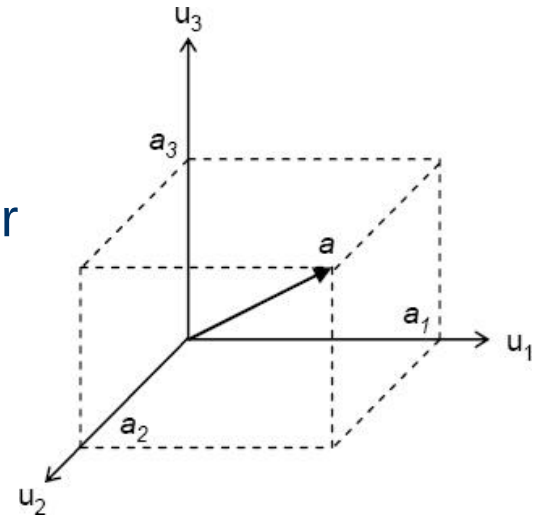


Two approaches to dimensionality reduction

Feature extraction: create a subset of new features by combinations
of the existing features
Feature selection: choose a subset of all the features (the more
informative ones)

$$\underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \longrightarrow \begin{pmatrix} x_{i_1} \\ x_{i_2} \\ \\ x_{i_M} \end{pmatrix}}_{\text{feature selection}}, \qquad \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \longrightarrow \begin{pmatrix} y_1 \\ y_2 \\ \\ y_M \end{pmatrix} = f\left(\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}\right)}_{\text{feature extraction}}$$

Vector spaces



- A set of vectors $u_1, u_2, \ldots, u_n$ is a basis for a vector space if any arbitrary vector x can be represented by a linear combination

  $x = a_1u_1 + a_2u_2 + \ldots + a_nu_n$
- The coefficients $a_1, a_2, \ldots, a_n$ are called the components of vector x with respect to the basis u_i
- In order to form a basis, it is necessary and sufficient that the u_i vectors be linearly independent
- A basis u_i is said to be orthogonal if $u_i^T u_j = \begin{cases} \neq 0, i = j \\ = 0, i \neq j \end{cases}$
- A basis u_i is said to be orthonormal if $u_i^T u_j = \begin{cases} = 1, i = j \\ = 0, i \neq j \end{cases}$

Linear transformation

- A linear transformation is a mapping from a vector space X^N onto a vector space Y^M, and is represented by a matrix
- Given a vector $x \epsilon X^N$, the corresponding vector y on Y^M is

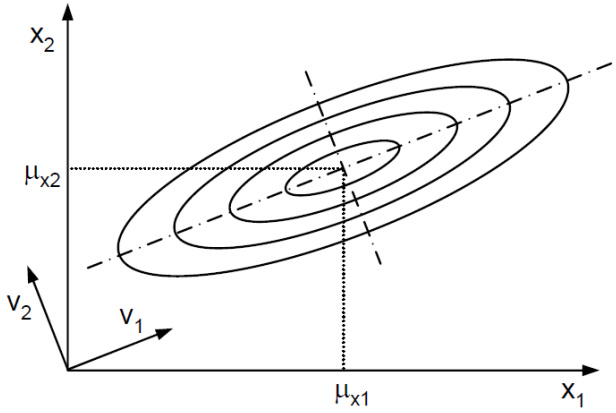
$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{11} & \cdots & a_{11} \\ a_{11} & a_{11} & \cdots & a_{11} \\ \vdots & \vdots & \ddots & \vdots \\ a_{11} & a_{11} & \cdots & a_{11} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}$$

Eigenvectors and eigenvalues

- Given a matrix $\boldsymbol{A}_{N\times N}$, we say that $\boldsymbol{v}$ is an eigenvector if there exists a scalar $\boldsymbol{\lambda}$ (the eigenvalue) such that $\boldsymbol{Av} = \boldsymbol{\lambda v} \Leftrightarrow \boldsymbol{v}$ is an eigenvector with corresponding eigenvalue $\boldsymbol{\lambda}$

$$\mathrm{Av} = \lambda \mathrm{v} \Rightarrow (\mathrm{A} - \lambda)\mathrm{v} = 0 \Rightarrow$$

$$|(\mathrm{A} - \lambda \mathrm{I})|\,\mathrm{v} = 0 \Rightarrow \underbrace{\lambda^N + a_1\lambda^N + \ldots + a_{N-1}\lambda^N + a_0 = 0}_{\text{Characteristic equation}}$$



- Zeroes of the characteristic equation are the eigenvalues of $\boldsymbol{A}$
- $\boldsymbol{A}$ is non-singular $\Leftrightarrow$ all eigenvalues are non-zero
- $\boldsymbol{A}$ is real and symmetric $\Leftrightarrow$ all eigenvalues are real, and eigenvectors are orthogonal

Interpretation of eigenvectors and eigenvalues

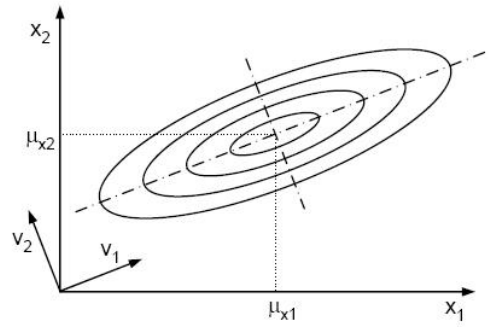
- The eigenvectors of the covariance matrix Σ correspond to the principal axes of equiprobability ellipses.
- The linear transformation defined by the eigenvectors Σ of leads to vectors that are uncorrelated regardless of the form of the distribution
- If the distribution happens to be Gaussian, then the transformed vectors will be statistically independent

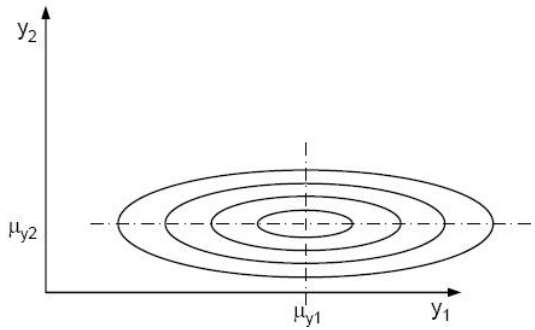
$$\Sigma M = M\Lambda \text{ with } M = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ v_1 & v_2 & \cdots & v_N \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \text{ and } \Lambda = \begin{bmatrix} \lambda_1 & & \\ & & \\ & & \lambda_N \end{bmatrix}$$

$$f_X(x) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)\right]$$

$$f_Y(y) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\lambda_i}} \exp\left[-\frac{(y_i-\mu_{y_i})^2}{2\lambda_i}\right]$$

$y = M^T x$





Dimensionality reduction

- Feature extraction can be stated as
  - Given a feature space ***xi*** $\in \mathbb{R}^n$ find an optimal mapping
  - ***y = f (x) :*** $\mathbb{R}^n \rightarrow \mathbb{R}^m$ with ***m < n***.
  - An optimal mapping in classification :the transformed feature vector ***y*** yield the same classification rate as ***x***.
- The optimal mapping may be a non-linear function
  - Difficult to generate/optimize non-linear transforms
  - Feature extraction is therefore usually limited to linear transforms ***y = A^{T}x***

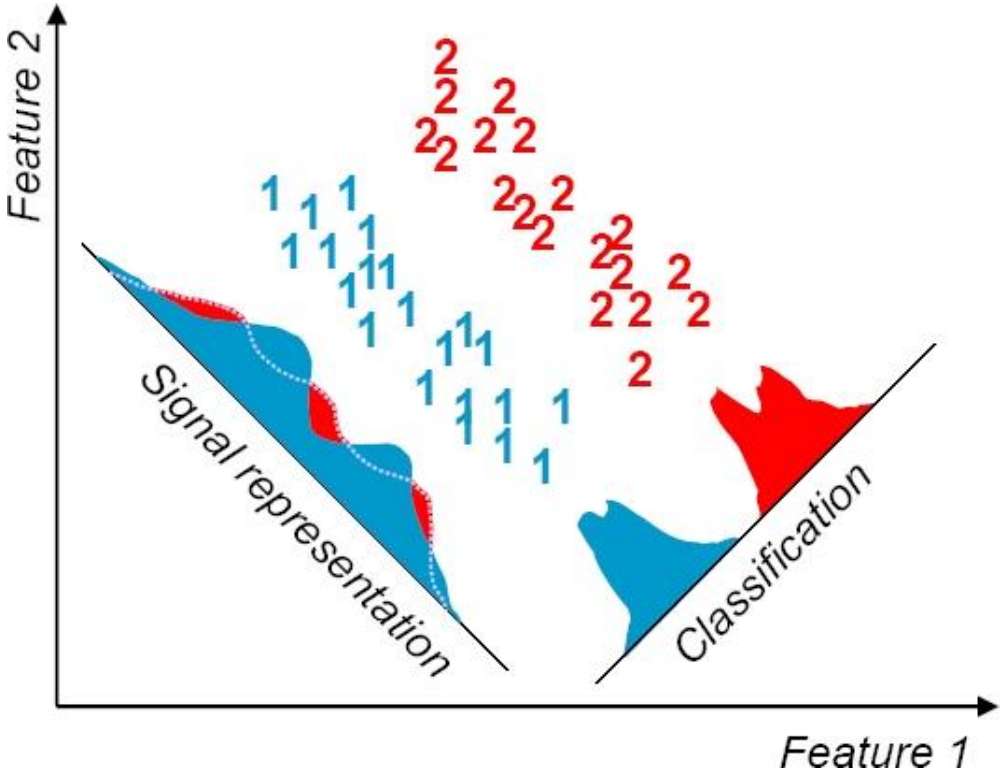
$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{11} & \cdots & a_{11} \\ a_{11} & a_{11} & \cdots & a_{11} \\ \vdots & \vdots & \ddots & \vdots \\ a_{11} & a_{11} & \cdots & a_{11} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}$$

Signal representation vs classification

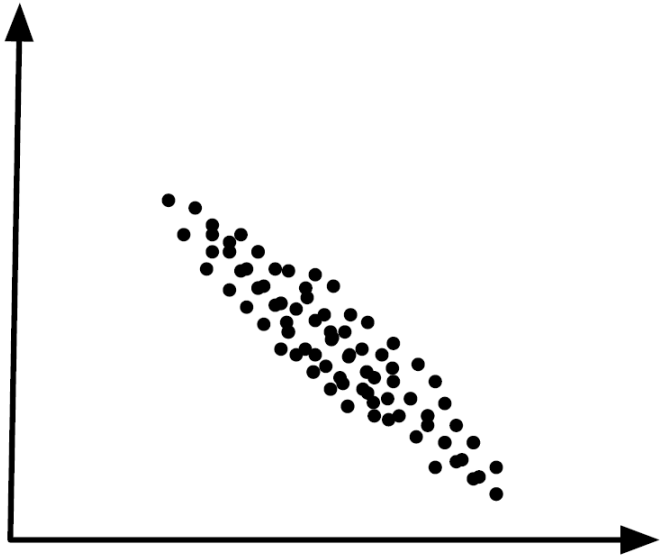
- The search for the feature extraction mapping $\boldsymbol{y} = \boldsymbol{f}\ (\boldsymbol{x})$ is guided by an objective function we want to maximize.
- In general we have two categories of objectives in feature extraction:
  - Signal representation: Accurately approximate the samples in a lower-dimensional space.
  - Classification: Keep or enhance class-discriminatory information in a lower-dimensional space.

Signal representation vs classification

- Dimensionality reduction via feature extraction.
- Principal Component Analysis (PCA), unsupervised
  - Emphasis is on representing the original signal as accurately as possible in the lower dimensional space.
- Linear Discriminant Analysis (LDA), supervised
  - Emphasis is to maximize the class-discrimination in the lower dimensional space

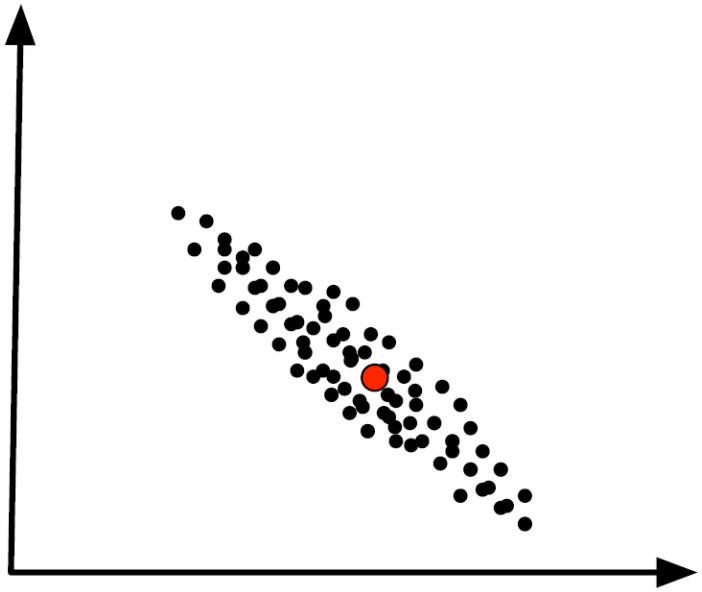


Intuitive motivation for PCA



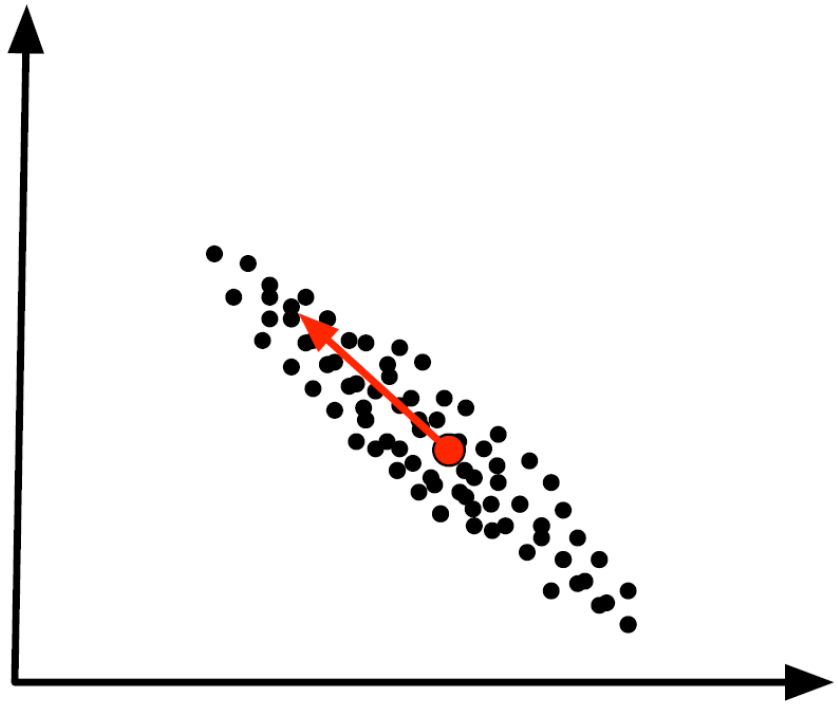
Say we want to encode as accurately as possible the position of the $\boldsymbol{m}$ points in this cluster. Can do so exactly with their $(\boldsymbol{x}, \boldsymbol{y})$-coordinate locations of which there are $\boldsymbol{2m}$. However, say we only have bandwidth to record $\boldsymbol{m} + \boldsymbol{4}$ numbers. Intuitively, what should these numbers represent ?

Intuitive motivation for PCA



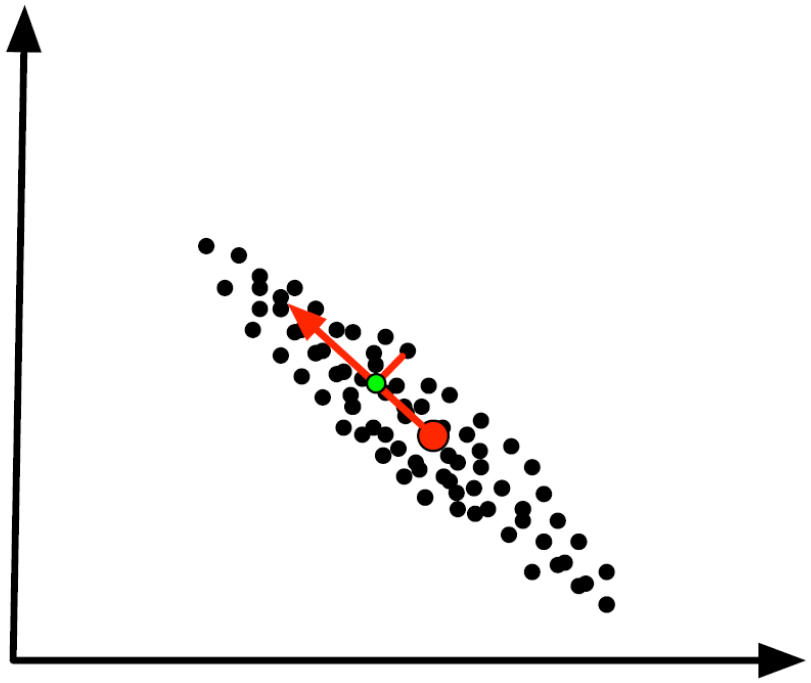
Use 2 to encode the center point

Intuitive motivation for PCA



Use 2 numbers to define a direction which corresponds to the
direction in which there is most variation.

Intuitive motivation for PCA



Let the other ***m*** numbers represent the distance of the point projected onto the line from the centre point.

PCA – Principal component analysis

- Reduce dimension while preserving signal variance (”randomness”)
- Represent ***x*** as a linear combination of orthonormal basis vectors $[\varphi_1 \perp \varphi_2 \perp \cdots \perp \varphi_n]$:

$$x = \sum_{i=0}^{n} y_i \, \varphi_i$$

- Approximate ***x*** with only ***m < n*** basis vectors. This can be done by replacing the components $[y_{m+1}, y_{m+2}, \ldots, y_n]^T$ with some pre-selected constants $\boldsymbol{b_i}$:

$$\hat{x}(m) = \sum_{i=0}^{m} y_i \, \varphi_i + \sum_{i=m+1}^{n} b_i \, \varphi_i$$

PCA – Principal component analysis

- Approximation error is then

$$\Delta x(m) = x - \hat{x}(m)$$

$$= \sum_{i=0}^{n} y_i \, \varphi_i - \sum_{i=0}^{m} y_i \, \varphi_i + \sum_{i=m+1}^{n} b_i \, \varphi_i = \sum_{i=m+1}^{n} (y_i - b_i) \, \varphi_i$$

- Measure the representation error by mean squared magnitude of Δx, to find the basis vectors φ_i and constants b_i that minimizes this error

$$MSE(m) = E[|\Delta x(m)|^2]$$

$$= E[\sum_{i=m+1}^{n} \sum_{j=m+1}^{n} (y_i - b_i)(y_j - b_j) \varphi_i^T \varphi_j] = \sum_{i=m+1}^{n} E[(y_i - b_i)^2]$$

PCA – Principal component analysis

- The optimal values of b_i can be found by computing the partial derivative of the objective function and setting it to zero

$$\frac{\partial}{\partial b_i} E[(y_i - b_i)^2] = -2(E[y_i] - b_i) = 0 \Rightarrow b_i = E[y_i]$$

  - Therefore, we will replace the discarded dimensions y_is by their expected value

- The mean-square error can then be written as

$$MSE(m) = \sum_{i=m+1}^{n} E[(y_i - E[y_i])^2]$$

$$= \sum_{i=m+1}^{n} E[(\phi_i{}^T x - E[\phi_i{}^T x])(\phi_i{}^T x - E[\phi_i{}^T x])]$$

$$= \sum_{i=m+1}^{n} \phi_i{}^T \underbrace{E[(x - E[x])\,(x - E[x])^T]}_{\text{Definition of covariance } \Sigma_x} \phi_i = \sum_{i=m+1}^{n} \phi_i{}^T \Sigma_x \phi_i$$

PCA – Principal component analysis

- We seek to find the solution that minimizes this expression subject to the orthonormality constraint, which we incorporate into the expression using a set of Lagrange multipliers λ_i:

$$MSE(m) = \sum_{i=m+1}^{n} \phi_i{}^T \Sigma_x \phi_i + \sum_{i=m+1}^{n} \lambda_i (1 - \phi_i{}^T \phi_i)$$

- Minimization of this criterion can also be done by partial derivation

$$\frac{\partial}{\partial \phi_i} MSE(m) = \frac{\partial}{\partial \phi_i} \left[\sum_{i=m+1}^{n} \phi_i{}^T \Sigma_x \phi_i + \sum_{i=m+1}^{n} \lambda_i (1 - \phi_i{}^T \phi_i) \right]$$

$$= 2(\Sigma_x \phi_i - \lambda_i \phi_i) = 0, \qquad \text{as } \frac{\partial x^T A x}{\partial x} = (A + A^T)x$$

- Thus $\Sigma_x \phi_i = \lambda_i \phi_i$ defines an eigenvalue problem

PCA – Principal component analysis

- We can then express the sum-square error as

$$MSE(m) = \sum_{i=m+1}^{n} {\phi_i}^T \Sigma_x \phi_i = \sum_{i=m+1}^{n} {\phi_i}^T \lambda_i \phi_i = \sum_{i=m+1}^{n} \lambda_i$$

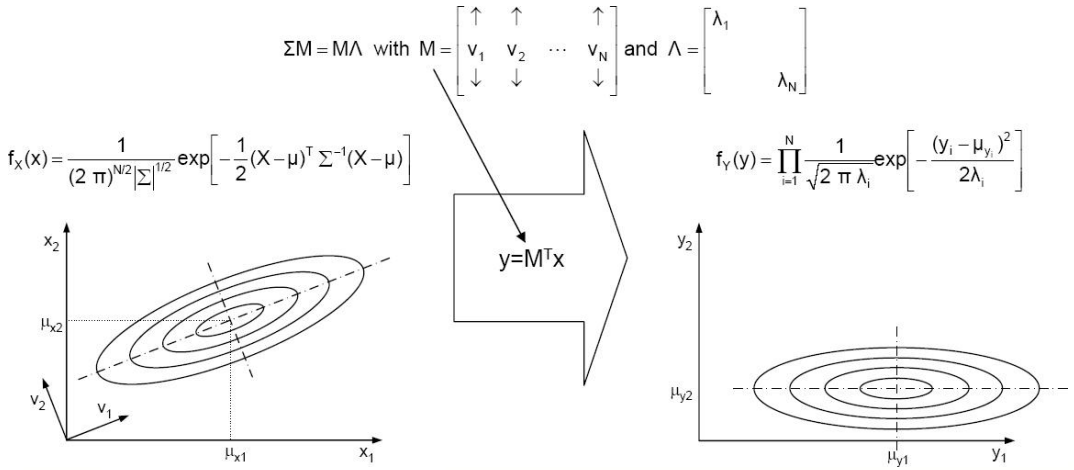
- To minimize this measure, choose λ_is to be the smallest eigenvalues.
- Therefore, to represent ***x*** with *minimum error*, choose the eigenvectors ϕ_i corresponding to the largest eigenvalues λ_i.

PCA Dimensionality Reduction

The optimal approximation of a random vector $x \in \mathbb{R}^n$ by a linear combination of $m(: m < n)$ independent vectors is obtained by projecting the random vector x onto the eigenvectors ϕ_i corresponding to the largest eigenvalues λ_i of the covariance matrix Σ_x.

A simple implementation of PCA

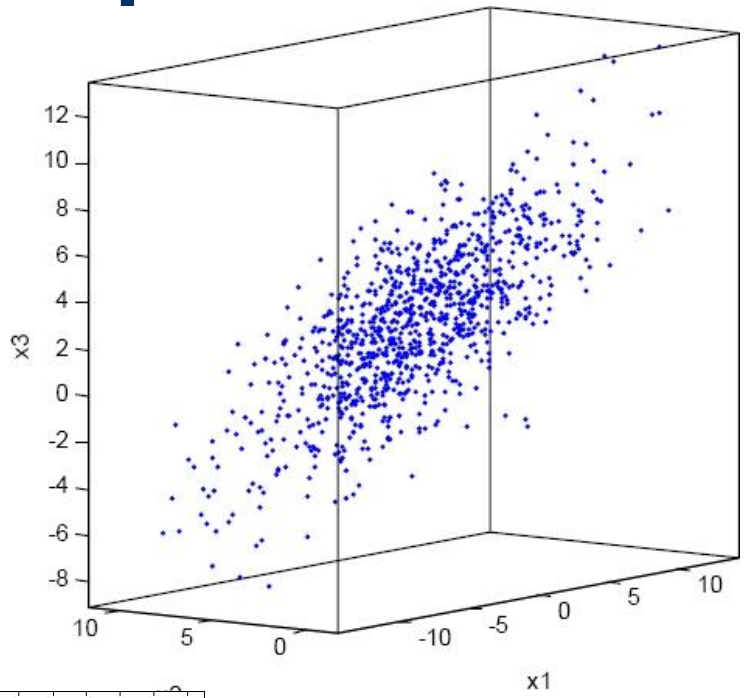
- Given m data points x_i each of dimension n.
- Compute the mean $\mu = \frac{1}{m}\sum_{i=i}^{m} x_i$, and subtract it from each data point (centering): ${x_i}^c = x_i - \mu$
- Compute the data matrix X where each column is a data point ${x_i}^c$
- Compute the covariance matrix, $\Sigma = \frac{1}{m} X X^T$
- Find the eigen-vectors and values of Σ
- The **principal components** are the k eigen-vectors with highest eigenvalues

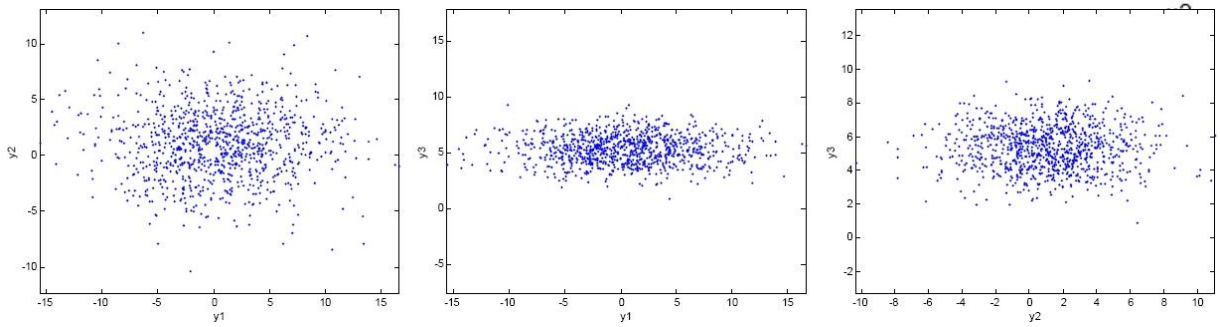


PCA – very simple example

- Assume data distributed as a Gaussian with $\mu = (0,5,2)^T$ and

$$\Sigma = \begin{bmatrix} 25 & 1 & 7 \\ 1 & 4 & -4 \\ 7 & -4 & 10 \end{bmatrix}$$





The 3 pairs of the principal component projections - the first projection has largest variation. The PCA projections are de-correlated.

Eigenfaces – PCA example

- Let $X = \{x_1, x_2, \ldots, x_m\}$ be a collection of feature vectors. Each feature vector $x_i \in [0, 255]^{N^2}$ corresponds to the pixel values of a visual image $(N \times N)$ of a face

Eigenfaces – PCA example

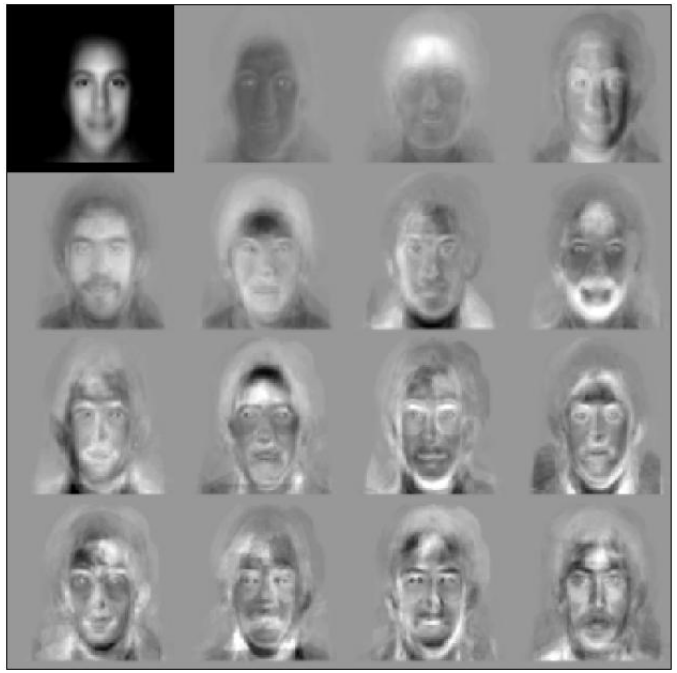
The mean face is $\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$, and

$x_i^* = x_i - \mu$

The eigenfaces are the PCA basis vectors. These are found by finding the eigenvectors of

$$\Sigma_x = \frac{1}{m}\sum_{i=1}^{m} x_i^* x_i^{*T} = (AA^T)$$

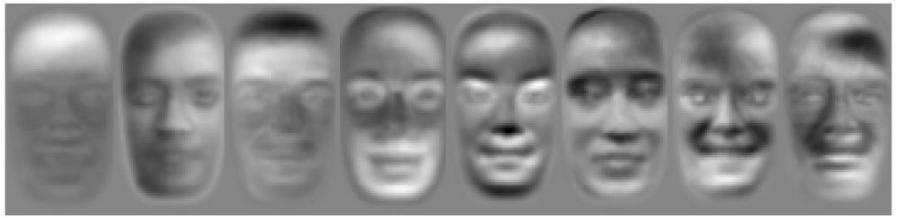
Note x may be huge ($N^2 \times N^2$). Need to use a trick to finding the eigenvectors of $A^T A$ (of size $m \times m$) since a lot of the eigenvalues must be zero.
If $(A^T A)v_i = \lambda_i v_i$ then
$\Sigma_x A v_i = (AA^T)Av_i = A(A^T A)v_i = \lambda_i A v_i \Rightarrow$
Av_i is an eigenvector of Σ_x .



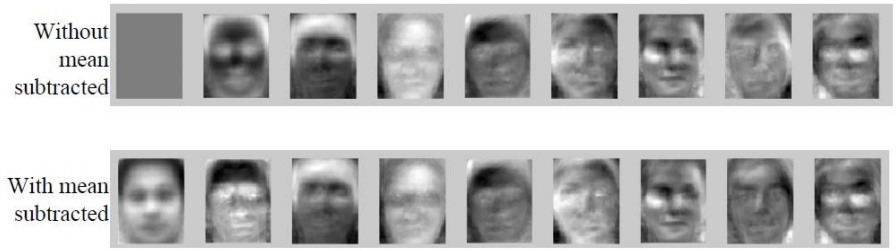
mean and **eigen-faces**

Eigenfaces – PCA example

- Train system on verified faces, finding the eigenfaces
- Map *new* faces onto eigenvectors
- *«Classify» by comparing coeffients / «distance».*



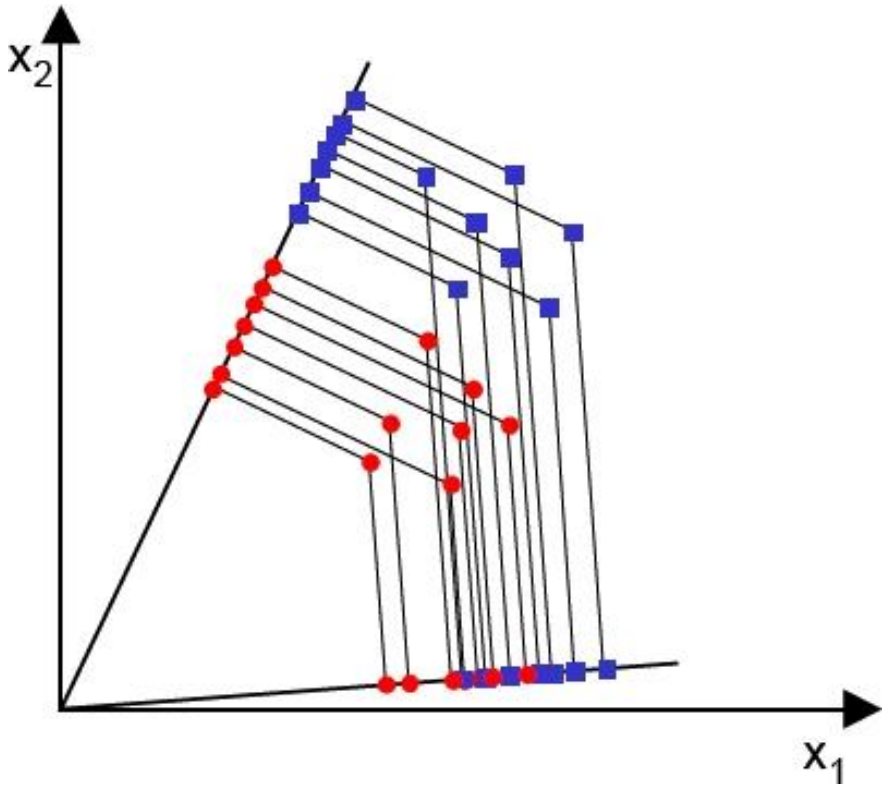
Standard Eigenfaces (← increasing eigenvalue)



Effect of the subtraction of the mean

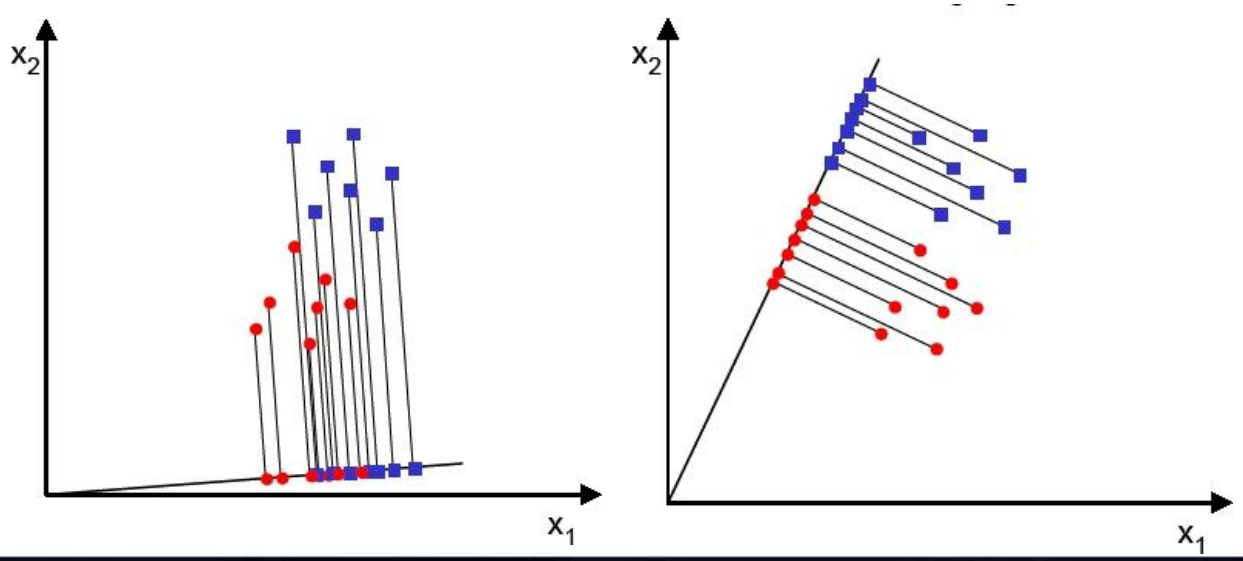
Linear discriminant analysis, LDA

The objective of LDA is to perform dimensionality reduction while preserving as much of the class discriminatory information as possible.



LDA, two classes

- Assume we have a set of D-dimensional samples $\{x_1, x_2, \cdots, x_N\}$, N_1 of which belong to class ω_1, and N_2 to class ω_2. We seek to obtain a scalar y by projecting the samples x onto a line $y = w^T x$
- Of all the possible lines we would like to select the one that maximizes the separability of the scalars



LDA, two classes

- In order to find a good projection vector, we need to define a measure of separation between the projections
- The mean vector of each class in x and y feature space

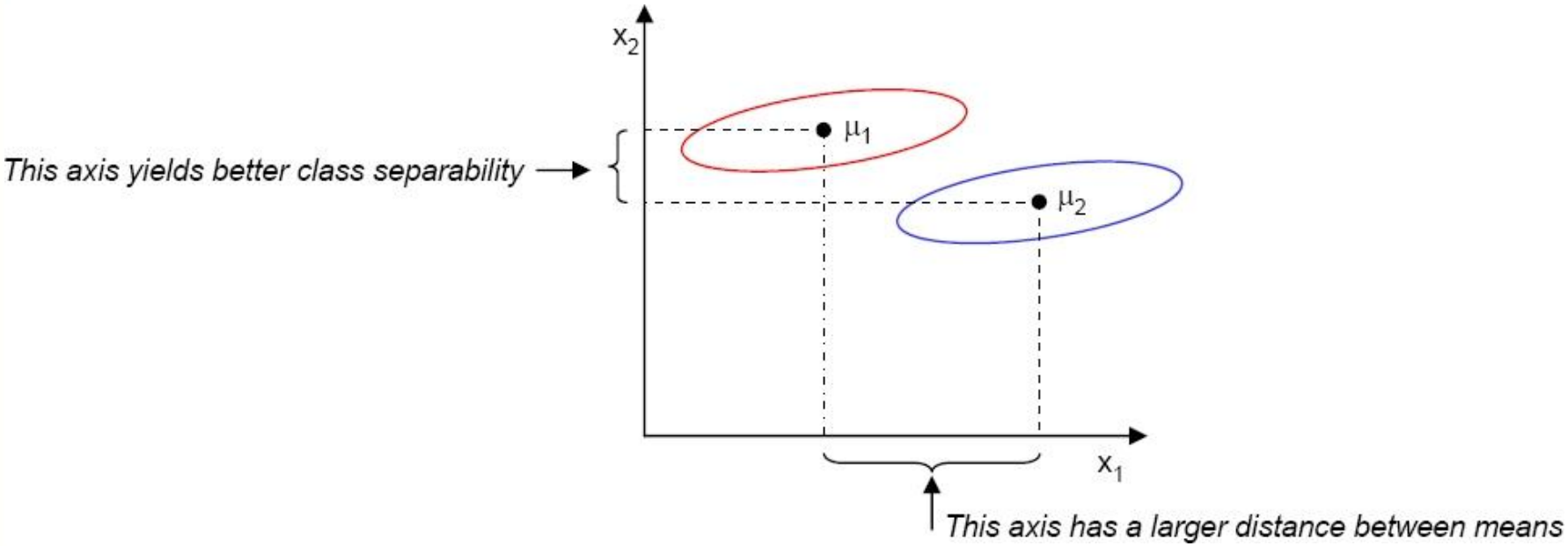
$$\mu_i = \frac{1}{N_i}\sum_{x\in\omega_i} x \,, \tilde{\mu}_i = \frac{1}{N_i}\sum_{y\in\omega_i} y = \frac{1}{N_i}\sum_{x\in\omega_i} w^T x = w^T \mu_i$$

- We could then choose the distance between the projected means as our objective function?

$$J(w) = |\tilde{\mu}_1 - \tilde{\mu}_2| = |w^T(\mu_1 - \mu_2)|$$

- However, the distance between the projected means is not a very good measure since it does not take into account the standard deviation within the classes.
- It is very easy to break this criterion!

