# Summary Pattern Analysis
# Sommersemester 2017

Nils Häusler

September 12, 2017

# