Mean shift Algorithm is a mode-seeking algorithm

⇒ Related application: clustering –

PDF

K-means:

- obtain (guess) initial distribution of cluster centers
- for each data point: identify the closest* cluster center
- each cluster center is replaced by the coordinate-wise average of all data points, that are closest to it

⇒ REPEAT (until convergence)

greedy step

[Convergence is guaranteed but the found minimum depends on the initialization]

GREEDY ALGORITHM / SEARCH:

is a strictly local optimization

Advantage: speed

Disadvantage: No brain

* eucledean

Book: elements of statistical learning
auflage 2 ver 10 . pdf

Two very common (dis-) similarity criteria for clusterings is the "within" or "intra" cluster distance $W(C)$ (⇒ Eq. 14.28) and the "between" or "inter" cluster distance $B(C)$ (⇒ Eq. 14.29)

→ the k-Means algorithm minimizes the within-cluster distances (greedily) (→ Eq. 14.31 – 14.33)

↑ local minimum

Question: How can we determine a reasonable parameter k?

⇒ Low-dimensional data set: Look at the data

⇒ High-dimensional data set: Need a mathematical criterion

↓

- K-means minimizes $W(C)$
- $W(C)$ decreases for increasing k, for $k = N$: $W(C) = 0$

„every point is in its own cluster"

- one trick to determine K is Tibshirani's "gap statistics":

stop at the k where

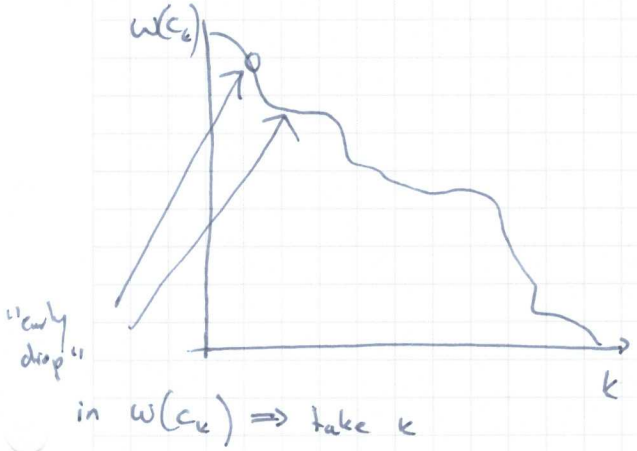
$$G(k) \geq G(k+1) - s'_{k+1}$$

where $G(k) = \log(W(C_k)) - \log(W(C_1))$

↑ negative number

s'_{k+1} is the standard deviation of the outcomes for $k+1$ clusters

Pattern Analysis (3) 03.05.14



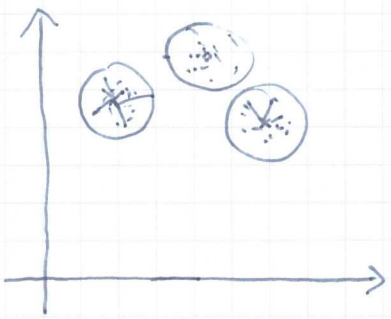
Book reference
↝ page 520

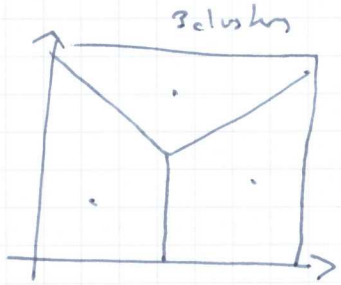
in $w(c_k) \Rightarrow$ take k

54:13

- A second trick is to create a reference curve from the uniform distribution : consider $w(c_k^{uniform})$

$\Rightarrow$ compute the ratio of $w(c_k)$ over $w(c_k^{uniform})$ and pick the minimum (or an early minimum).





good match → relatively low intra cluster distance

Pattern Analysis (4) / (1) 03.05.19 / 08.05.19

Example for agglomerative clustering: Felzenszwalb & Huttenlocher
"Efficient Graph-based Image Segmentation"

08.05.19

Clustering –

[Paper] Ng, Jordan, Weiss: "On Spectral Clustering" –

Algorithm: Given points $S = \{\vec{s}_1, \dots, \vec{s}_n\}$, $\vec{s}_i \in \mathbb{R}^\ell$

1.) Compute affinities $A \in \mathbb{R}^{n \times n}$, $A_{ij} = e^{\frac{-\|s_i - s_j\|_2^2}{2\sigma^2}}$, $A_{ii} = 0$ (diagonal is zero)

2.) Let $D = \text{diag}\left(\sum_j A_{1,j}, \sum_j A_{2,j}, \dots, \sum_j A_{n,j}\right)$ of the row affinities,

and set $L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

Graph Laplacian

3.) Compute eigendecomposition of L

Let $X = (\vec{x}_1\ \vec{x}_2\ \vec{x}_3 \dots \vec{x}_k)$, $X \in \mathbb{R}^{n \times k}$ denote a matrix consisting of the k largest eigenvectors $\vec{x}_1, \dots, \vec{x}_k$ of L

"eigenvectors associated with the largest eigenvalues"

4.) Normalize X: $y_{ij} = \frac{x_{ij}}{\left(\sum_j x_{ij}^2\right)^{1/2}}$

5.) Perform k-means clustering with k centers on the points that are the rows of y

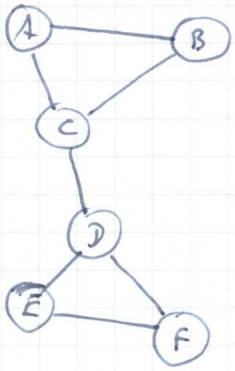
6.) ⇒ The points in the original space are assigned to the k clusters of their transformed version in y.

Pattern Analysis (2) 08.05.19

Quick, informal note on the Graph Laplacian:

A tool that links graph theory, linear algebra and probability theory.

Let's look at a graph:



Popular graph algorithms:

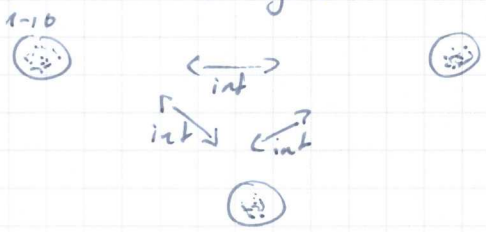
Maximum Flow / Shortest Path /.../ between two nodes?

The set of mathematical tools, that can be used on a graph is somewhat limited: what exactly should we calculate on a list of nodes and edges?

⇒ Spectral Graph theory opens a door to linear algebra, by treating a graph's adjacency matrix as an actual matrix in an algebraic sense ⇒ we can - for example - compute the eigenvalues to find out invariants about the underlying graph

→ A useful representation of the adjacency matrix is the Graph Laplacian L

Pattern Analysis (3) 08.05.19



Remarks on the "ideal case" (sec. 3.1):

- The set of eigenvalues of a block matrix $\hat{L} = \begin{pmatrix} L^{(11)} & & \\ & L^{(22)} & \\ & & L^{(33)} \end{pmatrix}$

is identical to the set of $\{$ eigenvalues $(L^{(11)})$, eigenvalues $(L^{(22)})$, eigenvalues $(L^{(33)})\}$

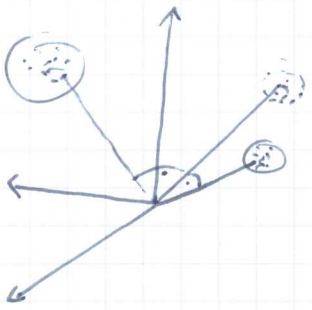
(can be treated independently)

The largest eigenvalue of a graph laplacian is 1, the second largest is lower than 1

$\Rightarrow \hat{L}$ has exactly 3 eigenvalues that are 1. These are the 3 largest ones, that we are looking for.

The associated 3 eigenvectors are orthogonal and thus span a 3-dimensional sub-space.

Note, however, that the exact choice of eigenvectors is not unique



Pattern Analysis ④ 08.05.19

The points are clustering around the rotation axes, that span the sub-space. These axes are spanned by 90° angles between them. Therefore k-Means can easily cluster the points in that subspace.

Sec 3.2, clusters that lie close to one another

The distribution of eigenvalues indicates, how well the data structures into k clusters: consider the eigenvalue gap

$$\delta = |\lambda_k - \lambda_{k+1}|$$

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4 \geq \lambda_5 \geq \dots \geq$$
$$1 \quad 1 \quad 1 \quad 1-5$$

eigenvalue gap

Cheeger constant:

$$h(S_i) = \min_{\bar{I}} \frac{\sum_{\substack{j \in \bar{I} \\ k \notin \bar{I}}} A_j}{\min\left\{\sum_{j \in \bar{I}} d_j^{(i)}, \sum_{k \notin \bar{I}} d_k^{(i)}\right\}}$$

"between cluster distance"

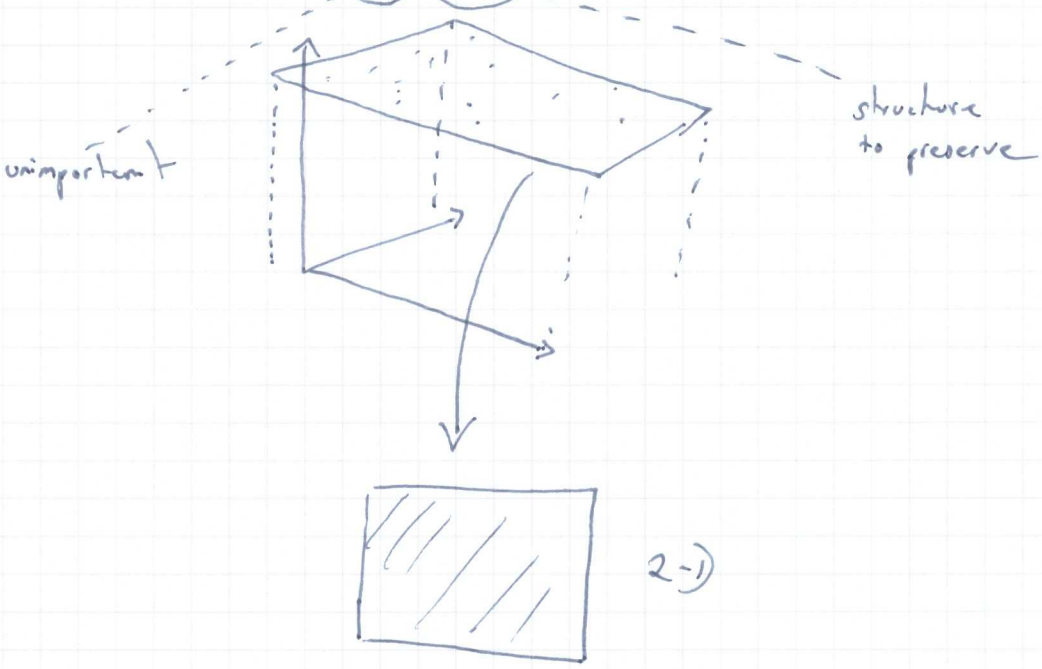
index subset

Pattern Analysis (1) 17.05.19

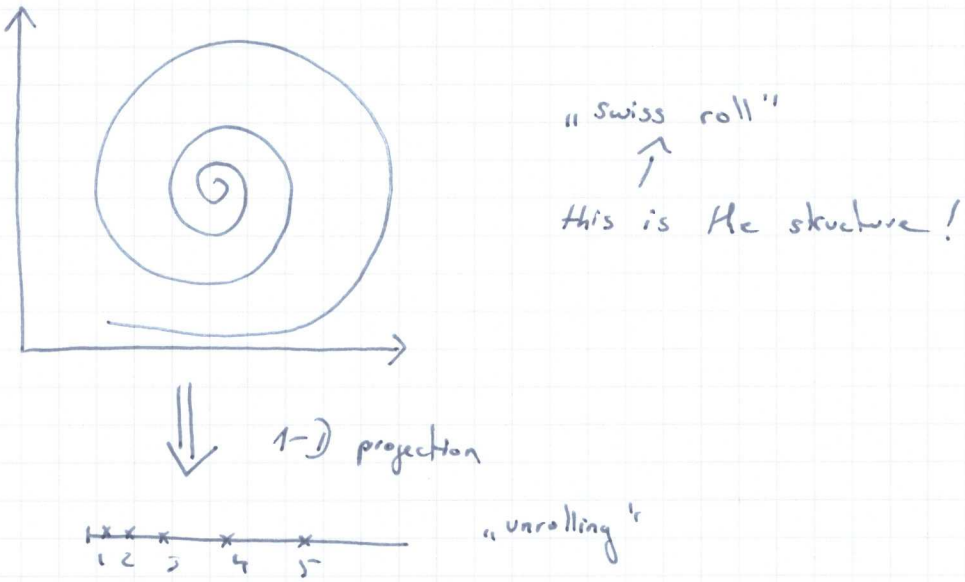
Manifold Learning –

Purpose / Goal: Reduce Dimensionality of the data, while preserving its structure.

Example: consider a noisy plane in 3-D:



2nd example:

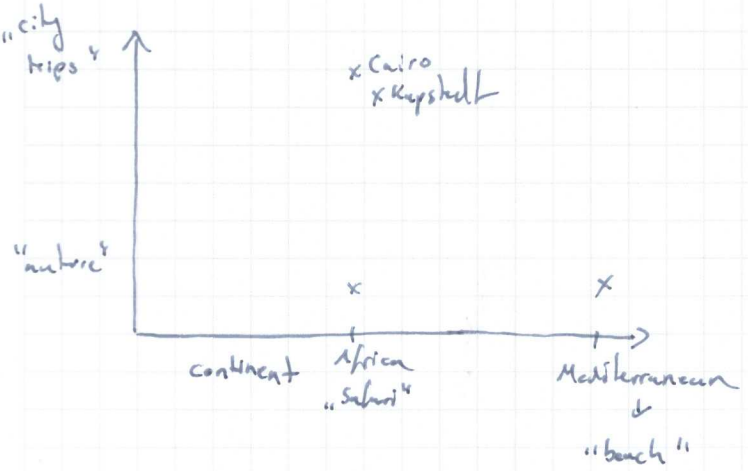


„swiss roll"
↑
this is the structure!

Pattern Analysis (2) 17.05.19

In many real world tasks the structure to preserve and the low-dim manifold is not super-sharply defined.

Consider for example the manifold of tourist pictures



„Semantic Image Classification"
From millions of pixels in millions of images to very few (< 10) dimensions
⇒ rather extreme task

In this class I would like to see manifold learning as the task of finding lower-dimensional representation of structure, essentially by ignoring the noise.

Why should we reduce the dimensionality?
⇒ to avoid the "curse of dimensionality"

Curse of Dimensionality: Distance metrics tend to "wash out" in higher dimensional spaces, i.e. to lose their discriminative power.

To see this, consider N features $\vec{x}_1, \dots, \vec{x}_n \in \mathbb{R}^d$, where

$0 \leq x_{i,k} \leq 1$
↑ kth element of the ith vector

Pattern Analysis (3) 17.05.19

To capture a fraction r of uniformly distributed features, we need to consider $r \cdot V$ of the volume V of the feature space.

This corresponds to a hypercube with d dimensions and edge length $e_d(r) = r^{\frac{1}{d}} = \sqrt[d]{r}$

For example, to find 1% of the features in a 10-dim space, $e_{10}(0,01) = (0,01)^{\frac{1}{10}} = 0,63$

to find 10% in 10-dim space:

$$e_{10}(0,1) = 0,1^{\frac{1}{10}} \approx 0,8$$

The median distance of the nearest neighbor to the origin of a d-dim space with N samples is

$$d(d,N) = \left(1 - \left(\frac{1}{2}\right)^{\frac{1}{N}}\right)^{1/d}$$

$\Rightarrow d(10, 5000) = 0,52$ most points are close to the boundary in some dimension (!)

PCA (Principle Component Analysis)

Find a linear mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^{d'}$, $d' \ll d$

that maximizes the variance in each dimension

Pattern Analysis ④ 17.05.19

Objective Function $J = \sum_{i,j=1}^{N} \left(\underline{\Phi} \cdot \vec{x_i} - \underline{\Phi} \vec{x_j} \right)^T \cdot$

$x \in \mathbb{R}^d$

$\cdot \left(\underline{\Phi} \cdot \vec{x_i} - \underline{\Phi} \vec{x_j} \right) +$

$+ \lambda \left(\underline{\Phi}^T \underline{\Phi} - 1 \right)$

⇒ PCA requires an Eigenvalue decomposition of the covariance matrix. PCA generates a low-dimensional orthogonal basis.

Multidimensional Scaling (MDS)

Goal: To compute a low-dimensional representation of data points where we only know the distances (or dissimilarities) between them.

	Nürnberg	Bejing	Berlin	Mumbai
Nürnberg	–	9000	482,3	8325
Bejing		–	8500	4000
Berlin			–	7900
Mumbai				–

⇒ can we reconstruct a map from the distances?

(→ note, that there are close links to kernel PCA!!)

⇒ see ELSL.pdf, search for "MDS"

Exercise 18.15

Pattern Analysis (5) 17.05.19

Let $X = (\vec{x}_1, \vec{x}_2, \vec{x}_3) \in \mathbb{R}^{d \times N}$ denote the feature vectors (as usual)

Set $B = X^T X$

Let furthermore denote $D^2 = [d_{ij}^2]_{i,j \in \{1,\dots,N\}}$

where $d_{ij}^2 = (\vec{x}_i - \vec{x}_j)^T (\vec{x}_i - \vec{x}_j)$ (Euclidean Distance)
(could be substituted for another metric)

Task: given D^2, compute X

$$d_{ij}^2 = (\vec{x}_i - \vec{x}_j)^T (\vec{x}_i - \vec{x}_j) = \vec{x}_i^T \vec{x}_i + \vec{x}_j^T \vec{x}_j - 2 \vec{x}_i \vec{x}_j$$

Assume that $\vec{x}_1, \dots, \vec{x}_N$ are zero-mean, i.e. $\sum_{i=1}^{N} \vec{x}_i = \vec{0}$

The squared distance matrix is $D^2 = \mathrm{diag}(X^T X) \cdot \vec{1}^T + \vec{1} \cdot \mathrm{diag}(X^T X)^T - 2 X^T X$

where $\vec{1} = (1, \dots, 1)^T \in \mathbb{R}^N$

"Magic matrix" / Centering matrix $C = \left(I - \frac{1}{N} \vec{1} \cdot \vec{1}^T\right)$

⇒ Multiplying D^2 by the centering matrix from left & right

and weighing the result by $-\frac{1}{2}$ yields:

$$-\frac{1}{2} C D^2 C = -\frac{1}{2} \left(I - \frac{1}{N} \cdot \vec{1} \cdot \vec{1}^T\right) \cdot \Big(\underbrace{\mathrm{diag}(X^T X)}_{(1)} \cdot \vec{1}^T + \vec{1} \cdot \underbrace{\mathrm{diag}(X^T X)}_{(2)} - 2 \cdot \underbrace{X^T X}_{(3)}\Big) \cdot \left(I - \frac{1}{N} \cdot \vec{1} \cdot \vec{1}^T\right)$$

Pattern Analysis ⑥ 17.05.19

Term (1): $\left[\left(I - \frac{1}{N}\vec{1}\vec{1}^T\right) \cdot \text{diag}\left(x^T x\right)\right] \cdot \underbrace{\left[\vec{1}^T \cdot \left(I - \frac{1}{N} \cdot \vec{1}\vec{1}^T\right)\right]}_{= \emptyset} =$

$= \ldots = \emptyset$

$\vec{1}^T \cdot I - \frac{1}{N} \underbrace{\vec{1}^T \vec{1}}_{N} \cdot \vec{1}^T$

$= \vec{1}^T - \vec{1}^T = 0$

Term (2): [analogously] $= \emptyset$

Term (3) $\left[-\frac{1}{2}\left(I - \frac{1}{N}\vec{1}\vec{1}^T\right) \cdot \left(-2X^T X\right) \cdot \left(I - \frac{1}{N}\vec{1}\vec{1}^T\right)\right] =$

$= I \cdot X^T - \underbrace{\frac{1}{N}\left(\vec{1}\vec{1}^T\right)X^T}_{\emptyset \text{ because of zero-mean of feature vectors}}\Big) \cdot \left(X - \frac{1}{N} X \cdot \vec{1}\vec{1}^T\right) \Rightarrow \boxed{X^T X = B}$

⇒ Factorize B to obtain X via SVD

$B = U^T \Sigma V$

B is symmetric ... $\Rightarrow \underbrace{U \Sigma^{1/2}}_{= X^T} \quad \underbrace{\Sigma^{1/2} V}_{= X}$

Pattern Analysis ① 24.05.19

Manifold Learning – (continued)

MDS: "PCA on distances / dissimilarities"

$$C = \left(I - \frac{1}{N} \vec{1}^T \vec{1} \right)$$

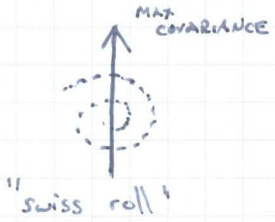
$$SVD\left(\frac{1}{2} C^T D^2 C \right) \Rightarrow U^T \Sigma^{\frac{1}{2}} = X$$

↑ zero out singular values for dim. reduction

Another item to relax from the PCA formulation:

Can we perform a local projection instead of a global one?

Example case (not nicely solvable with PCA/MDS):



[global projection fails]

MDS also has difficulties here:

because the globally applied distances can not distinguish local structures on the manifold.

"ISOMAP": ("Isometric Feature Mapping")

Perform MDS, but on graph distances instead of some global metric like the Euclidean distance.

Pattern Analysis (2) 24.05.19

Graph Distance:

1) Define a local neighborhood, compute Eucl. (or something else) distance to those neighbors ⇒ sparse adjacency matrix

2) Compute all-pairs-shortest-paths to obtain all missing distances

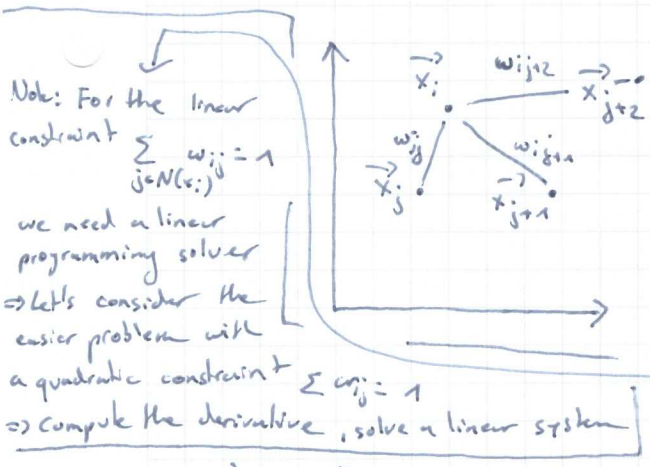
⇒ This is also called a "geodesic" distance

Locally Linear Embedding (LLE)

Idea: 1) Represent every sample as a linear combination of its neighbors,

2) Search for a lower-dimensional representation with the same or a similar linear combination of neighbors

($N(x_i) \hat{=}$ "Neighborhood of x_i")



Note: For the linear constraint $\sum_{j \in N(x_i)} w_{ij} = 1$ we need a linear programming solver
⇒ Let's consider the easier problem with a quadratic constraint $\sum w_{ij} = 1$
⇒ Compute the derivative, solve a linear system

1) Search for weights w_{ij}:

$$\min \sum_i \left\| \vec{x}_i - \sum_{j \in N(x_i)} w_{ij} \vec{x}_j \right\|_2^2$$

(*) subject to $\sum_{j \in N(\vec{x}_i)} w_{ij} = 1$

2) In the lower-dimensional space $\mathbb{R}^{d'}$, $d' < d$:

Solve for $\vec{x}'$ in

$$\min \sum_i \left\| \vec{x}_i' - \sum_{j \in N(\vec{x}_i)} w_{ij} \vec{x}_j' \right\|_2^2 \quad \text{subject to} \quad \frac{1}{N} \sum_i \vec{x}_i' \vec{x}_i'^T = \underline{I}$$

(covariance is identity)

$\sum_i \vec{x}_i' = \emptyset$ (zero-mean)

Pattern Analysis (3) 24.05.19

Let's work on the solution of the first part of the problem:

The objective function is invariant to translations:

$$\sum_{i=1}^{N} \left\| \vec{x}_i - \sum_{j \in N(\vec{x}_i)} \omega_{ij} \vec{x}_j \right\|_2^2 = \sum_{i=1}^{N} \left\| \vec{x}_i + \vec{t} - \sum_{j \in N(\vec{x}_i)} \omega_{ij} (\vec{x}_j + \vec{t}) \right\|_2^2$$

for an arbitrary but fixed i, we set $\vec{t} = -\vec{x}_i$

$$\Rightarrow \left\| \vec{x}_i - \vec{x}_i - \sum_j \omega_{ij} (\vec{x}_j - \vec{x}_i) \right\|_2^2 = \left\| \sum_j \omega_{ij} (\vec{x}_j - \vec{x}_i) \right\|_2^2$$

$$= \left\| \omega_{i1} (\vec{x}_1 - \vec{x}_i) + \omega_{i2} (\vec{x}_2 - \vec{x}_i) + \dots \right\|_2^2 = \left\| M_i \omega_i \right\|_2^2 \quad \text{where}$$

$$\vec{\omega}_i = \begin{pmatrix} \omega_{1i} \\ \omega_{2i} \\ \vdots \end{pmatrix} \quad \text{and} \quad M = \begin{pmatrix} \vec{x}_1 - \vec{x}_i, & \vec{x}_2 - \vec{x}_i, \dots \end{pmatrix}$$

$$\Rightarrow \text{minimize } (M_i \omega_i)^T (M_i \omega_i) + \underbrace{\lambda (1 - \vec{\omega}_i^T \vec{\omega}_i)}_{\text{quadratic constraint}}$$

(Note: $M_{ij} = 0$ if $\vec{x}_j \notin N(\vec{x}_i)$)

$$\frac{\partial}{\partial \omega_i} \left(\omega_i^T M_i^T M_i \vec{\omega}_i + \lambda (1 - \vec{\omega}_i^T \vec{\omega}_i) \right) = 2 M_i^T M_i \vec{\omega}_i - \lambda 2 \vec{\omega}_i \overset{!}{=} 0$$

$$\boxed{M_i^T M_i \cdot \vec{\omega}_i = \lambda \omega_i}$$

Eigenvalue / Eigenvector Problem

($\Rightarrow$ take eigenvector that belongs to the smallest non-zero eigenvalue)

Pattern Analysis (4) 24.05.19

2. step: solve second obj function with the calculated weights $\vec{w}_i$:
for $\vec{x}_i'$ (solution is similar)

Laplacian Eigenmaps

1.) Build an adjacency graph from the feature neighbourhoods
2.) Compute affinities between neighbored nodes ← weights
3.) Perform eigendecomposition of the Graph laplacian of the weights
4.) Low-dim embedding

Step 1: – Everything within fixed (Euclidean?) distance is a neighbor
– The k-nearest samples are neighbors (⇒ k-regular graph)

