Multidimensional data analysis

Ella Bingham
Dept of Computer Science, University of Helsinki
ella.bingham@cs.helsinki.fi

June 2008

The Finnish Graduate School in Astronomy and Space Physics
Summer School 2008:
Multi-dimensional Data Analysis and Data Mining

Practicalities

- My lectures and exercises are all of Monday, and Tuesday until lunch at noon
- Please ask questions at any time
- Contact: during the lectures and breaks (in Finnish and English). I will leave on Tuesday at noon!
- Available later via email and in person in Kumpula, Helsinki
- Choice of material: hopefully related to types of data you work with

- Acknowledgements to my colleagues Saara Hyvönen, Juho Rousu, Pirjo Moen etc for putting their lecture material in public domain

- Acknowledgements to Noora Partamies and Mikko Syrjäsuo for providing some data and codes

Contents

- Part I: Some general concepts
- Part II: Methods

Part I: Some general concepts

These concepts and issues are related to all kinds of data mining, irrespective of the method chosen.

What is data mining

- Data mining, statistical data analysis, multidimensional data analysis, etc will be used as synonyms
- Goals: (1) Summarize, describe and explore the data (2) Infer the nature of the process which produced the data.
- Often (1) is done to accomplish (2)
- "Extraction of interesting (= non-trivial, unknown, potentially useful) information or patterns from data in large databases"

- "Tell me something interesting about this data." "Describe this data."

- Data mining mainly means we take the data as it is:
  - we do not assume a model that the data would follow (An important difference to your daily research!)
  - we explore the properties of the data, and try to infer about the underlying phenomenon

- We try to construct a model of what we see in the data. This is called *learning*. Here "model" is a very general concept; details will depend on the type of data and the method chosen to model the data

Some literature

- Han & Kamber: Data Mining: Concepts and Techniques, Morgan Kaufmann, 2006. http://www-faculty.cs.uiuc.edu/ hanj/bk2/

- Tan & Stenbach & Kumar: Introduction to Data Mining, Addison-Wesley, 2006. http://www-users.cs.umn.edu/ kumar/dmbook/index.php

- An electronic mini version of the book H. Lohninger: Teach/Me Data Analysis, Springer-Verlag, 1999 is at http://www.vias.org/tmdatanaleng/

- Wikipedia

Variables and observations

An issue to decide at startup: what are your *variables*
(columns, dimensions, features, attributes)
and what are your *observations*
(rows, items, measurements, replications, . . .)

- The distinction is not always easy, but must be done!
- Examples: in image data of handwritten digits, each pixel is a variable, and each image is one observation
- In contrast, in spectral image data, you only have one image but several measurements of it: each spectral channel is a variable, and each pixel is an observation

- in time series, each time point is one observation, and whatever you measure are the variables

- Notation to be used:
  $j = 1, \ldots, D$ are the variables and
  $i = 1, \ldots, N$ are the observations.
  x_{ij} is the jth variable of the ith observation.

Data preprocessing

- Partly based on expert knowledge, but some guidelines exist too.
- In data mining, we aim at preprocessing the data so that the choice of the modeling method is independent of the original form of the data (whether images, text, . . .)
- Outlier removal: might be measurement errors, and might make subsequent preprocessing error-prone. Try to identify them by
  - plotting the data with respect to different variables
  - plotting the *histograms* of each variable separately

- Handling of missing values: use caution.
  - remove the whole observation if one of its variables is missing?
  - replace by the average value?
  - replace by NaN? (Often works in Matlab)
  - do **not** replace by 0 or 999999 or -1 etc!

- Preprocessing needed if the data are not in vector/matrix form

- Needed if the variables are not comparable to each other (e.g. one variable is measured as kilometres, another as millimetres, another as colours)

- Needed to make it meaningful to measure the distances between data points

  Euclidean distance: $d(x, z) = \sqrt{\sum_{j=1}^{D}(x_j - z_j)^2}$

  Manhattan distance: $d(x, z) = \sum_{j=1}^{D} |x_j - z_j|$

  Hamming distance: $d(x, z) = \sum_{j=1}^{D} 1_{x_j \neq z_j}$

  Here x_j are the variables in observation x.

- When measuring distances, each variable is equally important.
  If two variables measure a more or less similar phenomenon, then this phenomenon becomes double as important as the other variables.
  If you do not want this to happen, you must either remove such duplicate variables or make your variables *uncorrelated* — we will return to this

- Preprocessing needed if the data is not numerical and we want to use a method for numerical variables

- Example: Transforming a nominal (categorical) variable into numerical. Assume the variable Fruit takes values in the set {"apple", "pear", "orange"}.
  Replace the variable Fruit with 3 new variables: Apple, Pear, Orange.
  If the value of Fruit in observation i was "pear", then set the values of the new variables in observation i as Apple=0, Pear=1, Orange=0.

- Preprocessing needed if the variance within one variable j is much larger than the variance within another variable j'. Here a small difference in j' should be as important as a large difference in j, but our data mining method does not understand this.