LDA, two classes



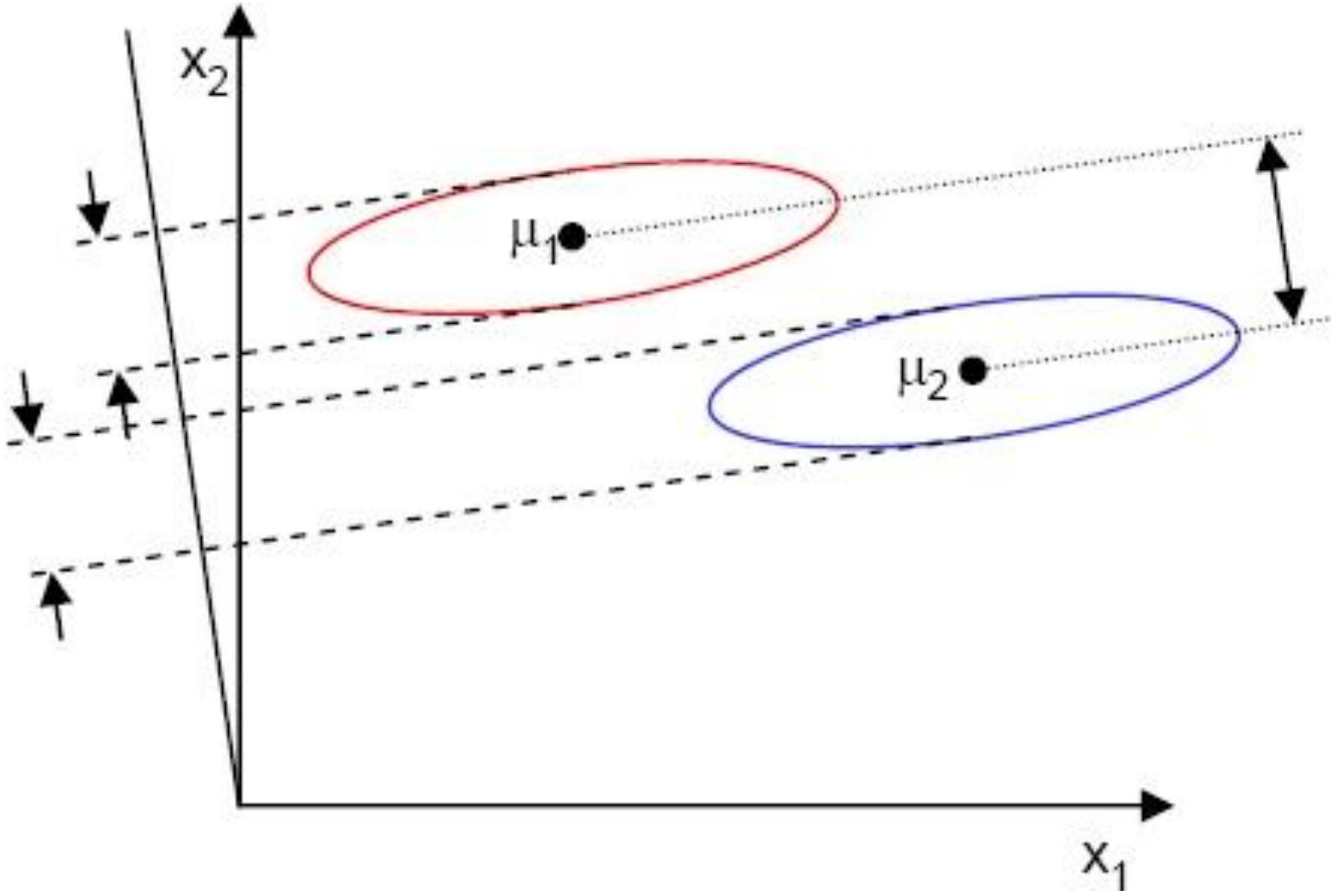
LDA, two classes

- The solution proposed by **Fisher** is to maximize a function that represents the difference between the means, normalized by a measure of the within-class scatter.
- For each class we define the scatter, an equivalent of the variance, as $\tilde{s}_i^2 = \sum_{y \in \omega_i} (\mathrm{y} - \tilde{\mu}_i)^2$
- Then the quantity $(\tilde{s}_1^2 + \tilde{s}_2^2)$ is called the within-class scatter of the projected examples.
- The Fisher linear discriminant is defined as the linear projection $w^T x$ that maximizes the criterion function

$$J(w) = |\tilde{\mu}_1 - \tilde{\mu}_2|^2 / (\tilde{s}_1^2 + \tilde{s}_2^2)$$

- Therefore, we will be looking for a projection where examples from the same class are projected very close to each other and, at the same time, the projected means are as far apart as possible.

LDA, two classes



LDA, two classes

- To find the optimum projection w, we need to express $J(w)$ as an explicit function of w.
- We define a measure of the scatter in multivariate feature space x, which are *scatter matrices*.

$$S_W = S_1 + S_2, \text{where } S_i = \sum_{x \in \omega_i} (x - \mu_i)^T (x - \mu_i)$$

- The matrix S_W is called the within-class scatter matrix.
- The scatter of the projection y can then be expressed as a function of the scatter matrix in feature space x.

$$\tilde{s}_i^2 = \sum_{y \in \omega_i} (\mathrm{y} - \tilde{\mu}_i)^2 = \sum_{x \in \omega_i} w^T (x - \mu_i)^T (x - \mu_i) w = w^T S_i w$$

$$\tilde{s}_1^2 + \tilde{s}_2^2 = w^T S_W w$$

LDA, two classes

- Similarly, the difference between the projected means can be expressed in terms of the means in the original feature space

$$(\tilde{\mu}_1 - \tilde{\mu}_2)^2 = w^T(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w$$

- The matrix S_B is called the between-class scatter. Note that, since S_B is the outer product of two vectors, its rank is at most one.
- We can finally express the Fisher criterion in terms of S_W and S_B as

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

LDA, two classes

- To find the maximum of $J(w)$ we differentiate and set to zero
- $$\frac{\partial}{\partial w} J(w) = \frac{\partial}{\partial w}\left[\frac{w^T S_B w}{w^T S_W w}\right] = 0$$

$$\Rightarrow (w^T S_W w)\frac{\partial}{\partial w}[w^T S_B w] - (w^T S_B w)\frac{\partial}{\partial w}[w^T S_W w] = 0$$

$$\Rightarrow (w^T S_W w)\frac{\partial}{\partial w}[w^T S_B w] = (w^T S_B w)\, S_W w$$

- From the definition of $S_B, S_B w \propto (\mu_1 - \mu_2)$
- Only direction of w is important. Why?
- Drop scale factors and multiply both sides by S_W^{-1}

$$w \propto S_W^{-1}(\mu_1 - \mu_2)$$

Fishers Linear Discriminant (LDA)

- Thus, the result of *Fisher (1936)*

$$w^* = \underset{w}{\operatorname{argmax}} \frac{w^T S_B w}{w^T S_W w} = S_W^{-1}(\mu_1 - \mu_2)$$

- Technically, it is not a discriminant, but rather a choice of projection down to one dimension where a treshold can be set.

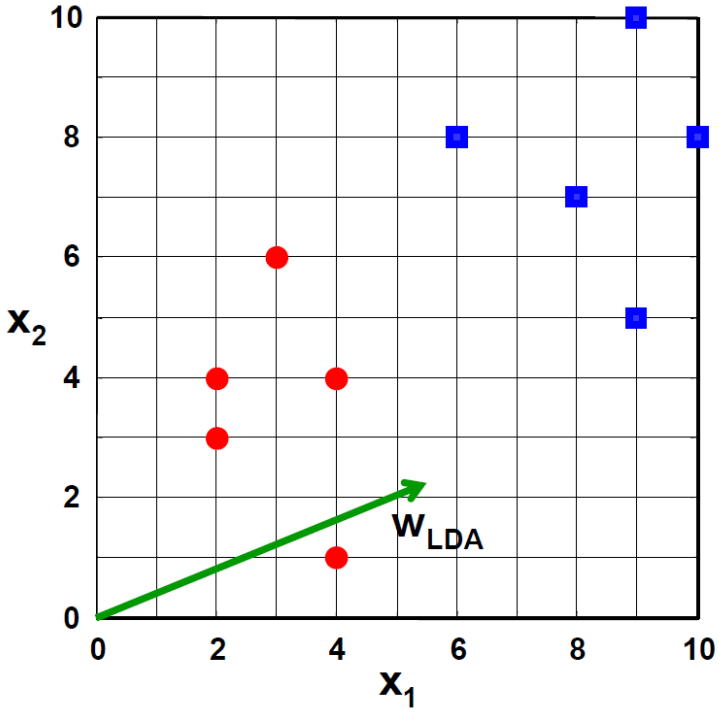
LDA as a classifier

- Use w to construct a classifier. Project the training data onto the line w. Must find a threshold θ such that:

  $w^T x \geq \theta \Rightarrow x$ belongs to class ω_1

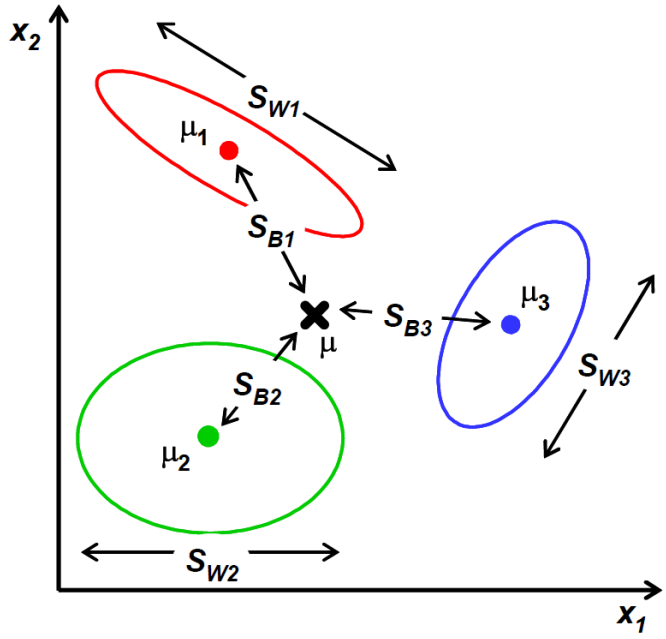
  $w^T x < \theta \Rightarrow x$ belongs to class ω_2

- How should we learn θ?
  - Project all training data onto w
  - Choose the θ that minimizes training error



LDA, several classes

- Fisher's LDA generalizes *very* gracefully for C-class problems. Instead of one projection y, we now seek $(C-1)$ projections $(y_1, y_2, \ldots, y_{C-1})$ by means of $(C-1)$ projection vectors w_i, which can be arranged by columns into a projection matrix $W = [w_1 w_2 \ldots w_{C-1}]$.

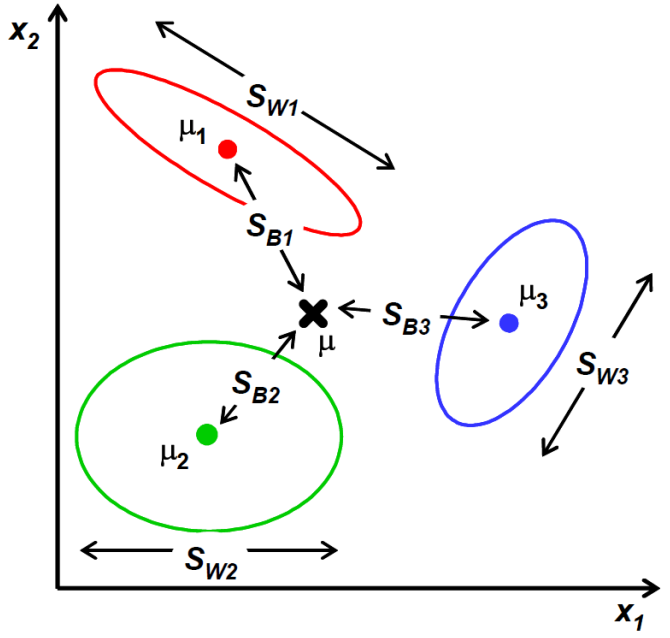


LDA, several classes

- The generalization of the within-class scatter is

$$S_W = \sum_{i=1}^{C} S_i, \text{where } S_i = \sum_{x \in \omega_i} (x - \mu_i)^T (x - \mu_i),$$

$$\text{and } \mu_i = \frac{1}{N_i} \sum_{x \in \omega_i} x$$



- The generalization for the between-class scatter is

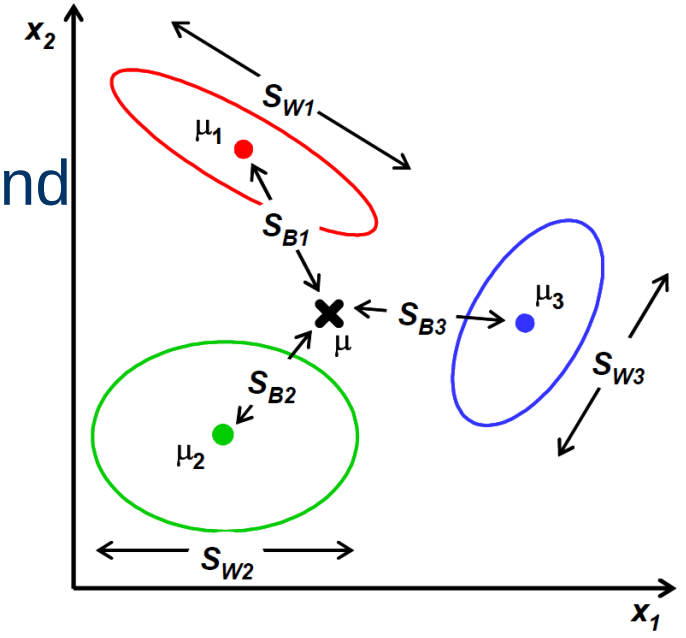
$$S_B = \sum_{i=1}^{C} N_i (\mu_i - \mu)^T (\mu_i - \mu), \text{where } \mu = \frac{1}{N} \sum_{\forall x} x$$

LDA, several classes

- The projected samples have mean and scatter

$$\tilde{S}_W = \sum_{i=1}^{C} \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^T (y - \tilde{\mu}_i),$$

$$\text{and } \tilde{\mu}_i = \frac{1}{N_i} \sum_{y \in \omega_i} y$$



$$\tilde{S}_B = \sum_{i=1}^{C} N_i (\tilde{\mu}_i - \tilde{\mu})^T (\tilde{\mu}_i - \tilde{\mu}) \text{, where } \tilde{\mu} = \frac{1}{N} \sum_{\forall y} y$$

LDA, several classes

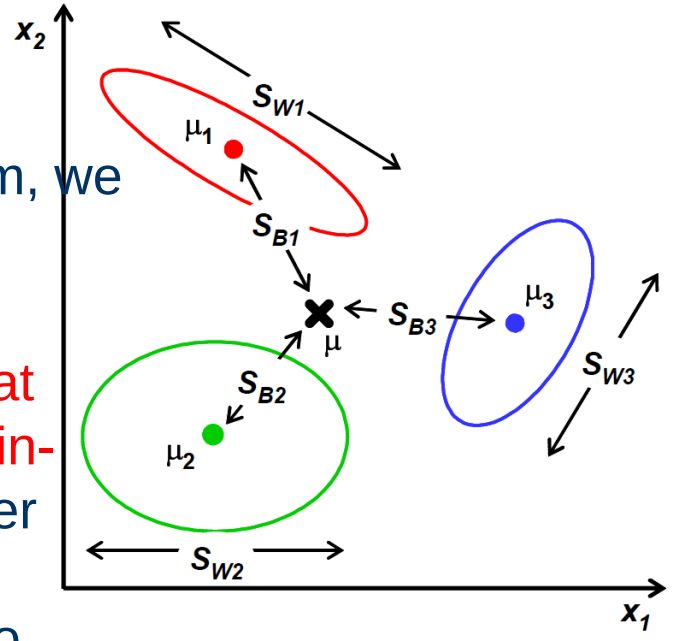
- From our derivation for the two-class problem, we can write

$$\tilde{S}_W = W^T S_W W, \tilde{S}_B = W^T S_B W$$

- Recall that we are looking for a projection that maximizes the ratio of between-class to within-class scatter. Since the projection is no longer a scalar (it has $C-1$ dimensions), we then use the determinant of the scatter matrices to obtain a scalar objective function:

$$J(W) = \frac{|\tilde{S}_B|}{|\tilde{S}_W|} = \frac{|W^T S_B W|}{|W^T S_W W|}$$

- We seek the projection matrix W that maximizes this ratio

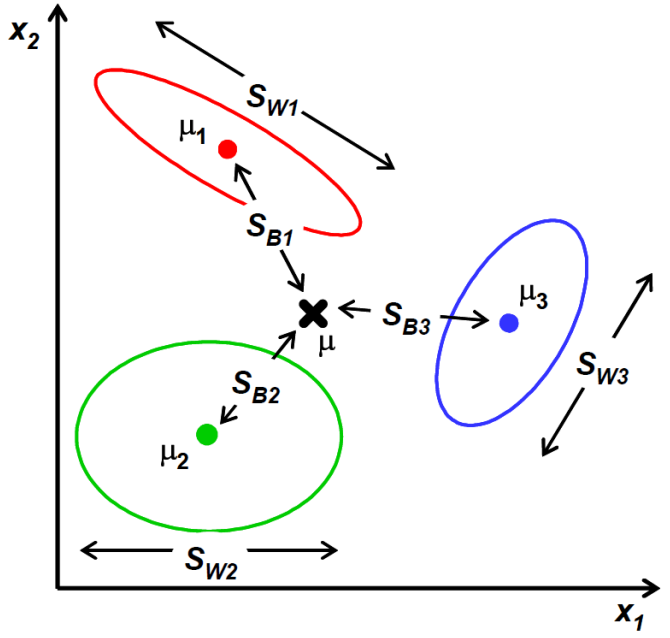


LDA, several classes

- It can be shown that the optimal projection matrix W is the one whose columns are the eigenvectors corresponding to the largest eigenvalues of the following generalized eigenvalue problem.

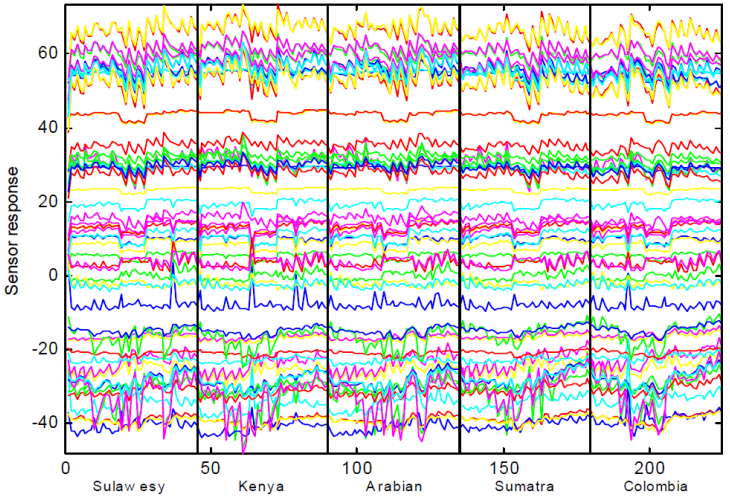
$$W^* = [w_1^* w_2^* \dots w_{C-1}^*] = \underset{W}{\operatorname{argmax}} \frac{|W^T S_B W|}{|W^T S_W W|}$$
$$= (S_B - \lambda_i S_W) w_i^*$$

- S_B is the sum of C matrices of rank one or less and the mean vectors are constrained by $\mu = \frac{1}{C}\sum_{i=1}^{C} \mu_i$
- Therefore, S_B will be of rank $\leq (C-1) \Rightarrow$ only (C−1) of the eigenvalues λ_i will be non-zero
- The projections with maximum class separability information are the eigenvectors corresponding to the largest eigenvalues of $S_W^{-1} S_B$



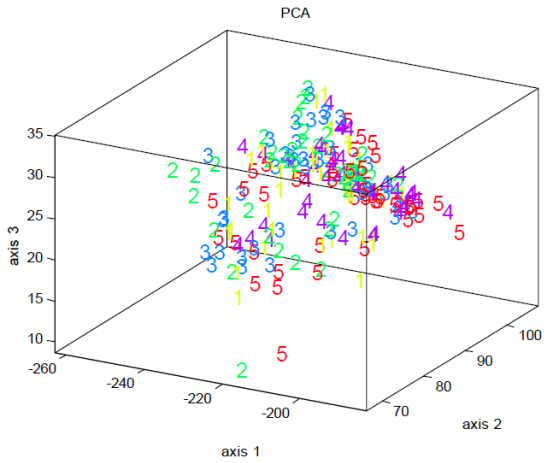
Coffee discrimination using a gas sensor array

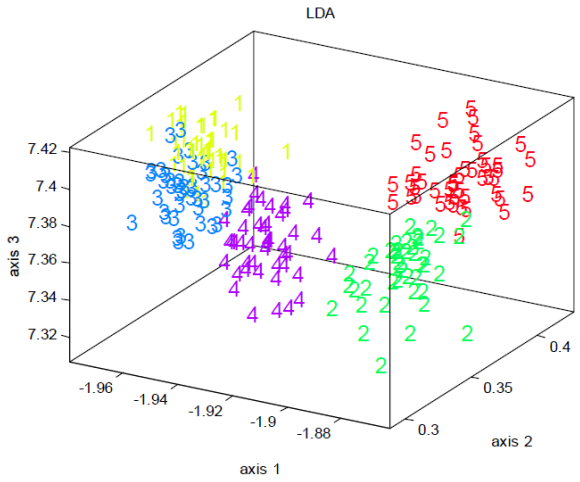
- **LDA on an odor recognition problem**
- Five types of beans were presented to an array of chemical gas sensors
- For each coffee type, 45 “sniffs” were performed and the response of the gas sensor array was processed in order to obtain a 60-dimensional feature vector



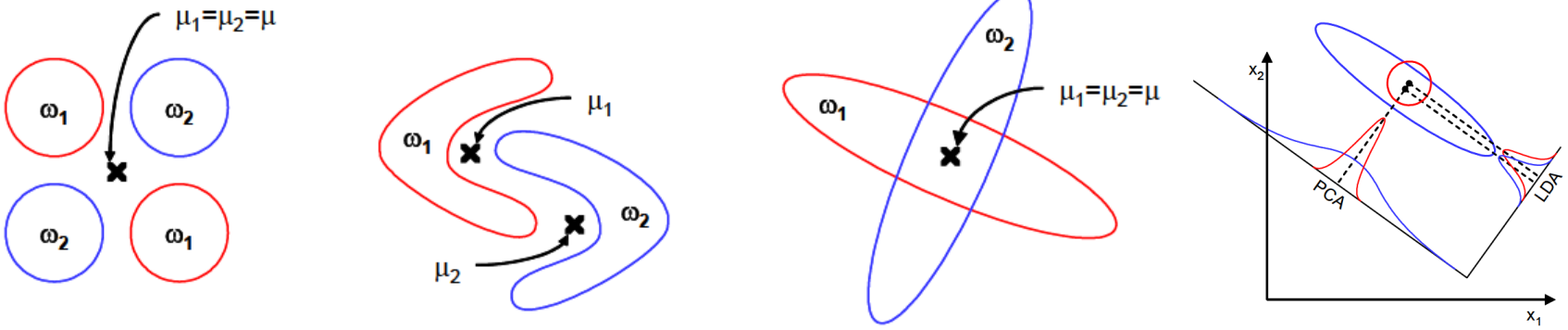
Coffee discrimination using a gas sensor array

- From the 3D scatter plots it is clear that LDA outperforms PCA in terms of class discrimination
- This is one example where the discriminatory information is not aligned with the direction of maximum variance
- Typically, this is the case for most well-behaved classification problems





Problems/limitations of LDA



- LDA produces at most $C - 1$ feature projections
  - If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features
- LDA is a parametric method since it assumes unimodal Gaussian likelihoods
  - If the distributions(/clusters) are significantly non-Gaussian, the LDA projections will not be able to preserve any complex structure of the data, which may be needed for classification.
- LDA will fail when the discriminatory information is not in the mean but rather in the *variance* of the data.

What to remember from this lecture

- Distance measures, feature selection algorithms
  - Understand basic search strategies
- Training data is a sparse resource, and should be used efficiently
  - Crossvalidation is usually a good approach for using data to decide parameters of classifiers
- PCA and LDA used for preprocessing of the data
  - before classification to beat the «curse of dimensionality»
  - to learn something about class distribution when data is very high dimensional