Step 2: Pick any function that decreases with increasing distance

1 specific proposal: heat kernel

$$w_i = e^{\frac{-\|x_i - x_j\|_2^2}{t}}$$

another (non-differentiable) possibility:
binary affinity $w_i = \begin{cases} 1 & \text{if } x_j \in N(x_i) \\ 0 & \text{else} \end{cases}$

Pattern Analysis (5) 24.05.19

Step 3, theoretical derivation:

(in the low-dim space)

$$\text{minimize} \sum_j \sum_i \left(\vec{x}_i' - \vec{x}_j' \right)^2 \cdot \omega_{ij}$$

$$\text{subject to } \vec{x}^T D \cdot \vec{x} = 1$$

↑ "Volume constraint"

$$\text{where } D = \begin{pmatrix} \sum_i \omega_{1i} & & \emptyset \\ & \sum_i \omega_{2i} & \\ \emptyset & & \sum_i \omega_{3i} \\ & & & \ddots \end{pmatrix}$$

Rewrite the objective function:

$$\sum_{ij} \left(\vec{x}_i' - \vec{x}_j' \right)^2 \cdot \omega_{ij} = \sum_{ij} \left(\vec{x}_i'^2 + \vec{x}_j'^2 - 2 \vec{x}_i' \vec{x}_j' \right) \cdot \omega_{ij}$$

$$= \sum_{ij} \vec{x}_i'^2 \cdot \omega_{ij} + \sum_{ij} x_j'^2 \cdot \omega_{ij} - 2 \sum_{ij} \vec{x}_i' \vec{x}_j' \omega_{ij} =$$

↑ identical

$$= 2 \sum_{ij} \vec{x}_i'^2 \omega_{ij} - 2 \sum_{ij} \vec{x}_i' \vec{x}_j' \omega_{ij} = 2 \cdot \left(\vec{x}'^T D_{\vec{x}'} - \vec{x}' W_{\vec{x}'} \right)$$

$\sum_i \vec{x}_i'^2 \cdot \sum_j \omega_{ij}$ ↑ $W = [\omega_{ij}]$

$= 2 \vec{x}'^T (D - W) \vec{x}'$ ← Graph Laplacian L !

Pattern Analysis ⑥ 24.05.19

Minimize $\vec{x}'^T \Sigma \vec{x}'$ subject to $\vec{x}^T D \vec{x}' = 1$

$$\Rightarrow \frac{\partial}{\partial x}\left(\vec{x}'^T \Sigma \vec{x}' + \lambda\left(1 - \vec{x}'^T D \vec{x}'\right)\right)$$

$$= 2\Sigma\vec{x}' - 2\lambda D\vec{x}' \stackrel{!}{=} 0$$

$$\Rightarrow D^{-1}\Sigma\vec{x}' = \lambda\vec{x}'$$

Pattern Analysis ① 29.05.19

Detour : Gaussian Mixture Models :

$$p(\vec{x}) = \sum_{i=1}^{N} \beta_i \, \mathcal{N}(\mu_i, \Sigma_i; \vec{x})$$

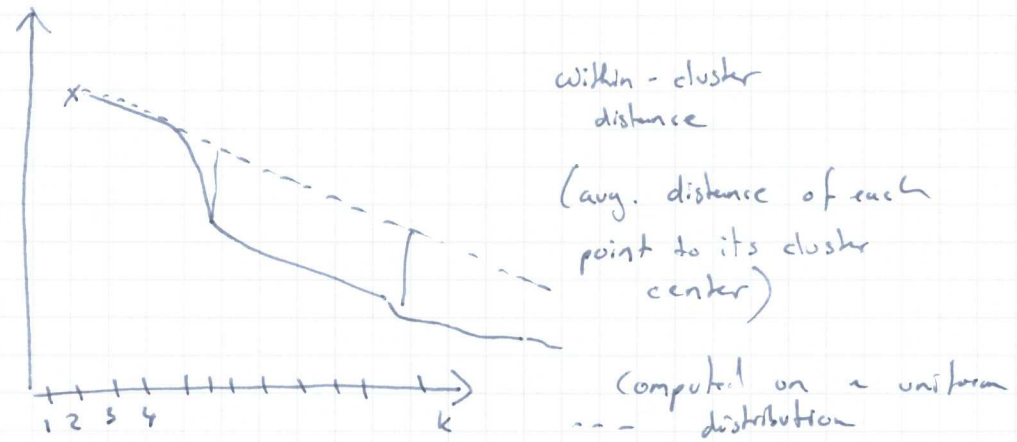
Model selection problem : choose N

weighted sum

params that need to be fitted

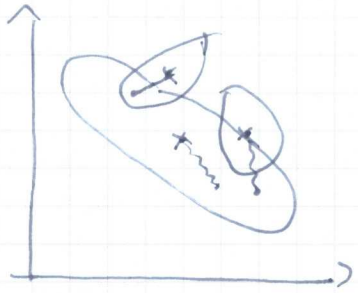
For clustering - let's say k-means clustering - how did we tackle the model selection problem (i.e. select the number of clusters)?

Gap statistics :



Mean shift Algorithm

Gradient Ascent



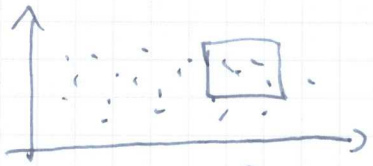
Pattern Analysis ②

29.05.19

$$\frac{\sum_{x_i \in K(x)} (\vec{x}_i - \vec{x})}{\sum \dots}$$ ← mean shift step

"hyper parameter"

gradient ascent is performed with respect to some kernel window



Number of samples

⇒ continuous PDF

density estimation problem

Parzen-Rosenblatt-window

$$p(\vec{x}) = \frac{\sum_{i \in N(x)} k_\lambda (\vec{x} - \vec{x}_i)}{N \cdot V}$$ ← binary

Cross-Validation can be used to determine the quality of the chosen Parzen window size λ for density estimation

Pattern Analysis (3) 29.05.19

Cross validation:

Remove 1 (or k) samples from the input set.

For a reasonable set of window sizes $S = \{\lambda_1, \lambda_2, \ldots, \lambda_w\}$, evaluate

$1 - \vec{p}_{\lambda_i, \vec{x}_j}(\vec{x}_j) \longleftarrow p(\vec{x})$, computed for window λ_i and without sample $\vec{x}_j$

⇒ repeat for all samples $\vec{x}_1, \ldots, \vec{x}_N$

⇒ Choose the window size λ_i with the smallest interpolation error

---

Model selection problem for manifold learning:

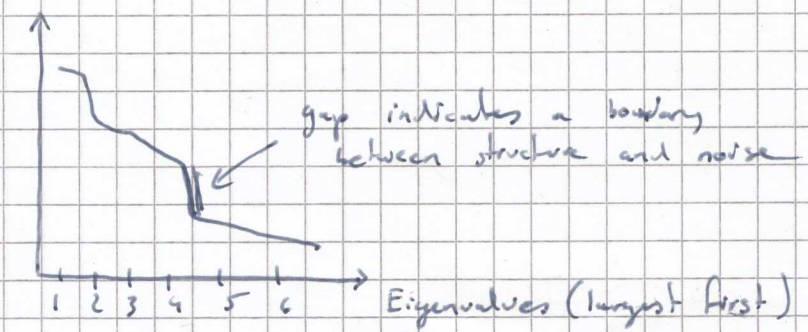
what is a good (intrinsic!) dimensionality of the data?

⇒ All methods, that we considered are based on some eigendecomposition. The distribution of eigenvalues quantifies the quality of the model, similar to the within-cluster distance for the k-means clustering.

A "good" dimensionality is indicated by a gap in the distribution of eigenvalues

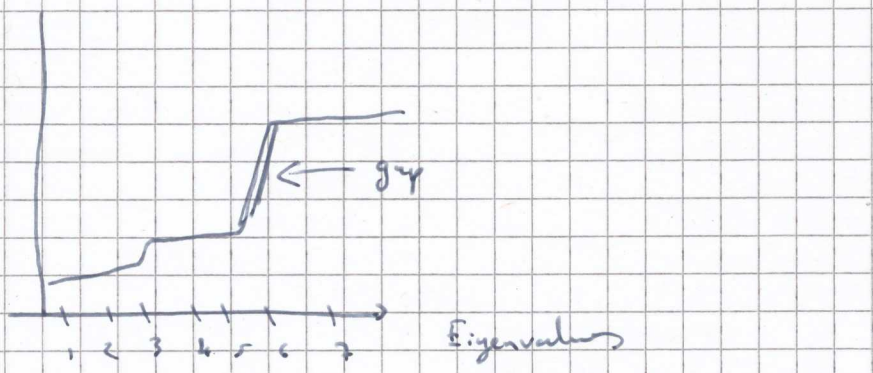
Pattern Analysis ④ 2205.19

For example: PCA:



a we chose the PCA components that contain 99% of the covariance of the data"

Example LE: (Laplacian Eigenmaps)



Random Forests

... as a learning-based tool for feature space partitioning

you can use your partitioned feature space for many different tasks, specifically

- classification, (1)
- regression (2)
- density estimation (3)
- manifold learning (4)
- semi-supervised learning (5)

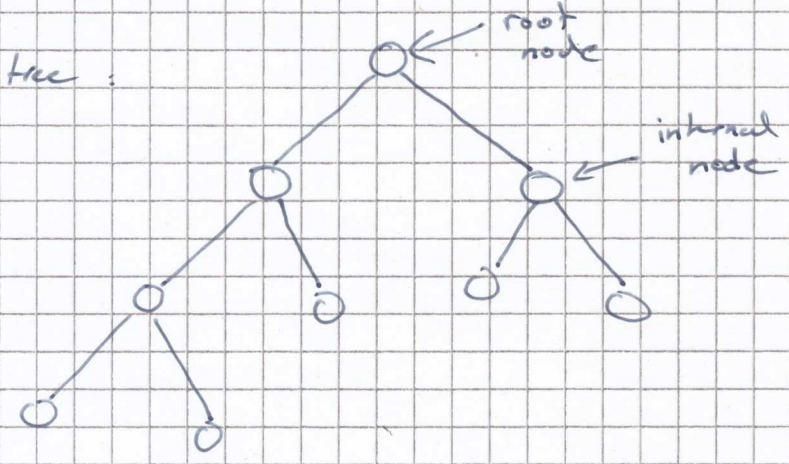
We follow Criminisi, Shotton, Konukoglu:

"Decision Forests: A Unified Framework for (1), (2), (3), (4), (5)

A forest is an ensemble / set of trees.

Decision forest: ensemble of decision trees.

Decision tree:

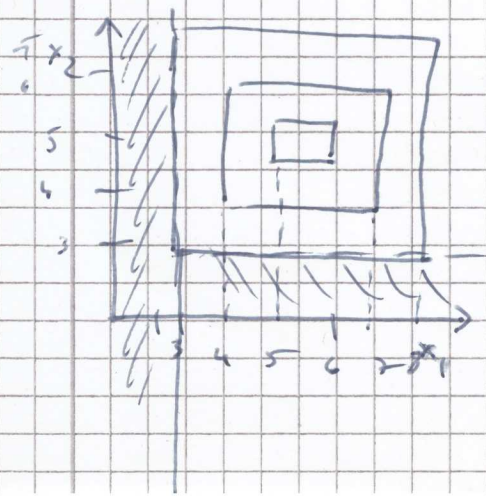


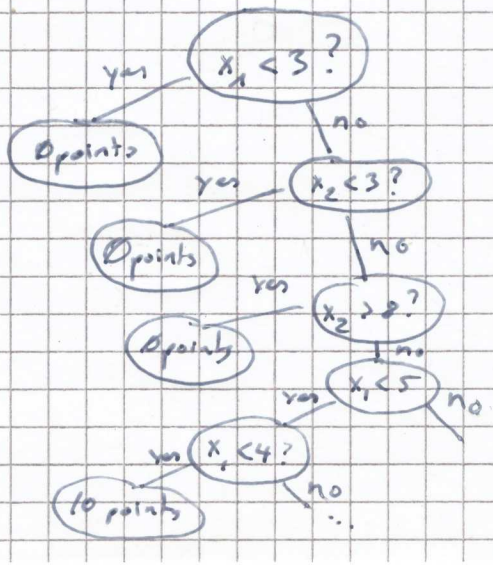
General workflow:

- start at root node
- answer for each internal node with "yes" or "no" to a question
- For "yes" proceed to left child
  For "no" " " right "

Terminal node gives the result.

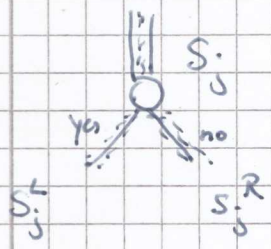
"Square dart!"





Random Forests

Training: Determine the functions ("yes/no questions") at each internal node, and the leaf output.