- *Normalization* solves the problems of different scales (kilometres vs millimetres) and variances:
  First, center the variable j by subtracting the mean μ_j over all observations; this makes the variable zero mean.
  Then divide by the standard deviation σ_j of variable j.
  The new, normalized variable is thus
  $x_j^{new} = (x_j - \mu_j)/\sigma_j$.
  Replace all observations x_{ij} by x_{ij}^{new}.

- Also other possibilities: logarithmic transformation $x_j^{new} = \log x_j$ if the values of x_j vary very much, and it is hard to see the behaviour at small values. Also gaussian and sigmoid transformations.

- Instead of preprocessing heterogenous data, we can construct a suitable distance function at each variable — but we might prefer automatic methods!

Bootstrapping

- Resampling: Assume your data set had N observations.
  Create a new data set having N observations by sampling from the original data *with replacement* N times.
  (Some observations will appear several times, some not at all.)

- Create M such new data sets; typically $M = 1000$ or $M = 10000$.

- Measure the phenomenon of interest in each of these new data sets.
  Does it vary a lot? If yes then the phenomenon was not "real" or "consistent".

- Confidence intervals for the phenomenon of interest can be obtained by bootstrapping (software exists)

Characterizing the uncertainty

How do you know when you don't know?
Most data mining methods always give you an output, also when there is not much evidence behind the output.
It would be good to know how reliable the output is.

Methods:
Bootstrap confidence intervals of the phenomenon
Permutation tests
Randomization

Permutation tests

You have observed an interesting phenomenon, using a nice method. Could it occur "just by chance" in your data, using your method?

Example: You notice a strong correlation between variables x_j and $x_{j'}$.

- Reorder the observations randomly (that is, permute)
- and measure the correlation between x_j (in the original ordering) and $x_{j'}^{perm}$ (in permuted ordering).
- Is the correlation still as strong?

- Repeat the permutation 1000 times. In how many cases did you get as strong correlation as (or stronger than) in the original setting?

- Several cases $\rightarrow$ The phenomenon was "just by chance".
  Very few cases $\rightarrow$ The phenomenon is "real".

- You can even get a P value: P = percentage of repeatments in which the phenomenon was as strong or stronger than in the original setting.

- Applicable also when your data is not Normally distributed (and many other methods are inappropriate)

Overfitting

A problem to be concerned about. Your model is too flexible, it will learn unnecessary details in your data, and is not applicable to other data.

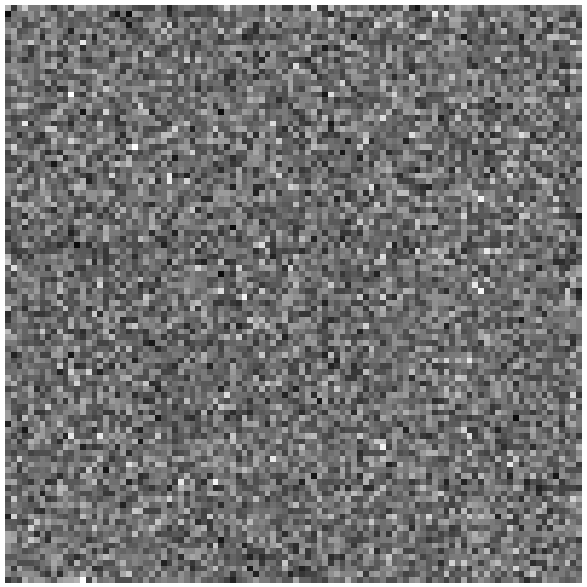
Will occur if you use too complicated a model, especially in high dimensional spaces where the observations are far away from each other, and it is practically impossible to have dense observations.

Curse of dimensionality

Dimensionality = number of *variables*.

Suppose you have observations that follow a uniform distribution, in 1 dimension (left) and 2 dimensions (right).

N=100000, 1D



N=100000, 2D

- You don't know the distribution but try to infer it from the observations.
- You proceed by dividing the data into 100 bins in each dimension, and calculate the number of observations in each bin.
- If the number is roughly the same in each bin, you conclude that the distribution is uniform.
- Assume you have N observations that fall into these bins randomly.
- In a perfect uniform distribution you should have exactly $N/100$ (1-dimensional) or $N/(100 \cdot 100)$ (2-dimensional) observations in each bin.

- In practice there is some variation, and the average squared distance from the uniform distribution depends on N:

number of observations N	Average squared distance from uniform distribution	
	1-dimensional	2-dimensional
100	$1.06 \cdot 10^{0}$	$1.01 \cdot 10^{2}$
1000	$1.09 \cdot 10^{-1}$	$1.03 \cdot 10^{1}$
10000	$1.23 \cdot 10^{-2}$	$1.00 \cdot 10^{0}$
100000	$1.17 \cdot 10^{-3}$	$1.03 \cdot 10^{-1}$
1000000	$9.45 \cdot 10^{-5}$	$1.02 \cdot 10^{-2}$
10000000	$1.09 \cdot 10^{-5}$	$1.02 \cdot 10^{-3}$

- If you want to reach the same accuracy in 2D as in 1D, and you had N observations in 1D, you need $100 \cdot N$ observations in 2D!
- In 3D you would need $100 \cdot 100 \cdot N$ observations — exponential increase!
- Very high dimensional spaces have many counterintuitive properties:
- Observations are far away from each other
- To obtain a "densely populated" set of observations (important when learning), a large number N is needed
- Some other motivation: Small dimensions are easier for a human being to understand

- and storing the data is cheaper
- and finding outlier observations is easier
- and noise removal is easier
- We will see how to reduce the dimensionality
- A high number of *observations* can also be a problem $\longrightarrow$ take random samples.

Global or local?

The models can be divided into *global* and *local*

- global: the model characterizes all observations
- local: the model describes a small subset of observations

Supervised or unsupervised?

Data mining can be divided into *supervised* and *unsupervised* learning

- supervised: we have some *training data* (training observations) in which we *know what the model should give* as an output.

- Example of training data: we have measured a few variables and observed that Aurora borealis always appear when the variables take values in a certain range, and never appear when the variables take some other values.

- We construct a model that fits these observations, the output of the model is whether Aurora borealis appear or not.

- After that, we take a separate *test data* set and check if our model outputs a correct value at those observations.
- Methods of supervised learning: classification, regression
- Unsupervised: no such training data nor output exists
- reasons for this: it is very expensive or difficult to get values of the output, or hard to define what the output would be
- but also in unsupervised learning, we can split our data randomly into training and test data.
- Construct the model using the training data, and measure how well our model fits the test data

- Methods of unsupervised learning: clustering, latent variable models
- At some unsupervised methods, it is difficult to determine when the model "fits" the test data, if our aim is only to "explore" the data

Cross validation

- We would like to use as many observations as possible in training, to get the best possible model
- But then the test data set becomes very small, and we cannot draw conclusions on it
- Solution: cross validation
- Divide the observations into S subsets of equal size
- Use $S - 1$ of them as training data to construct the model, and 1 as test data to measure the error

- Repeat S times, using each subset in turn as test data
- The final error is the average of these S errors.
- In practice $S = 5$ or $S = 10$ are used

Part II: Methods

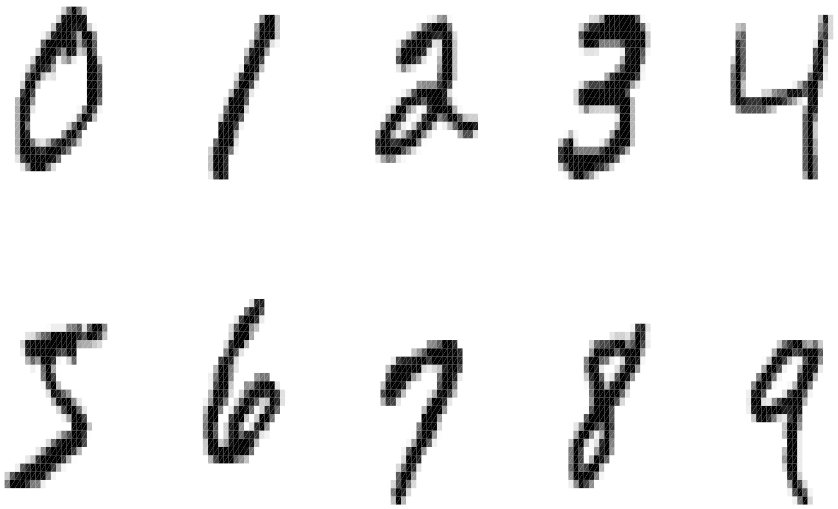
We will look at some modeling methods in detail,
and others only superficially.
But you can go and find more information yourself later.

Dimensionality reduction

- These methods can either be used as preprocessing, prior to some other data mining methods; or as such, to describe the data
- Two approaches:
- Feature selection: discard some variables (=features) completely
- Latent variables: Form a small set of new variables that describe the main characteristics of the data
- Let's look at latent variables first.

Example: Handwritten digits

We want to classify handwritten digits into classes 0–9



- Each digit is presented as a 100x100 gray scale image: one observation in a 10000 dimensional space.

- But the contents of the image can be described much more simply than using 10000 variables

- The contents are: which digit is shown, and how is it deformed (e.g. translated, rotated or scaled)

- The data can be presented with a much smaller number of variables: latent variables

Latent variables

A data matrix of size $N \times D$ is presented using K latent variables as a product of two matrices:

$$\mathbf{X}_{N \times D} = \mathbf{Y}_{N \times K} \mathbf{W}_{K \times D},$$

where the columns of $\mathbf{Y}$ are the latent variables
and the rows of $\mathbf{W}$ are called basis vectors.

A lot of different methods, outputting different latent variables.

Here we have a linear latent variable representation; could also be nonlinear

Singular value decomposition (SVD)

Every matrix $\mathbf{X}$ of size $N \times D$ can be presented as

$$\mathbf{X}_{N\times D} = \mathbf{U}_{N\times M}\mathbf{S}_{M\times M}(\mathbf{V}_{D\times M})^T,$$

where $M = \min(N, D)$.

The columns of matrix $\mathbf{U}$ are orthonormal (that is, perpendicular and of unit length), and they are called the *left singular vectors*.

The columns of $\mathbf{V}$ are also orthonormal, and called the *right singular vectors*.

Matrix $\mathbf{S}$ is a diagonal matrix:

$$\mathbf{S} = \text{diag}(\sigma_1, \sigma_2, ..., \sigma_M), \quad \sigma_1 \geq \sigma_2 \geq ... \geq \sigma_M \geq 0.$$

The values σ are called the *singular values*. The ith singular value shows the amount of variation along the ith dimension. Large value $\leftrightarrow$ important dimension

The ith singular value corresponds to the ith left singular vector and to the ith right singular vector

The singular value decomposition can also be written as

$$\mathbf{X}\mathbf{v}_j = \sigma_j \mathbf{u}_j,$$
$$\mathbf{X}^T\mathbf{u}_j = \sigma_j \mathbf{v}_j.$$

In practice we use some ready made software: Matlab, GNU Scientific Library, ALGLIB, JAMA, LAPACK, SVDLIBC

(Note that SVD is unique up to sign flips, and depending on your software you may get different results: some columns of $\mathbf{U}$ will have a minus sign, and similarly the corresponding columns in $\mathbf{V}$.)

Matlab example

```
X=      1      1
        1      2
        1      3
        1      4

[U,S,V]=svd(X,'econ')

U=    -0.2195    0.8073
      -0.3833    0.3912
      -0.5472   -0.0249
      -0.7110   -0.4410

S=     5.7794         0          V=    -0.3220    0.9467
            0    0.7738               -0.9467   -0.3220
```

Approximating a matrix

Let $\mathbf{U}_K = (\mathbf{u}_1\ \mathbf{u}_2\ ...\ \mathbf{u}_K)$, $\mathbf{V}_K = (\mathbf{v}_1\ \mathbf{v}_2\ ...\ \mathbf{v}_K)$ and $\mathbf{S}_K = \text{diag}(\sigma_1, \sigma_2, ..., \sigma_K)$, and let

$$\mathbf{X}_K = \mathbf{U}_K \mathbf{S}_K \mathbf{V}_K^T.$$

Now

$$\min_{\text{rank}(\mathbf{B}) \leq K} \|\mathbf{X} - \mathbf{B}\|_2 = \|\mathbf{X} - \mathbf{X}_K\|_2 = \sigma_{K+1}.$$

That is: the best (in terms of the Euclidean norm, and some other norms) rank K approximation of a matrix is given by the singular value decomposition, by taking the K first singular vectors and singular values.

Intuitively: in linear regression we find a one-dimensional line that best fits the two-dimensional points.
In SVD we find a K-dimensional space that best fits a D-dimensional space where $K \ll D$.

(What is a rank of a matrix? It is the number of linearly independent columns. And the columns are linearly independent if they cannot be presented as sums of other columns, possibly scaled. Linearly dependent columns do not bring any new information to the data. Also, the rank of a matrix is the number of nonzero singular values.)

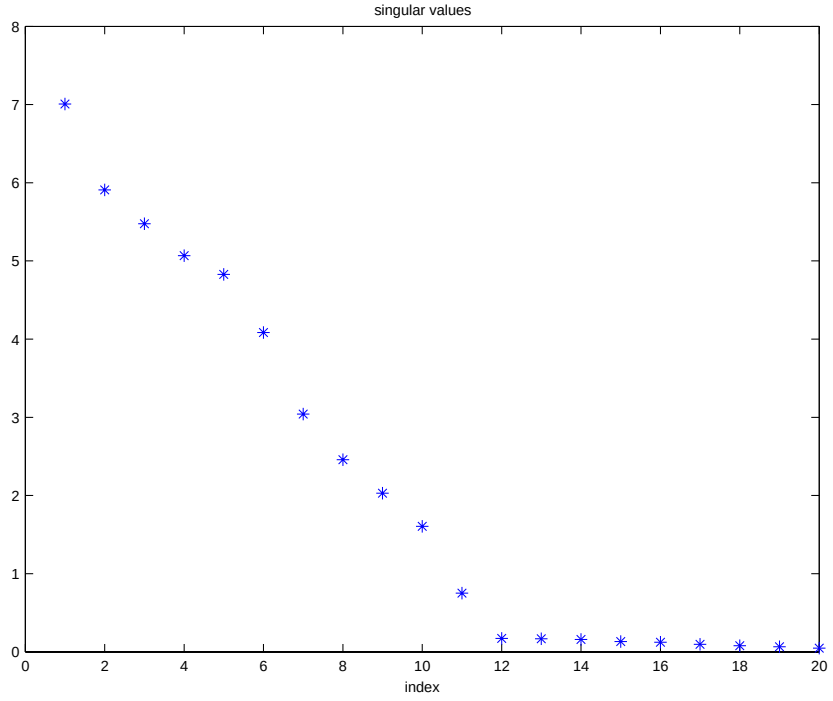
Then what?

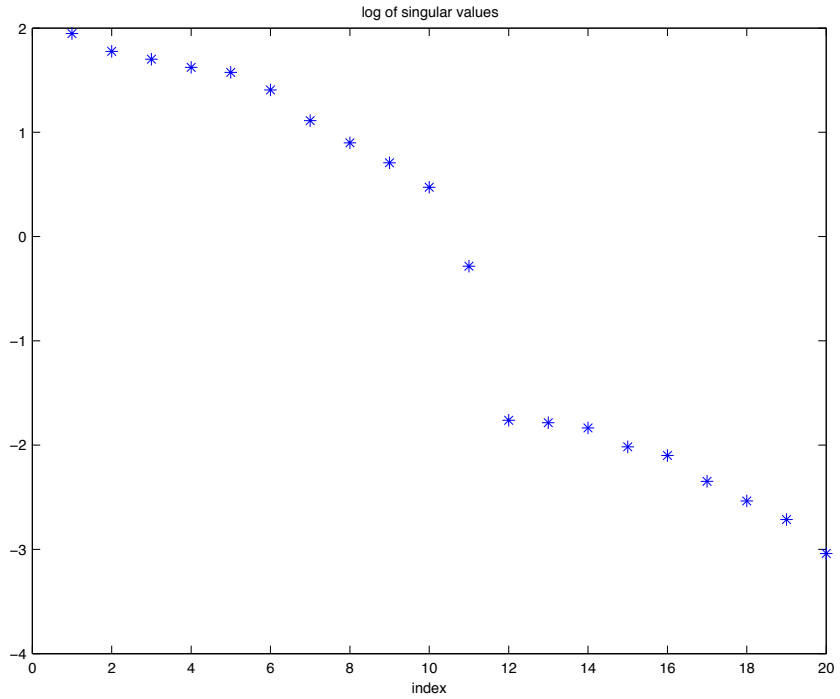
The low rank representation can be used in

- noise removal
- compressing the data
- finding the "sensible" number of dimensions in a matrix
- as a means of preprocessing

Example: noise removal

Let $\mathbf{X} = \mathbf{X}_K + \mathbf{N}$, where $\mathbf{X}_K$ is a matrix having a (low) rank K, and the noise in $\mathbf{N}$ is negligible such that $\|\mathbf{N}\|$ is small. In such a case the singular values reveal the amount of "interesting"content in the data:

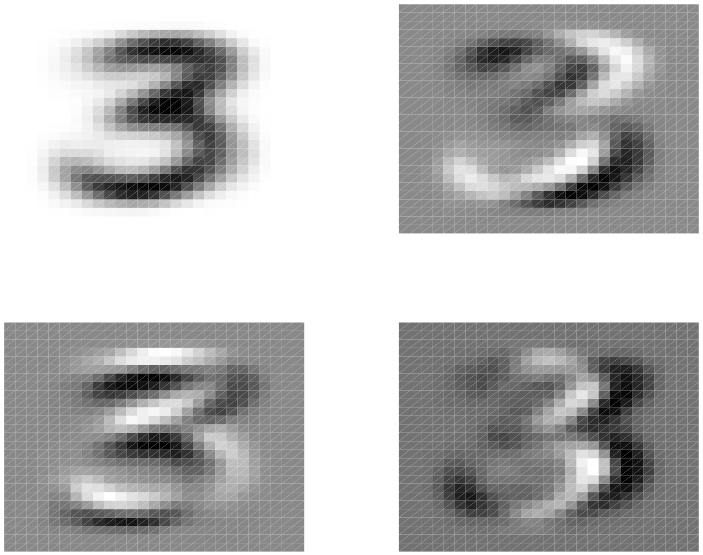




In practice: plug your data into SVD, and the output might reveal how much information there is, and how much the data could be compressed, by representing it using the SVD.

Example: handwritten digits

In a data set containing handwritten digits 3, represent each 3 as a vector of pixel values, put the vectors into rows of a matrix, and do the SVD. Left singular vectors $\mathbf{u}_j$:



The first singular vector is typically some kind of an average, and the others bring new aspects of the data

Computational complexity of SVD

- The time required for computing grows as M^2 ($M = \min$(nr of rows, nr of columns)) and K^3 (K=number of singular values)
- Efficient implementations exist if data is sparse
- If the number of observations N is large, we can compute SVD for a subset of the observations only . . .
- . . . and if the rest of the observations must also be presented in the SVD basis, multiply them by $\mathbf{V}$
- But most importantly, use as small K as possible, but not too small

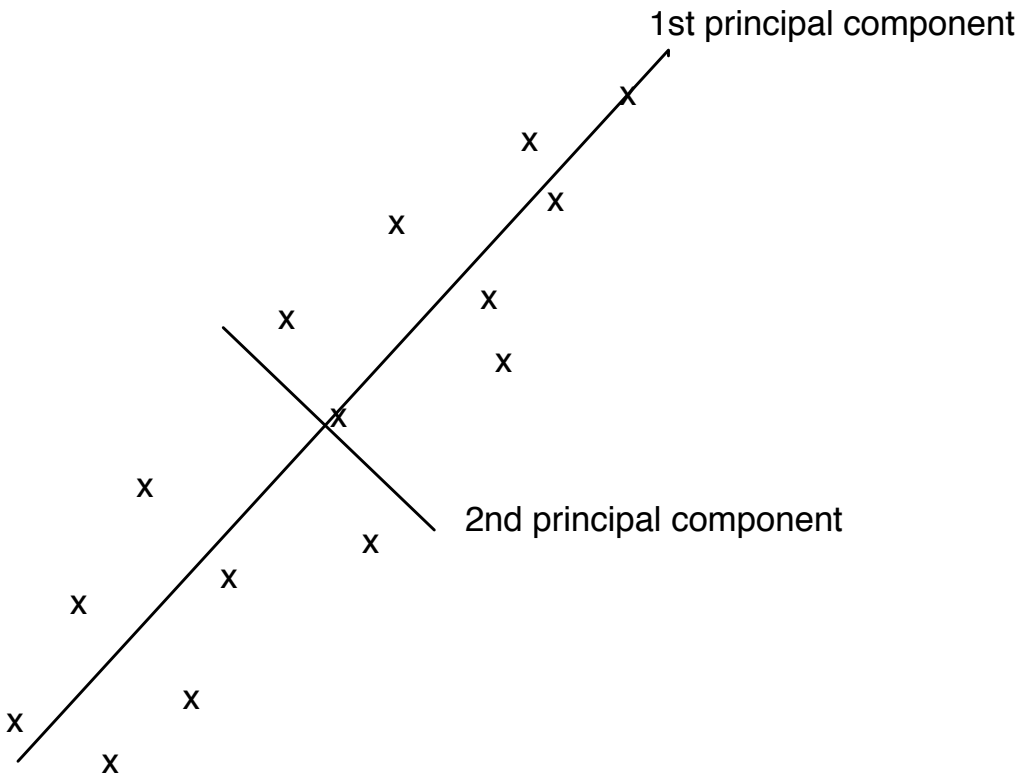
Choosing K

In practice, a few dozens is often fine. More advanced methods:

- Monitor the decay of the singular values and stop when there is a large gap between two values
- Compute the Frobenius norm of $\mathbf{X}_K$ versus $\mathbf{X}$: let $\mathbf{Y} = \mathbf{X} - \hat{\mathbf{X}}$, then F=$\sqrt{sum(diag(\mathbf{Y}^T\mathbf{Y}))}$ and compare this to the norm of the original $\mathbf{X}$; if small then $\hat{\mathbf{X}}$ is good enough and k is large enough
- Compare the sum of all singular values and the sum of K largest singular values, and choose K for which e.g. 85 % is obtained. (But this requires computing all of them)

Principal component analysis (PCA)

- The goal is to find a small set of artificial variables such that as much as possible of the variance of the data is retained.
- The first principal component is a vector in whose direction the variation of the data is the largest
- The next principal component is orthogonal to the first one, and it is in the direction of the second largest variation
- PCA is most suitable when the observations are Normally distributed or at least continuous. The output is hard to interpret if data are discrete.


1st principal component
2nd principal component

Computing the PCA

- Make the data zero mean (**Important!**) by subtracting from each column (variable) the mean of the column. Denote the new data by $\mathbf{X}$.
- Compute the SVD: $\mathbf{X} = \mathbf{USV}^T$. The principal components are the right singular vectors in $\mathbf{V}$.
- The variance along the ith principal component is given by the square of the ith singular value σ_i^2
- The new coordinates of the observations in the PCA basis are $\mathbf{US}$
- The new observations $\mathbf{US}$ are *uncorrelated* and behave nicely: remember we talked about measuring the distances in the 1st lecture

- The new observations are of smaller dimensionality and hopefully less noisy. They can be used in clustering, classification etc. instead of the original observations.

PCA and eigenvalue decomposition

- You might have heard that PCA is given by the eigenvalue decomposition of the covariance matrix of the (zero-mean) data

- Correct.

- But the eigenvalue decomposition is numerically less stable than the singular value decomposition, so we prefer the latter.

- There are nice connections between the eigenvectors and singular vectors

Matlab code

```
% Data matrix X, columns: variables, rows: data points
% Matlab function for computing the first k principal components of X.

function [pc,score] = pca(X,k);

[rows,cols] = size(X);
Xmeans = repmat(mean(X,1),rows,1); % matrix whose rows are column means
X = X - Xmeans; % centering the data
[U,S,V] = svds(X,k); % k is the number of pc:s desired
pc = V;
score = U*S; % now X = score*pc' + Xmeans;
```

Literature:

- Tutorial site by Todd Will: http://www.uwlax.edu/faculty/will/svd/
- Wikipedia: Singular value decomposition

Other uses of SVD

- Google's PageRank: ranking the pages to be displayed at a given query
- Spectral ordering: ranking the observations according to their similarity with respect to which variables are active ("nonzero")

Independent component analysis (ICA)

- A statistical technique that represents multidimensional data as a linear combination of nongaussian variables ('independent components')
- The variables are statistically as independent as possible
- Similar but stronger than PCA: suitable for non-Normally distributed data
- ICA has many applications in data analysis, source separation, and feature extraction
- An efficient algorithm, FastICA, implemented into Matlab, C++, R etc.

- ICA components of natural images are “edges” etc.
- http://www.cis.hut.fi/projects/ica/fastica/

Other latent variable methods

- Nonnegative matrix factorization
- Probabilistic PCA for continuous data (does not assume that data is Normally distributed)
- Mixtures of Bernoulli, Probabilistic latent semantic analysis, Multinomial PCA, Latent Dirichlet Allocation, Logistic PCA etc. for discrete-valued data

Random projection

- Not a latent variable method actually,
  but a dimensionality reduction method

- Suitable for high dimensional data where the variables are comparable to each other and equally important:
  e.g. pixels of an image; words of a document

- Take a random matrix $\mathbf{R}_{D\times K}$ whose entries are Normally distributed.
  Multiply the original data matrix by it:
  $\mathbf{X}^{RP}_{N\times K} = \mathbf{X}_{N\times D}\mathbf{R}_{D\times K}$

- The new data is of smaller dimensionality

- Random projection preserves the similarities of data points: on average, points that are close to each other in the original data $\mathbf{X}$, are still close to each other in $\mathbf{X}^{RP}$

- Very useful if we must measure the similarities of high dimensional data points (images, text documents etc.)

- Succesful in noise reduction in image data, too

- Computational complexity: cheaper than PCA, and comparable to discrete cosine transform (DCT). Preserves similarities almost as well as PCA and better than DCT.

- Ella Bingham and Heikki Mannila: "Random projection in dimensionality reduction: applications to image and text data", Proc. KDD-2001, pp. 245-250.
  http://www.cis.hut.fi/ella/publications/randproj_kdd.pdf

Feature selection

- Select the variables according to the final goal of your modeling task (regression, classification, clustering . . .)
- Suitable if you have a modest number of variables,
- and you want to find a smaller subset of them.

Forward stepwise selection:

- At each of your variables separately, construct a model.
- Measure the error of each model (regression error, classification error, clustering error . . .)

- Take the variable that gave the smallest error.
- Then construct a 2-variable model using the chosen variable and all other variables separately.
- Again measure the error of each 2-variable model
- and choose the 2nd variable which gave the smallest error
- Continue: at each step, choose the variable which (together with the chosen variables) gives the smallest error
- Stop when adding new variables does not significantly decrease the error any more, compared to the previous step.

Backward stepwise selection works in the other way round:

- Start by constructing a model using all variables
- Then remove one variable and measure the error; repeat at each variable in turn
- In which case was the error smallest? Continue with this subset of variables.
- Again try removing each of the remaining variables in turn, and find the subset of variables giving the smallest error
- Stop when removing new variables increases the error significantly, compared to the previous step.

Segmentation

- This is unsupervised learning
- For time series data
- Partition the time points into segments such that observations inside a segment are similar and different segments are dissimilar
- Easy for one-dimensional data
- Applicable to multi-dimensional data too
- A theoretically optimal method is Dynamical programming. Computationally slow.

- Can also combine with PCA: dimensionality reduction first, then segmenting of the PCA representation of the data

- Ella Bingham, Aristides Gionis, Niina Haiminen, Heli Hiisilä, Heikki Mannila, Evimaria Terzi, "Segmentation and dimensionality reduction". 2006 SIAM Conference on Data Mining, pp. 372-383 http://www.cs.helsinki.fi/u/ebingham/publications/basis_segmentation_pa

Clustering

- This is unsupervised learning
- Task: Divide the observations into groups (clusters) so that "similar" observations go together
- Must choose:
  - Which method to use?
  - How to measure the goodness of clustering?
  - Number of clusters K

Three types of clustering methods

- Partition the data directly into K clusters (example: K-means)
- Hierarchical methods: iteratively merge the observations into groups, and merge similar groups together, until you have K groups
- Probabilistic mixture models (Will be omitted)

Measuring the goodness of clustering

- Choose a representative for each cluster C_k as the "average" member:
  For numerical data, $r_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$
  For categorical data, $r_k =$ the most common element in the cluster
  We will call r_k as the *mean* of the cluster

- Observations inside each cluster should be similar:
  Minimize the within cluster (wc) variation.
  Measure the distance between each observation and the cluster mean:
  $wc_{total} = \sum_{k=1}^{K} wc(C_k) = \sum_{k=1}^{K} \sum_{x \in C_k} d(x, r_k)$

- Different clusters should be dissimilar:
  Maximize the between cluster variation

K-means clustering

Start from a random clustering, and iteratively change the clustering so that the goodness of clustering (=similarity of observations in a cluster) increases at each step. Continue until the clustering does not change.

Pseudo code:

Algorithm K-means(data, K)
Initialize the cluster means $r_1, \ldots, r_K$ by picking K observations
$k(x) = argmin_{k=1,\ldots,K} d(x, r_k)$ is the cluster of x
$C_k = \{x | k(x) = k\}$ for all $k = 1, \ldots, K$
while any C_k was changed
$r_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$ is the mean of cluster k; repeat at all k
$k(x) = argmin_{k=1,\ldots,K} d(x, r_k)$ is the cluster of x
$C_k = \{x | k(x) = k\}$ for all $k = 1, \ldots, K$
end while

Properties of K-means

- Works often well in practice
- Assumes the clusters are of nearly similar size, and roughly "ball-shaped"
- Time needed to run K-means is linearly dependent on K, N, ℓ and D where ℓ is the number of "while" iterations (typically smallish)
- So this is reasonably quick
- Problem: might find "local minima": clusterings that are not as good as they could be

- Solution: repeat at several different initializations, and choose the best clustering (in terms of within cluster variation, or other)

Choosing K: no clear answer. Some attempts:

- Stability of clustering — a "small" change in data will only result in a "small" change in the clustering, if K is "correct"

- Try different $K = 1, 2, \ldots$ and compute the within cluster variation (wc) at each K.
  The wc decreases steeply until the correct K is found, and then only decreases a little.

- In hierarchical clustering the output shows the behavior at different K

Hierarchical clustering

- Visualized as a *dendrogram*, a "tree"
- Choice of K not required as an input
- Agglomerative (bottom-up): Start by placing each object in its own cluster,
- and iteratively merge the clusters that are close to each other. Divisive (top-down): Start by placing all objects in the same cluster,
- and iteratively divide the cluster into two smaller clusters.

- In both approaches, the change in the within cluster error is seen at each step. The choice of K can be based on this.

Other clustering methods

- Self-organizing map for continuous data
- Spectral clustering for discrete or categorical data (Closely related to spectral ordering, mentioned along SVD)

k-nearest-neighbors classification

- This is supervised learning
- In your training data, you are given observations and their class labels
- For a new observation, find the k nearest neighbors among the training observations using a suitable distance function
- Classify the new observation by the major vote of those k labeled observations

Pseudo code:
Algorithm KNN(trainX,trainY,testX,k)
for each observation x in testX
compute the Euclidean distance between x and all observations in trainX
find the k observations having the smallest distance to x
find the most frequent class label in these k observations
output the class label for observation x
end for

Using SVD as preprocessing prior to classification:

- One might guess that representing a new handwritten "3" using the singular vectors from a data of several "3"s is easier than when using the singular vectors from a data of several "5"s.

- "Easier" meaning that the representation error is smaller.

- Application to classification: find the SVD basis of each of the numbers "1", "2", ...

- Classify a new observation into the set whose SVD basis is most suitable for representing the observation.

Support vector machine

- Supervised learning
- Classification when the classes are not linearly separable
- State-of-the art method of classification today
- Matlab: smvtrain

Decision trees

- Supervised learning: prediction and classification
- Suitable especially when the data contain both numerical and categorical variables
- Then it is difficult to construct a distance function for regression or k-nearest-neighbors etc.
- A decision tree can help

Linear regression

- Supervised learning, continuous valued data
- Simple if there is one unknown variable to be predicted
- Multiple unknown variables: matrix $\mathbf{Y}$ contains them as columns
- $\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{Y}$
- This is a set of equations, solved by Gaussian elimination, or Matlab's "\"

Contingency tables

- This is unsupervised learning
- For categorical data
- Interaction analysis, association analysis, . . .
- Are two phenomena dependent?
- Count the numbers of occurrence and non-occurrence of both of them, and apply a Chi squared test or binomial test
- What is the application area here?

Association rules

- Applicable to both supervised and unsupervised learning
- Applicable to data in which the numerical values are not interesting, but only the presence or absence of a variable in an observation
- Example: market basket data. Observations are the customers, and variables are the products. We are not interested in how many pieces of each product the customer is buying
- Applicable when the data is quite sparse (each customer only buys a small amount of different products, compared to the total number of products available). Do you have such data?

- Gather the *frequent itemsets* in the data: sets of products that are typically bought together

- Construct rules "if a customer is buying bread and diapers, he will also buy milk". Observing subset of variables implies that another subset of variables will be observed too.