The question for a good node function is: what is a good "splitting function" for all incomming features

⇒ Maximize the Information Gain for our task at hand

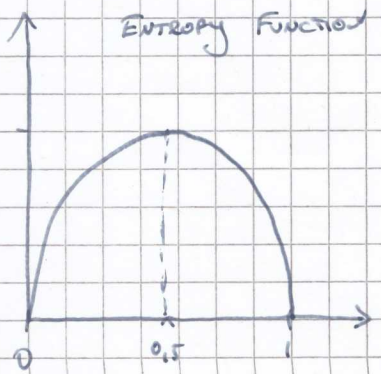
$$I(S_j^i) = H(S_j^i) - \sum_{i \in \{L,R\}} \frac{|S_j^i|}{|S_j^i|} \cdot H(S_j^i)$$

where $H(S_j^i)$ denotes the entropy of set S_j^i

For example, for the case of classification

$$H(S_j^i) = -\sum_{c \in C} p(c) \cdot \log_2(p(c))$$

where C is the set of all class labels, and $p(c)$ denotes the relative frequency of label c in set S_j^i

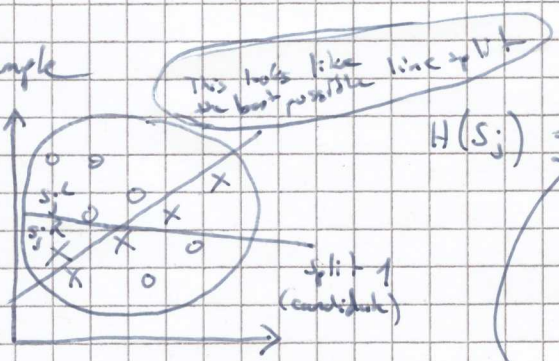


The binary entropy function is lowest if the input is always 0 or always 1 "not surprising"

Translated to our classification task, the entropy is lowest if all class labels are identical in a subset S_j^L or S_j^R

$\Rightarrow$ lower entropy in $S_j^L, S_j^R \Rightarrow$ larger information gain

Example



$$H(S_j) = -\left(\frac{4}{9}\log\left(\frac{4}{9}\right) + \frac{5}{9}\log\left(\frac{5}{9}\right)\right)$$

$p(\circ)$ $p(\times)$

negative

cancelled minuses

Let us look for a good split

The entropies of the left and right subsets of the split are:

$$H(S_j^L) = -\left(\frac{3}{5}\log\left(\frac{3}{5}\right) + \frac{2}{5}\log\left(\frac{2}{5}\right)\right)$$

$$H(S_j^R) = -\left(\frac{1}{4}\log\left(\frac{1}{4}\right) + \frac{3}{4}\log\left(\frac{3}{4}\right)\right)$$

Generally speaking (for classification): we are looking for splitting functions that separate our labelled data as cleanly as possible.

Plain Decision Trees are quite prone to overfitting

(consider a greedy selection of always the best information gain)

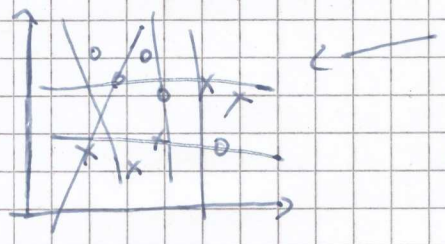
$\Rightarrow$ Breiman's idea: add randomization into the training process

This leads to a "weak learner" $\Rightarrow$ a tree that is not optimal, but "quite ok" / better than guessing

⇒ Repeat this a couple of 100s / 1000s of times to obtain a forest of weak decision trees. Average their weak opinions ⟶ if the weak learners are statistically independent, the ensemble vote converges to the true answer.

Options for randomization:

- Randomization is typically set for one decision tree. We can randomize:
  - the subset of samples for training the tree ("bagging")
  - the subset of feature dimensions for the decision functions
  - The parameters of the decision functions



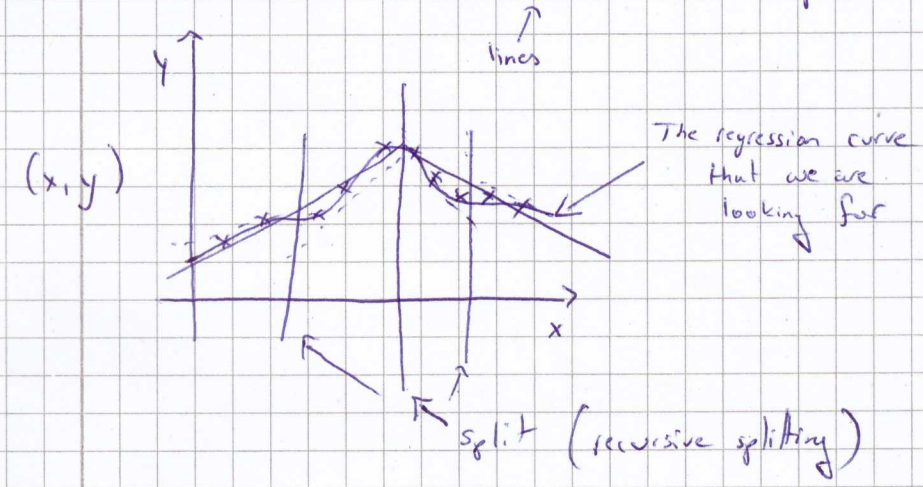
randomly draw 50 splits

⇒ evaluate $\underline{I}$ for each split, use best split

Random Forests

"Regression Forest":

- Redefine information gain for continuous labels y.
- Fit local curve model to the samples in a leaf node



discontinuities at splitting boundaries

⇒ Again, by averaging multiple trees, the discontinuities smooth out, and the final regression curve is "smooth enough" (for our numerical needs)

Information gain for regression: $I(S_j) = H(S_j) - \sum_{i \in \{L,R\}} \frac{|S_j^i|}{|S_j|} H(S_j^i)$

$$H(S) = -\frac{1}{|S|} \sum_{x \in S} \int_y p(y|x) \cdot \log(p(y|x))$$

Let $p(y|x)$ be a Gaussian distribution:

$$p(y|x) = \mathcal{N}\left(y; \bar{y}(x), \sigma_y^2(x)\right)$$

⇒ At each position x we have a Gaussian distribution of possible values (see Bishop's book on how to perform a probabilistic line fit.

"Density Forest"

⇒ use the Random Forest splitting mechanism to estimate a PDF

- Idea: Fit (the samples in) for each leaf node a Gaussian function
  The Gaussian is bounded the boundaries of the leaf (and has discontinuities there)

  ⇒ Average over many trees to smooth out the discontinuities

Adjust the objective function: $I(S_j)$ is again identical to classification case

Entropy $H(S_j)$ for a multivariate Gaussian distribution:

$$H(S) = \frac{1}{2} \cdot \log\left((2\pi e)^d\right) \cdot |\Lambda(S)|$$

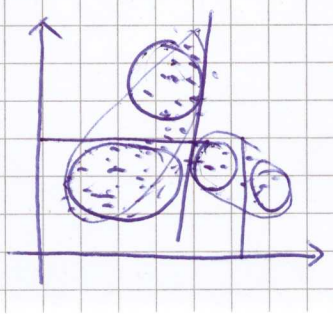
where d is the dimension of the Gaussian and Λ its covariance

⇒ to evaluate $H(S)$ you effectively fit a Gaussian to the samples in S.

Remark: We can plug $H(S)$ into $I(S_j)$ and obtain

$$I(S_j) = \log\left(|\Lambda(S_j)|\right) - \sum_{i \in \{L,R\}} \frac{|S_j^i|}{|S_j|} \cdot \log\left(|\Lambda(S_j^i)|\right)$$

⇒ Goal: Split to obtain compact Gaussian on each side



"Manifold Forest"

Goal: Manifold Learning

side note:
A parametric model has to be finite

Idea: Fit a density forest to the data
Use the "uncertainty" of the splitting boundaries across the trees to determine how tightly two data points are coupled
↑
this is an affinity!
Apply Laplacian Eigenmaps on these affinities

Affinity Model

Let $l(\vec{v})$ denote the leaf partition function that assigns each feature vector to a leaf.

Let k denote the number of training samples, then affinity in tree t between points v_i and v_j is

$$w_{ij}^{+} = e^{-Q^{+}(v_i, v_j)},$$

Let $\vec{d}_{ij} = \vec{v}_i - \vec{v}_j$

where Q^{+} is a distance function.

Choices for Q:

- Mahalanobis affinity: $Q^{+}(v_i, v_j) = \begin{cases} \vec{d}_{ij}^{\,T} \left(\Lambda^{+}_{l(\vec{v}_i)}\right)^{-1} \vec{d}_{ij} & \text{if } l(\vec{v}_i) = l(\vec{v}_j) \\ \infty & \text{otherwise} \end{cases}$

- Gaussian affinity ...

- Binary affinity:

$$Q^{+}(v_i, v_j) = \begin{cases} 0 & \text{if } l(v_i) = l(v_j) \\ \infty & \text{otherwise} \end{cases}$$

Find Affinity $W = \frac{1}{T} \cdot \sum_{t=1}^{T} \omega^t$

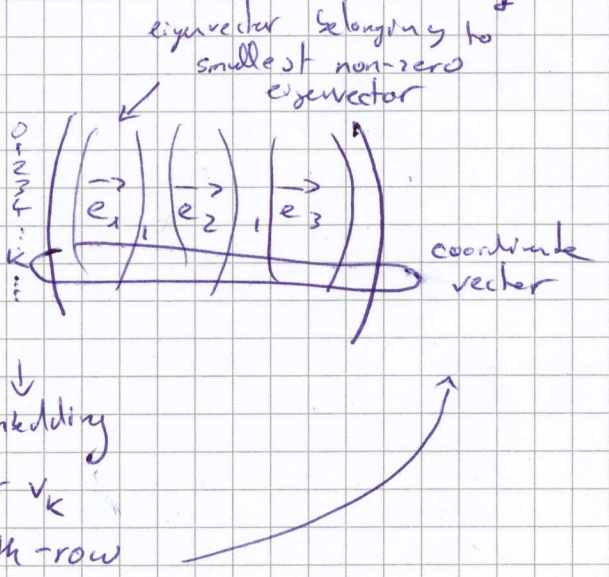
⇒ Compute Laplacian Eigenmaps on W

$L = I - \Upsilon^{-\frac{1}{2}} W \Upsilon^{\frac{1}{2}}$, where $\Upsilon_{\blacksquare} = \mathrm{diag}\left(\sum \omega_{1j}, \sum_j \omega_{2j}, \ldots, \sum_j \omega_{Nj}\right)$

↑ Graph Laplacian

⇒ Eigendecomposition

obtain lower-dimensional embedding:



Random Forests: A flexible tool to describe sample neighborhoods

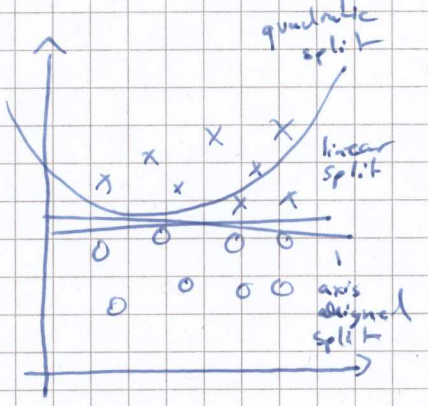
Single Tree: performs a partitioning

Ensemble of trees: "softens" the partitioning from 0-1 labels to continuous probabilities.

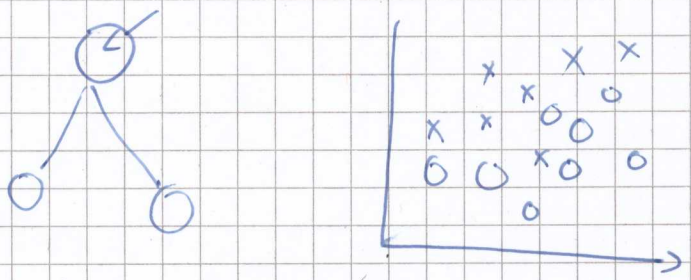
⇓

Learned from the data, depending on the tree / forest parameters

- depth of a tree
- number of trees
- early abortion criteria when growing the tree (not discussed)
- Randomization parameters: [- number of splitting functions to test] ← always
  - % of samples to select
  - number of feature dimensions to select } optional
- Parametric form / type of the splitting functions



Tree depth and the number of trees are related when it comes to overfitting / underfitting



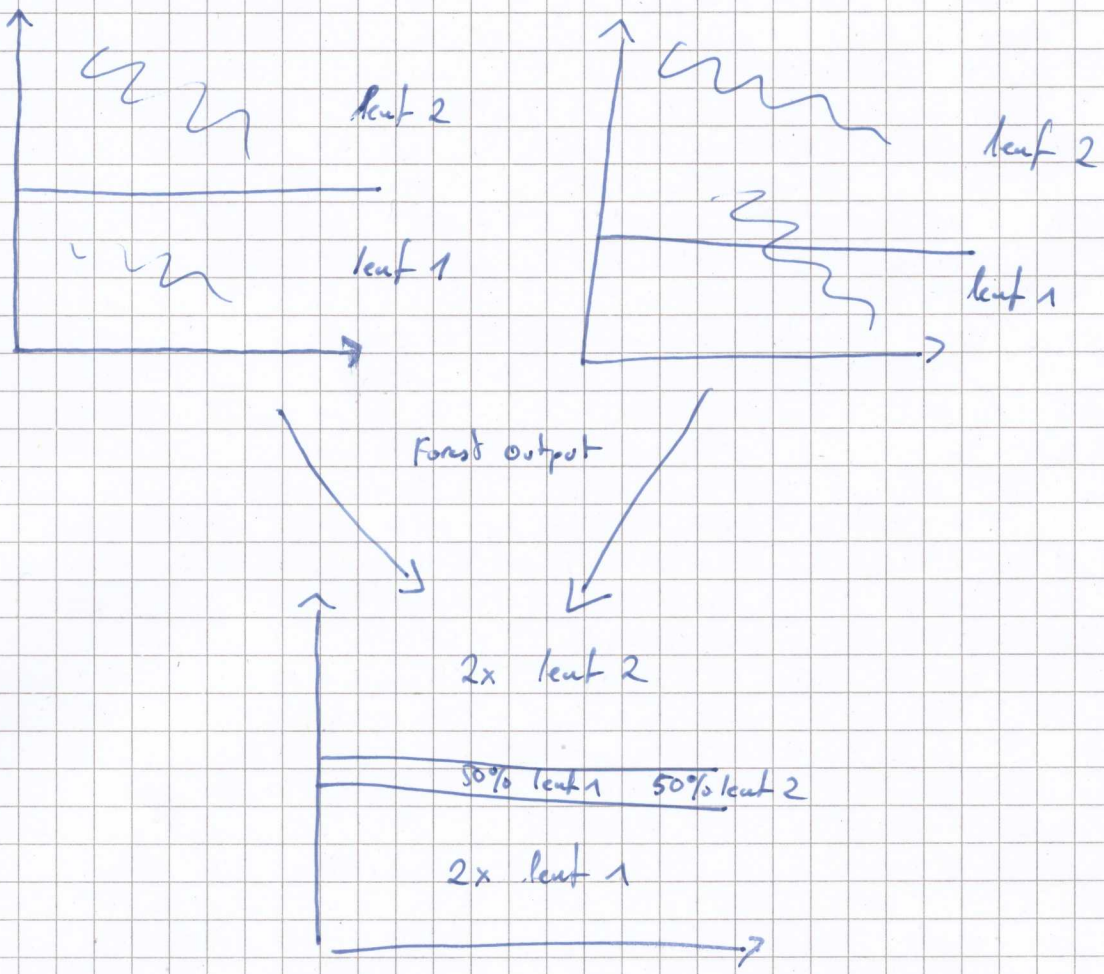
Height 1: invitation to underfitting (as long as our splitting function is simple – which it always is)

Many trees: blur the decision boundary

→ Height 1 and MAX_NUM of trees maximizes the underfitting

Height log N (for N samples): roughly each sample has its own node → overfit

1 Tree: maximally sharp decision boundary maximizes overfitting

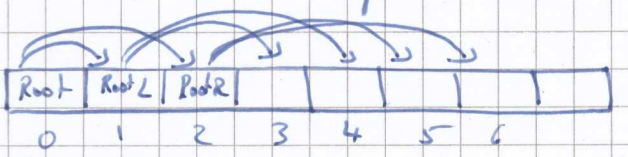


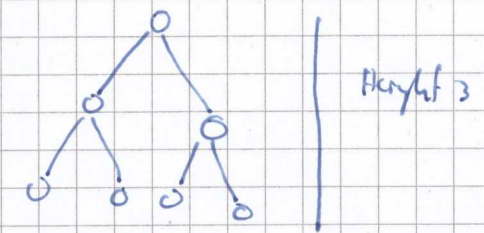
Generally deeper trees also need larger forests to counter overfitting ④

> Good question from audience:
> If we have infinite trees, won't the errors introduced by randomization cancel out and lead to overfitting again?

④ How to detect overfitting: high training performance, low test performance

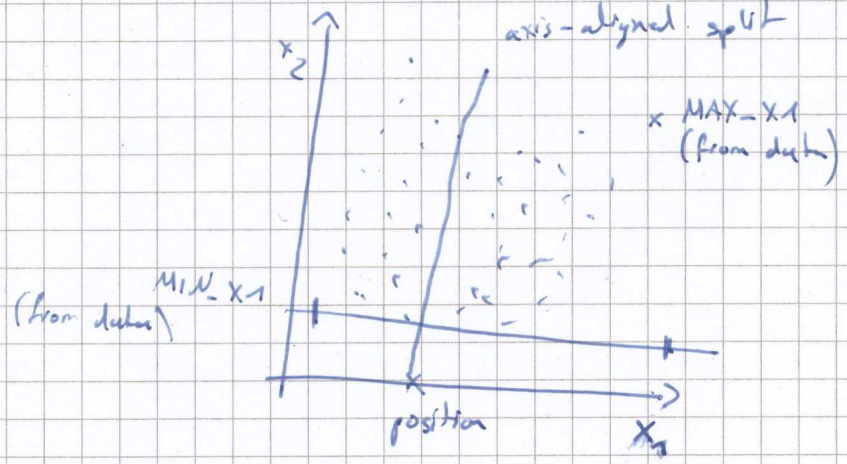
Few notes on the implementation





$2^3 - 1$ nodes

axis-aligned-split
- int dimension [x_1 / x_2]
- double position [some value]



Sample decision functions:

randomly draw uniformly distributed values for "dimension" and "position" within their respective value range

How many decision functions do we need to test in total for training a single tree?

Height (H=3) no early stopping

Test per node C = 500 candidate splits

⇒ sample $2^{(H+1)} \cdot C \cdot 2$ parameters

$3 \cdot 500 \cdot 2 = 3000$

⇒ Determine which split is best: (for classification)

for ext. extremal split

Number of samples N is constant per tree level

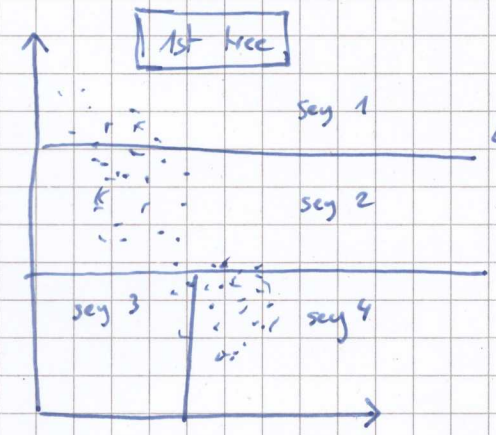
$$N \cdot (H-1) \cdot C = N \cdot 1000$$

=> For, say, 5000 trees costs grow by a factor of 5000, but each tree is independent => parallelize: each tree can go on each node

---

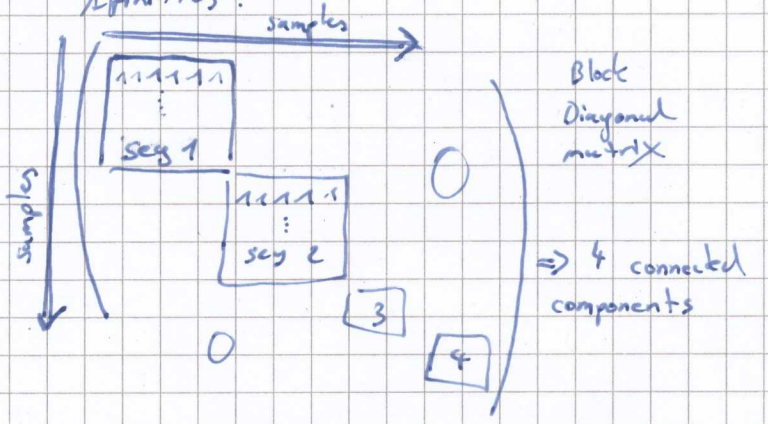
A brief sketch on the Manifold Forests:

A manifold forest is Laplacian Eigenmaps that use the partition function of a density forest to define its affinities



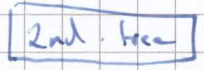
split from 1 tree:

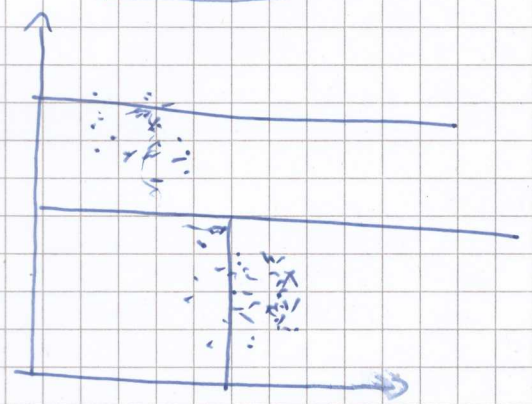
Affinities:

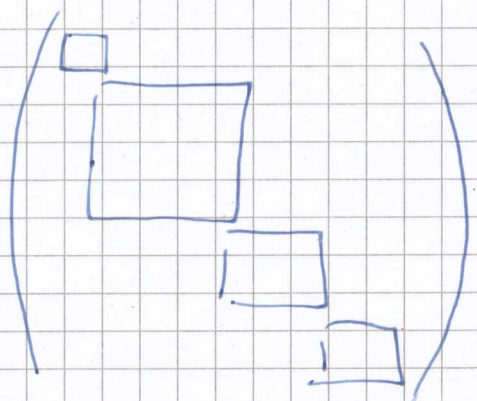


Block Diagonal matrix

=> 4 connected components

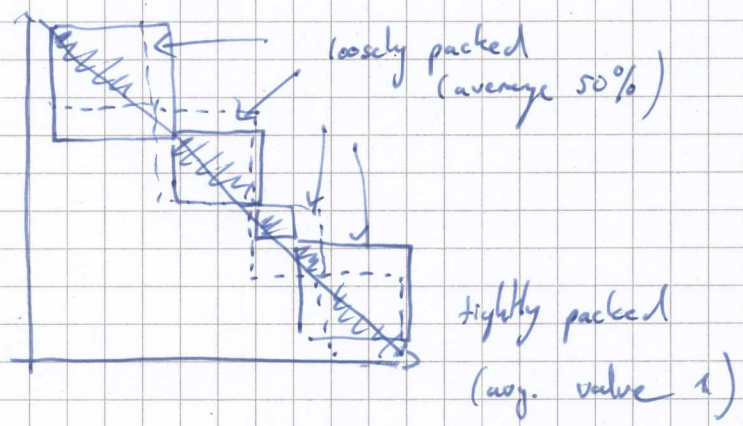






=> again 4 components

Forest Output: Average



The loosely packed links can be stretched a little bit when looking for a lower-dimensional manifold.

Nice: ☺ The "rigidness" is deduced from the variations in the tree training

Probabilistic Graphical Models

- Hidden Markov Models (HMM)
- Markov Random Fields (MRF)

Introduction: ermongroup.github.io / cs228-notes /

Goal: to effectively and efficiently work with a huge joint probability function $p(x_1, x_2, x_3, \dots, x_n)$

A head-on model for $p(x_1, \dots, x_n)$ is almost certainly intractable, consider for example the space of all spoken sentences or the space of all pictures

⇒ A key component in a probabilistic model are the independence assumptions

$p(A, B) = p(A) \cdot p(B)$ iff A and B independent

A compact Bayesian network is a distribution in which each factor depends only on a small number of "ancestor variables" X_{A_i}:

$$p(x_1, \dots, x_n) = p(x_1) \cdot p(x_2 | x_1) \cdot p(x_3 | x_2, x_1) \cdot \dots \cdot p(x_n | x_{n-1}, x_{n-2}, \dots, x_1)$$

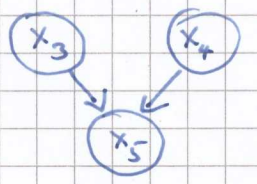
(Chain rule for Bayes formula)

$$p(x_i | x_{i-1}, \dots, x_1) = p(x_i | x_{A_i})$$

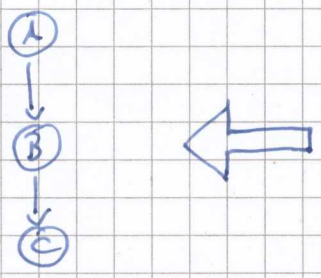
(for example $x_{A_5} = \{x_3, x_4\}$)

Graphical representation:

Express a conditional probability as directed edges in a graph, where the variables $x_1, \dots, x_n$ are the nodes



3-node structures to study independence:



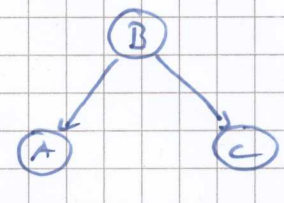
If B is observed, then $A \perp C \mid B$, that is A and C given B are independent. Conversely, if B is not observed, then $A \not\perp C \mid B$.

$p(C \mid B)$ $\qquad$ $p(B \mid A)$

Because C depends only on B. If B is already given, the $p(B \mid A)$ doesn't play any role for C and hence A and C are independent

Pattern Analysis (3) 14.06.19

Common Parent

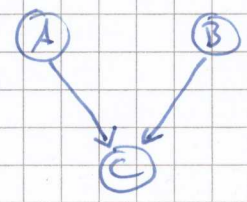


independent

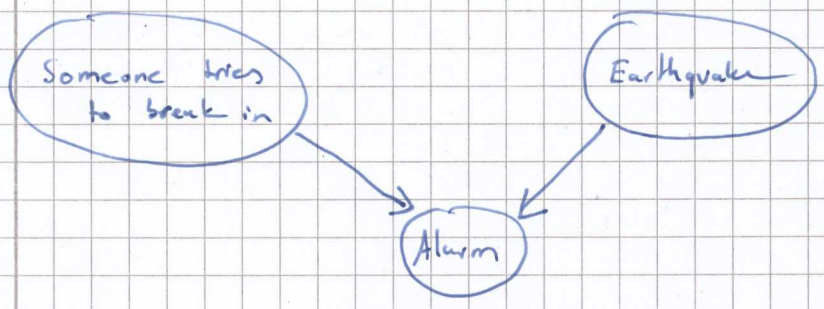
If B is observed, then $A \perp C \perp B$. If B is observed then $A \not\perp C$

Because: B holds all information that affects A and C

'V' structure

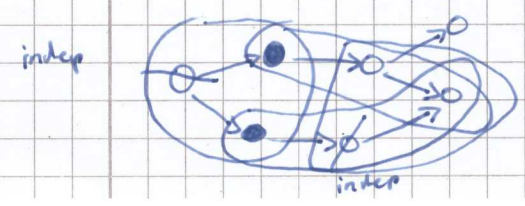


If we know c, then $A \not\perp B$. Conversely, if c is unobserved, then $A \perp B$. "Explaining away"



Q, W are d-separated when variables O are observed if they are not connected by any active path

Active path: A path that consists of triplets, where for each triplet the variables are dependent.



19.06.15

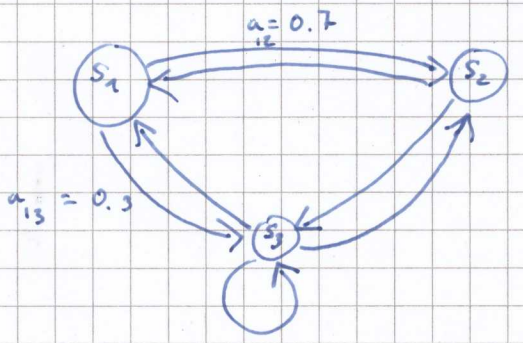
Hidden Markov Model (HMM)

Probabilistic graphical model. The graph is directed.

Markov assumption: $P(q_t = S_i \mid q_{t-1} = S_j, q_{t-2} = S_k, \dots)$

$= P(q_t = S_i \mid q_{t-1} = S_j)$

each conditional probability consists of only one ancestor



nodes in HMM language: "states"

edges: "state transitions"

an edge induces a conditional probability, for example consider

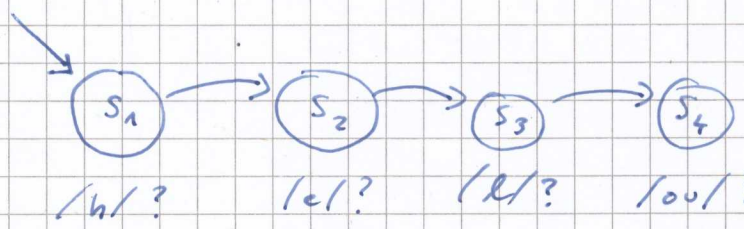
$P(q_t = S_2 \mid q_{t-1} = S_1)$

Most popular application: speech processing

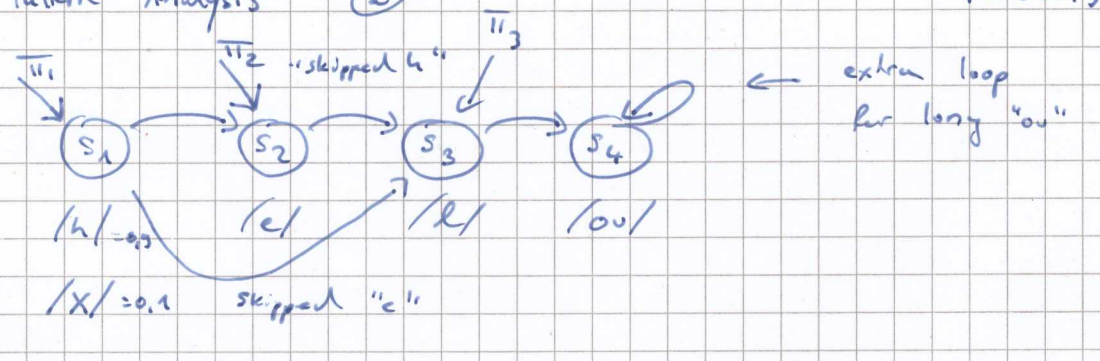
More general: sequential data.

Example: Model a word with a HMM

/h/ /e/ /l/ /ou/ : phonemes ("sound atoms")



Ambiguities in pronounciation (dialects, accents, slang) should also be modelled: we introduce additional possibilities for the word of interest (here: "hello") in the graph



State transition matrix: **A**

Output probability matrix: **B**

$b_i(o_j)$ is the probability to observe/produce in state i the symbol o_j.

$b_1(o_j = /h/) = 0.9$ $\Sigma = 1$

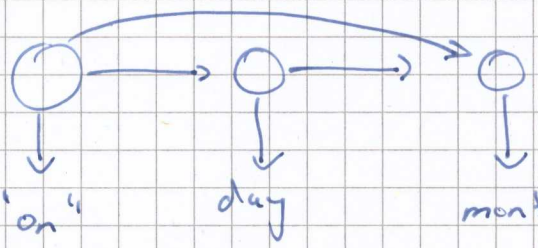
$b_1(o_j = /x/) = 0.1$

Starting probabilities $\pi_1, \ldots, \pi_Q$ (max number of states)

Overall application: Ticket booking system

Construct a dialogue ⇒ prior for the content of the next sentence

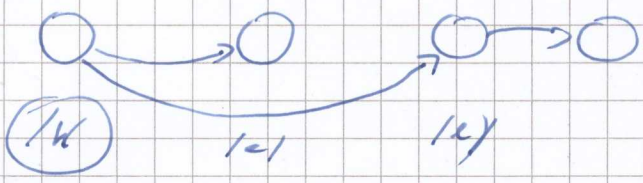
Absolute date:



"Hidden" is the actual state: We only observe the phonems (or more generally: the output symbols), and are interested in the question how well a HMM matches that sequence, but we don't care about the exact visited states.

$$p(\langle o_1, \dots, o_T \rangle, \langle s_1, \dots, s_Q \rangle) =$$
$$p(\langle s_1, \dots, s_Q \rangle) \cdot p(\langle o_1, \dots, o_T \rangle \mid \langle s_1, \dots, s_Q \rangle) = \underline{\underline{p(s_i \mid s_{i-1})}}$$

Markov property: Joint probabilities can be factorized into small terms, because every probability conditioned on states depends only on the last state



Task 1: How to recognize a word?

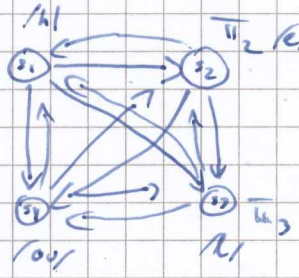
Let $\lambda = (A, B, \vec{\pi})$ denote our HMM for a specific word (e.g. "hello")

$\Rightarrow$ we seek $p(\langle o_1, \dots, o_T \rangle \mid \lambda_{\text{Hello}})$

$\Rightarrow$ We need to marginalize out all hidden state sequences $\langle s_1, \dots, s_Q \rangle$

Brute force: Start at π_{i_1}

$$\sum_{i_1=1}^{Q=4} \pi_{i_1} \cdot b_{i_1}(o_{i_1}) \cdot \sum_{i_2=1}^{Q} a_{i_1 i_2} \cdot b_{i_2}(o_{i_2})$$



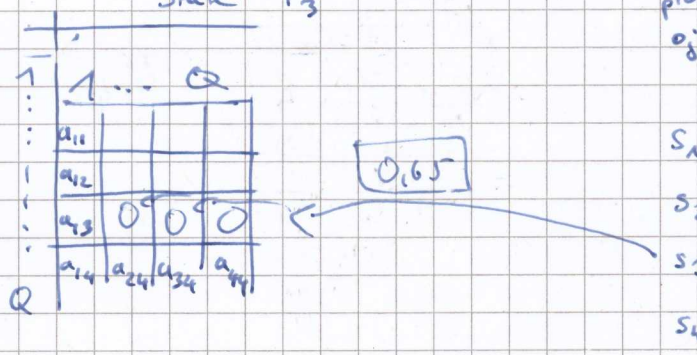
$$b_4(o) = \begin{pmatrix} 0.01 \\ 0.07 \\ 0.08 \\ 0.9 \end{pmatrix} \begin{matrix} /h/ \\ /e/ \\ /l/ \\ /o/ \end{matrix}$$

$$\cdot \left(\sum_{i_3=1}^{Q} a_{i_2 i_3} \cdot b_{i_3}(o_{j_3}) \cdot \left(\sum_{i_4=1}^{Q} a_{i_3 i_4} \cdot b_{i_4}(o_{j_4}) \right) \right)$$

Dynamic Programming

Due to Markov property we only need to cache the probabilities of the previous state

create a list of state numbers and probabilities for producing o_{j_4} (1 or 1)



HMM

Given: sequence of samples / observations

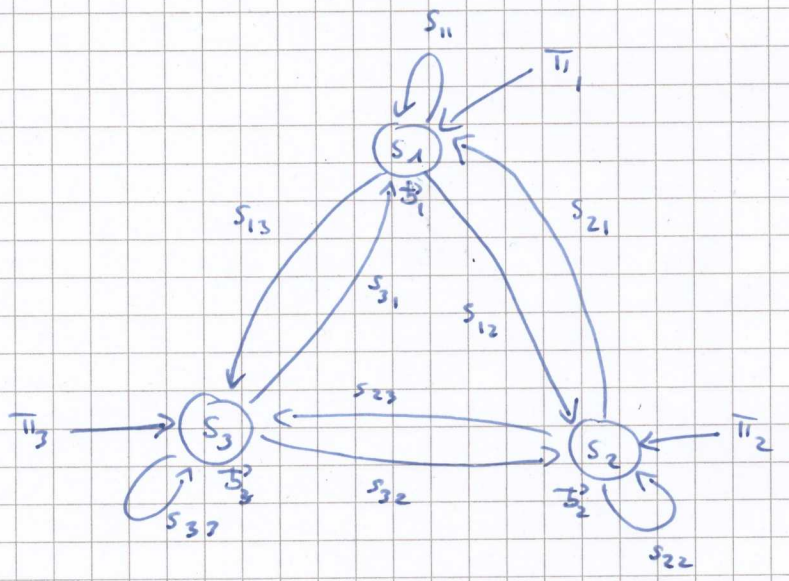
$\langle o_1, \dots, o_n \rangle$

Desired goal: model for representing $\langle o_1, \dots, o_n \rangle$, for example to compute a probability, that $\langle o_1, \dots, o_n \rangle$ describes a certain label (speech: a word from the vocabulary)

Rabiner: "A Tutorial on Hidden Markov Models with selected ..."

Approach: Use a probabilistic model, expressed in a directed graph.

3-state example sketch:



Length of input sequence: N

# of states: M — set of states $S = \{s_1, \dots, s_M\}$

# of symbols in alphabet: V

$\vec{\pi}$: PDF of starting probabilities

$A = [a_{s_i s_j}]$ Transition probabilities $A \in \mathbb{R}^{M \times M}$

$B = [b_{s_i}(o_j)]$ output probabilities $B \in \mathbb{R}^{M \times V}$

"Hidden" is the sequence of states for matching of $\langle o_1, \dots, o_n \rangle$

The HMM models the joint probability $p(\langle o_1, \dots, o_n \rangle, \langle s_1, \dots, s_n \rangle)$

The Markov property implies that the $p(s_i \text{ at time } t \mid s_j \text{ at time } t-1)$

$$= p(s_i \text{ at time } t \mid s_j \underbrace{(t-1)}_{\text{"at time } t-1\text{"}}, s_k (t-2), \dots, s_2 (1))$$

that is, the current state only depends on the previous state and not on the whole previous path (or future path)

$\Rightarrow$ this makes inference tractable

3 Methods to work with HMMs

1.) Matching: find $p(\langle o_1, \dots, o_n \rangle \mid \lambda)$: Forward Algorithm
Backward Algorithm
$\lambda = (A, B, \vec{\pi})$

2.) Most likely state sequence for a given observation

$$\underset{\langle s_1, \dots, s_n \rangle}{\operatorname{argmax}} \; p(\langle o_1, \dots, o_n \rangle, \langle s_1, \dots, s_n \rangle)$$

Viterbi algorithm

3.) Training: find $\lambda = (A, B, \vec{\pi})$ given some training samples

Baum-Welch-Formula(e)

Task 1.) $p(\langle o_1, \dots, o_n \rangle, \langle s_1, \dots, s_n \rangle) =$

$p(\langle s_1, \dots, s_n \rangle) \cdot p(\langle o_1, \dots, o_n \rangle \mid \langle s_1, \dots, s_n \rangle)$

constant for different inputs

goal: $p(\langle o_1, \dots, o_n \rangle)$

$$p(\langle o_1, \dots, o_n \rangle) = \sum_{s_1=1}^{M} \sum_{s_2=1}^{M} \sum_{s_3=1}^{M} \dots \sum_{s_n=1}^{M} \left(\pi_{s_1} \cdot b_{s_1}(o_1) \cdot a_{s_1 s_2} \cdot b_{s_2}(o_2) \cdot \dots \right.$$

N-digit number with base M

$$\left. \dots a_{s_{n-1} s_n} \cdot b_{s_n}(o_n) \right)$$

Naive Approach is too expensive (exponential), but there is a dynamic programming solution

$$\sum_{s_1=1}^{M} \pi_{s_1} b_{s_1}(o_1) \cdot \left(\sum_{s_2=1}^{M} a_{s_1 s_2} \cdot b_{s_2}(o_2) \cdot \left(\sum_{s_3=1}^{M} \dots \left(\sum_{s_n=1}^{M} a_{s_{n-1} s_n} b_{s_n}(o_n) \right) \right) \right)$$

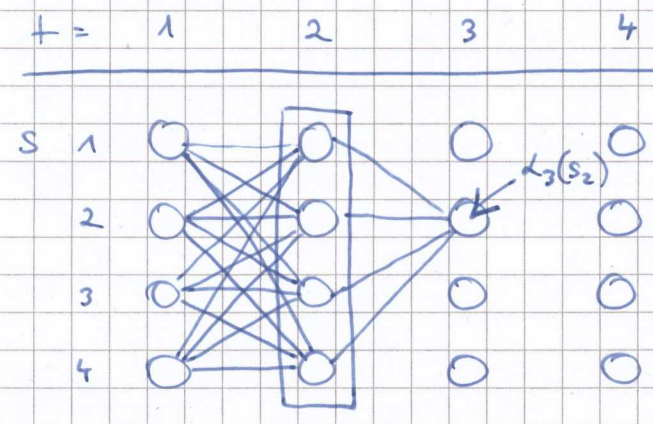
FORWARD ALGORITHM

1.) $t=1$: $\alpha_1(s_1) = \pi_{s_1} \cdot b_{s_1}(o_1)$

2.) $1 < t \leq N$: $\alpha_t(s_t) = \sum_{s_{t-1}=1}^{M} \alpha_{t-1}(s_{t-1}) \cdot a_{s_{t-1} s_t} \cdot b_{s_t}(o_t)$

3.) afterwards $p(\langle o_1, \dots, o_n \rangle) = \sum_{s_n=1}^{M} \alpha_N(s_N)$

$\alpha_t(s_i)$ caches the probability for all paths starting at time 1 up until time t, that meet at time t in state s_i:



BACKWARD ALGORITHM

1.) $\beta_N = 1$

2.) $1 \leq t \leq N$: $\beta_t(s_t) = \sum_{s_{t+1}=1}^{M} a_{s_t s_{t+1}} \cdot b_{s_{t+1}}(o_{t+1}) \cdot \beta_{t+1}(s_{t+1})$

[3) $t=1$: $\sum_{s_1=1}^{M} \pi_{s_1} \cdot b_{s_1}(o_1) \cdot \beta_1(s_1)$ (wikipedia!) Termination?]

it's missing in the paper ...

Pattern Analysis (4) 21.06.19

$$p(\langle o_1, \dots, o_n \rangle) = \sum_{i=1}^{M} \pi_i \beta_1(s_i) \cdot b_1(o_1)$$

(↑ wrong?)

Pattern Analysis (1) 26.06.19

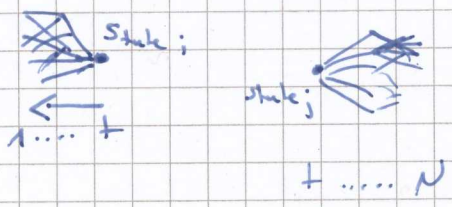
For exam: read through the "must-read" papers
- write down equations and explain

HMM (continued)

Three methods:

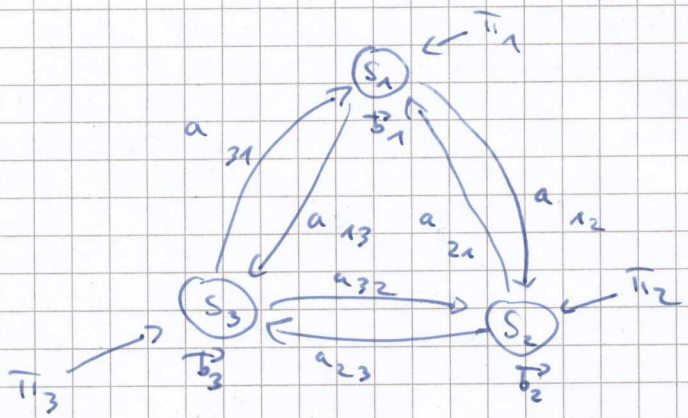
1.) $p(\langle o_1, \dots, o_n \rangle \mid \lambda)$ ↑ or $p(\langle o_1, \dots, o_n \rangle)$: FORWARD - / BACKWARD algorithm

$\alpha_t(s_i)$ $\beta_t(s_j)$



2.) Most likely state sequence for $\langle o_1, \dots, o_n \rangle$: Viterbi

3.) Training of $\lambda = (A, B, \pi)$: Baum-Welch-formulas.



Example path

$\pi_2 \cdot b_2(o_1) \cdot a_{21} \cdot b_1(o_2)$

Pattern Analysis (2) 26.06.19

Missing Termination step in BACKWARD: (Footnote in paper)
omitted bc. we never backtrack to the beginning (since we use FORWARD from there, I guess)

Solution to Task 2: (Viterbi algorithm):
Most likely state sequence:

Method is identical to FORWARD algorithm, with two exceptions:
1.) Replace every sum by max
2.) Technical detail: store the actual path in a second list ψ.

Solution to Task 3 (Training):

Determine $A, B, \vec{\pi}$: no analytical solution ⇒ solve iteratively

Auxiliary quantities

$\xi_t(s_i, s_j) = p(q_t = s_i, q_{t+1} = s_j)$

probability for transition $s_i \to s_j$ at t

$= \frac{\alpha_t(s_i) \cdot a_{s_i s_j} \cdot b_{s_j}(o_{t+1}) \cdot \beta_{t+1}(s_j)}{\sum_{k=1}^{M} \sum_{l=1}^{M} \alpha_t(s_k) \cdot a_{s_k s_l} \cdot b_{s_l}(o_{t+1}) \cdot \beta_{t+1}(s_k)}$

Normalization for all transitions between time t and $t+1$

$A = [a_{ij}]$ matrix of all transition probabilities

$B = [b_i(o_j)]$ matrix of all output probabilities

$\vec{\pi}$: starting probabilities

Given: - example sequences $\langle o_1, \ldots, o_n \rangle$ ("training data")
- number of states
- rough connection structure (e.g. complete graph, left-right HMM)
↑ triangle transition matrix

$$\gamma_t(s_i) = \sum_{s_j=1}^{M} \xi_t(s_i, s_j)$$

↑ probability of being in state s_i at time t

$\sum_{t=1}^{T} \xi_t(s_i, s_j)$: expected # of transitions $s_i \to s_j$

$\sum_{t=1}^{T} \gamma_t(s_i)$: expected # of transitions <u>from</u> s_i

<u>Baum-Welch-Formulas</u> :

EM-style-iteration :
↑
Expectation Maximization

Iterate {

$\overline{\pi}_i$ = expected # of times in s_i at time $t=1$

$= \gamma_1(s_i)$

$\overline{a}_{ij}$ = $\dfrac{\text{expected \# of transitions } s_i \to s_j}{\text{expected \# of transitions from } s_i} = \dfrac{\sum_{t=1}^{T} \xi_t(s_i, s_j)}{\sum_{t=1}^{T} \gamma_{s_i}(s_i)}$

$\overline{b}_{s_j}(k) = \dfrac{\text{expected \# of times in } s_j \text{ and observing symbol } v_k}{\text{expected \# of times in } s_j}$

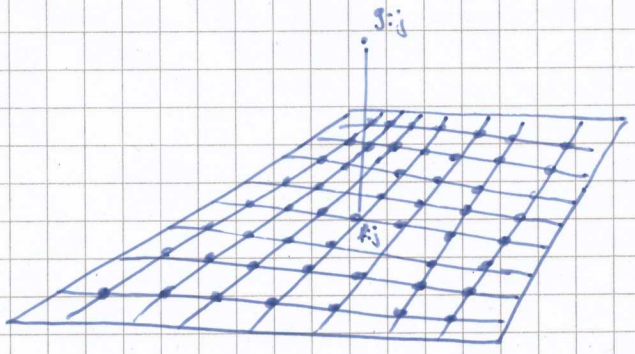
$= \dfrac{\sum_{t=1, \text{ where } o_t = v_k}^{T} \gamma_k(s_j)}{\sum_{t=1}^{T} \gamma_t(s_j)}$

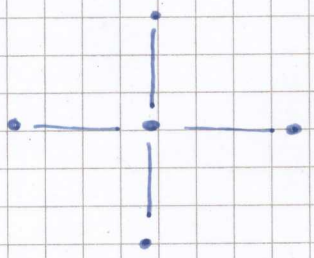
Markov Random Fields

Probabilistic graphical model on undirected graph

The graph structure is not limited to modeling 1-D sequences (like in an HMM), but can cover arbitrary neighborhood relations.

Important special case: consider the pixel grid of an image as a graph:





Observations: grid of noisy pixel $[g_{ij}]$

Hidden Variables: grid of ideal (noiseless) image pixels $[f_{ij}]$

Each hidden variable depends on:

- the associated observation
- its neighborhood of other hidden variables

It is independent of all other quantities ⇒ "the trick" to make inference over $p(f_{ij}, g_{ij})$ tractable

all hidden vars — all observations

More specifically, we model the joint probability $p(f_{11}, f_{12}, \dots, f_{NM}, g_{11}, \dots, g_{NM})$

$$p([f_{ij}], [g_{ij}]) = p([g_{ij}] | [f_{ij}]) \cdot p([f_{ij}])$$

The Markov assumption limits the statistical dependency of the hidden variables: $p([f_{ij}]) =$

$$= p(f_{ij} | f_{11}, f_{12}, \dots, f_{i,j-1}, f_{i,j+1}, \dots, f_{NM}) \cdot \cdot p(f_{11}, f_{12}, \dots, f_{i,j-1}, f_{i,j+1}, \dots, f_{NM})$$

Markov:

$$p(f_{ij} | f_{11}, f_{12}, \dots, f_{i,j-1}, f_{i,j+1}, \dots, f_{NM}) = p(f_{ij} | N(f_{ij}))$$

$N(f_{ij})$: neighborhood of f_{ij}

for example: $N(f_{ij}) = \{ f_{i,j-1}, f_{i-1,j}, f_{i,j+1}, f_{i+1,j} \}$

→ goal is to look for the most probable assignment of values to the hidden variables

$$[f_{ij}] = \underset{f_{ij}}{\operatorname{argmax}} \; p([g_{ij}] | [f_{ij}]) \cdot p([f_{ij}])$$

2 open problems: 1.) How to define $p([g_{ij}] | [f_{ij}])$ and $p([f_{ij}])$?

2.) How can we efficiently find the argmax?

Task 1:

$p([g_{ij}] \mid [f_{ij}])$ defines the relationship of a hidden variable to one or more observations.

For the example of denoising, we could define this as a PDF for, e.g. Gaussian-distributed noise in the observations $[g_{ij}]$

This term is also called "data term"

↑

describes relationship to the data / observations

$p([f_{ij}])$ defines the relationship of hidden variables within a local neighborhood.

For the denoising example, this PDF can capture the fact, that the ideal image $[f_{ij}]$ is smooth, i.e. neighboring pixels / labels are very often identical.

Conceptually, we are done. However for most tasks it is very difficult to find "good" pdfs that translate our concept into the language of probabilistic modeling.

⇒ Solution: The Hammersley-Clifford Theorem states, that a MRF is (essentially) equivalent to a Gibbs Random Field (GRF).

This allows to cast our concept into Gibbs Potentials, which are much more intuitive

A GRF is given by

$$p(\vec{x}) = \frac{1}{Z} \cdot e^{-H(\vec{x})}$$

where $Z = \sum_{\vec{x}'} H(\vec{x}')$ is called "Partition Function" and normalizes the probability. It is hard to compute, but doesn't depend on $\vec{x}$, so for the maximization of $\vec{x}$ it is ignored.

$H(\vec{x})$ is an "energy function", consisting of potentials $V_m(\vec{x})$:

$$H(\vec{x}) = \sum_{m \in S} V_m(\vec{x})$$

If we minimize the energy function, we maximize the GRF

=> Let's define the potentials in such a way, that they are closer to $\emptyset$ in cases where $p([f_{ij}], [g_{ij}])$ is closer to our expectations

Specific example for image smoothing:

$$p([g_{ij}] \mid [f_{ij}]) = \prod_{ij} \frac{1}{\sqrt{2\pi}\,\sigma_{ij}} \cdot e^{-\frac{1}{2\sigma_{ij}^2} \cdot (f_{ij} - g_{ij})^2}$$

$$= \frac{1}{\prod_{i,j} \sqrt{2\pi}\,\sigma_{ij}} \cdot e^{-\sum_{ij} \frac{1}{2\sigma_{ij}^2} \cdot (f_{ij} - g_{ij})^2}$$

Gaussian centered around f_{ij}

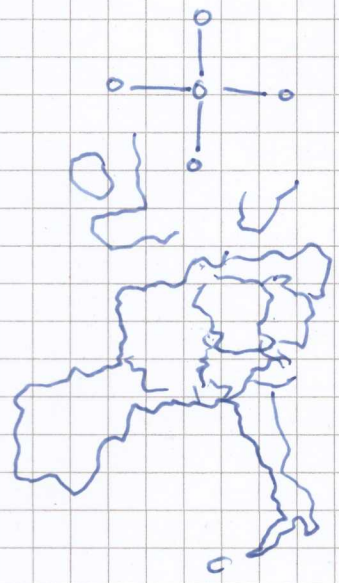
$$p([f_{ij}]) = \frac{1}{Z} \cdot e^{-\sum_{ij} \|\nabla f_{ij}\|^2}$$

↑ potential

MRF: Markov Random Field

Probabilistic graphical model.

Markov property: probability of a node labeling only depends on the neighbors of the node.



Probabilistic Model: MRF

⇕ Hammersley-Clifford-Theorem

Gibbs Random Field (GRF)

GRF: potential functions

Image smoothing example: "data term" / unary potential

$$-\log\left(\mathcal{N}(g_{ij}, f_{ij}, \sigma)\right) = \frac{(f_{ij} - g_{ij})^2}{\sigma}$$

"pairwise potential" / "regularizer"

$$E\left(f_{ij}, f' \in N(f_{ij})\right) = \|f_{ij} - f'\|_2^2$$

These potentials form an energy function. Minimizing these Gibbs potentials leads (under some mild assumptions) to the same result as maximizing an MRF

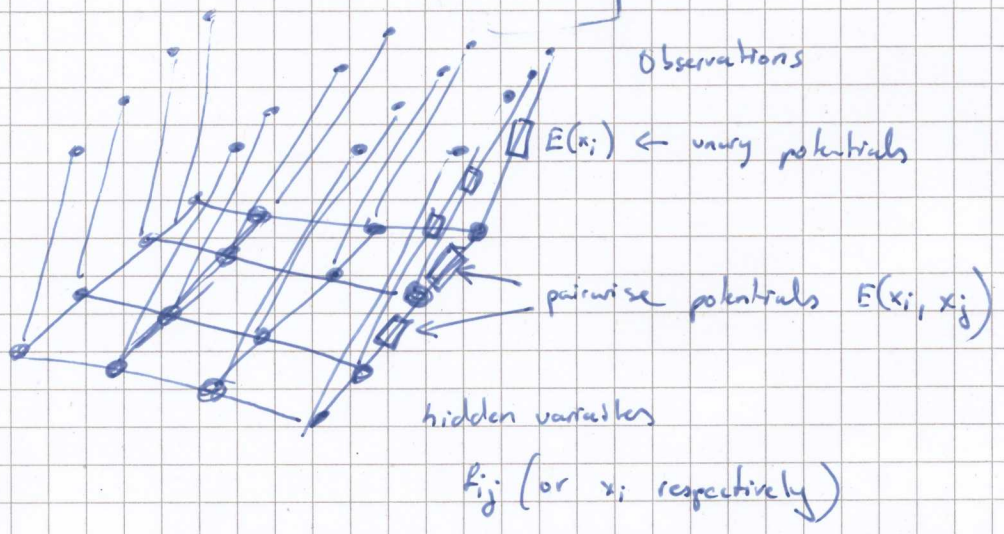
If we only consider the energy terms, the unary potentials are

$$E(x_i),$$

where x_i denotes the i-th hidden variable and the pairwise potentials are $E(x_i, x_j)$.

the minimization problem is just $\sum_{i=1}^{N} E(x_i) + \sum_{i=1}^{N} \sum_{j=1}^{N} E(x_i, x_j)$

$$p(..) = e^{-\sum E(x_i) + \sum\sum E(x_i, x_j)}$$



An efficient solution for a finite set of labels can be computed via Graph Cuts

Kolmogorov, Zabih: "What Energy Functions Can Be Minimized via Graph Cuts?", PAMI 2004

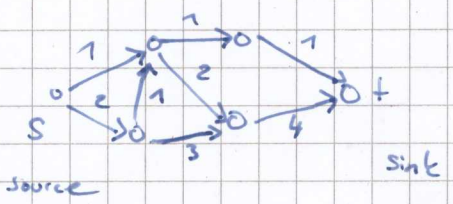
α-Expansion algorithm for L labels:

```
while (changes):
    for each label l:
        binaryGraphCut(l); // 1 vs. all Graph cut:
                           // does the number of assignments
                           // to label l grow?
                           // => keep solution
```

⇒ the core of the label assignment is a binary labeling via graph cuts. This is what we discuss below

Maximum flow

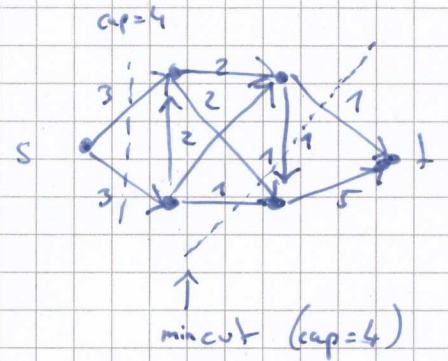
Edge weights = pipe capacities



Question: what is the maximum throughput?

equivalent to

what is the minimum cut?



Separates source s and sink t, such that the sum of edges from source to sink is minimal

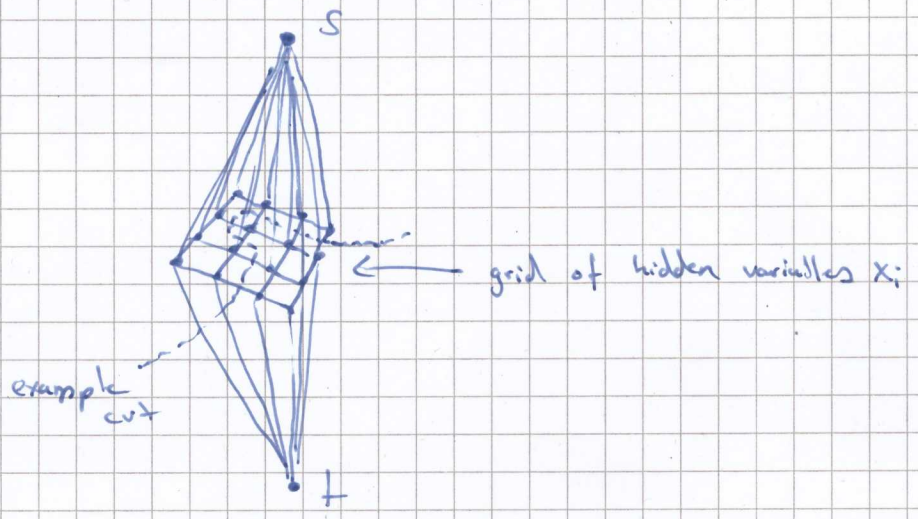
Here: MinCut = 4

Idea: Identify the nodes in a graph with the hidden variables

x: identify source s with label ∅, drain t with label 1

Task: Define the graph topology and weights such that the min cut separates the nodes into ∅'s and 1's with an optimal solution to the energy minimization problem

Graph layout:



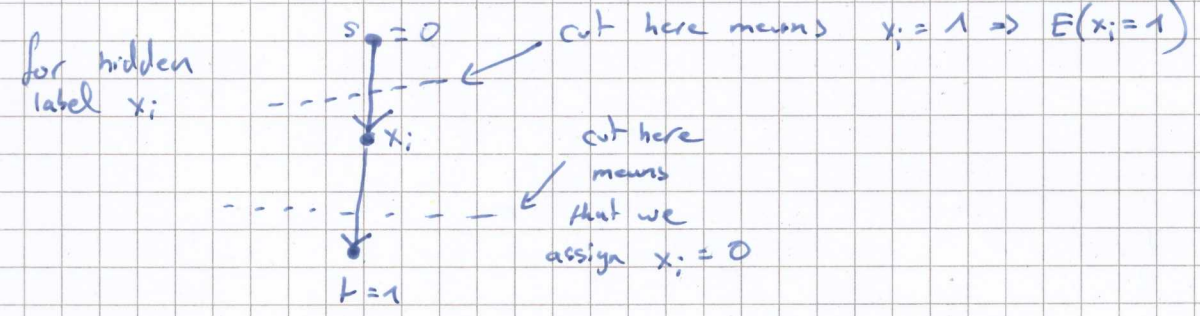
Assignment of edge weights

1.) MinCut works on parts of a graph the same way as the sum of the parts

⇒ decompose $\sum E(x_i) + \sum\sum E(x_i, x_j)$ into individual terms

$E(x_1), E(x_2), \dots, E(x_1, x_4), \dots, E(x_1, x_3)$

2.) Part of the graph for $E(x_i)$ ← unary potential



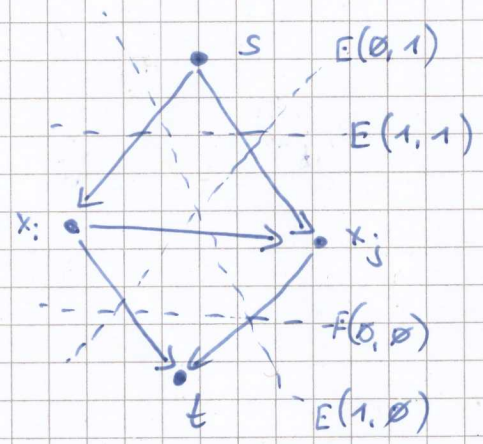
⇒ the cost should be

$E(x_i = \emptyset)$

For example with our image smoothing function $E(x_i) = \|x_i - g_{ij}\|_2^2$

(observation) $\emptyset/1$

3.) Part of the graph for $E(x_i, x_j)$ ← pairwise potential



Depending on the cut, we can have 4 different label assignments

x_i \ x_j	0	1
0	E(0,0)	E(0,1)
1	E(1,0)	E(1,1)

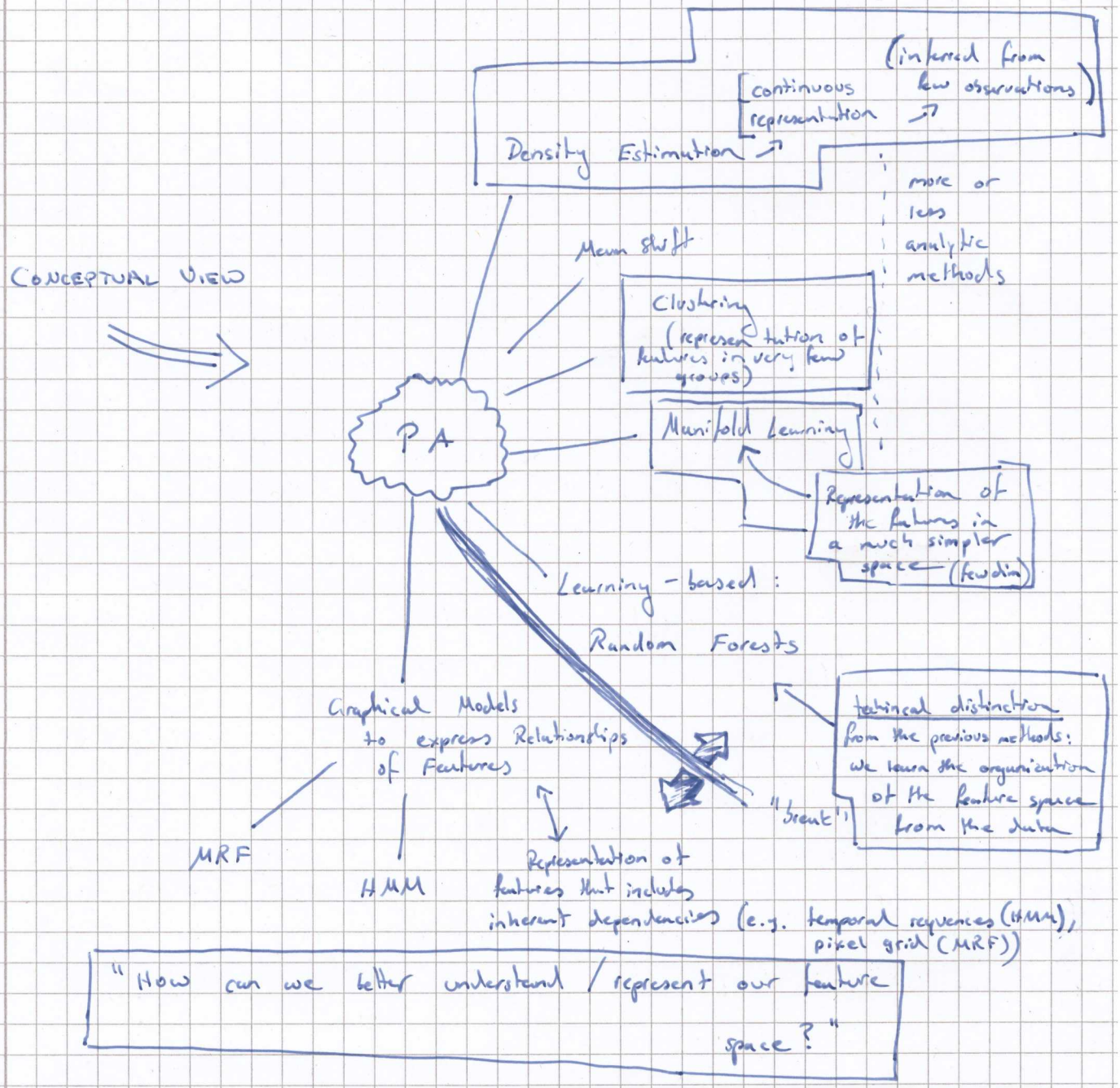
The conversion GRF → MinCut can always be done (and the MinCut solution is always equivalent to the GRF) <u>if</u> the so-called submodularity condition holds:

$$E(0,0) + E(1,1) \leq E(0,1) + E(1,0)$$

(assigning identical labels to neighbors may not be more expensive / "worse" than assigning different labels)

$$\left(\|x_i - x_j\| \Rightarrow \text{(holds for smoothing, for example)} \right)$$

Summary of the class



Pattern Analysis (R) 05.07.14

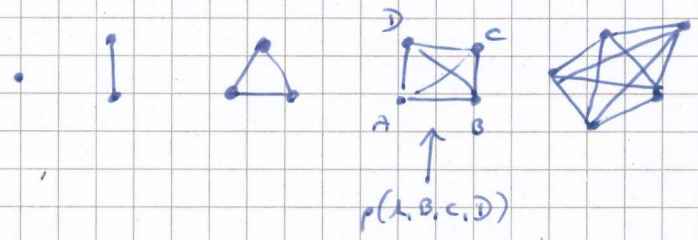
Technical view: Several tools reappear from time to time.

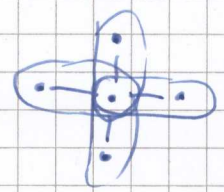
- all methods in the manifold learning part in one way or another enclose an Eigenvalue solution at their core
- also the concept of the graph Laplacian reappears in spectral clustering, Laplacian Eigenmaps, "Manifold Forests"
- in both graphical models an underlying question is how dependent variables are modeled. In both cases we use the notion of Markovianity.

Notes on MRFs (paper on github: "ermongroup") and graph factorization

:

Clique: fully connected subgraph





$\Rightarrow \underline{E}(x_i, x_j)$

$\underline{E}(x_i)$

$\Rightarrow$ MRFs with a maximum clique size of 2 + submodularity condition can be readily (efficiently) solved via graph cuts.

$\Rightarrow$ Cliques larger than 3 can in general not efficiently be solved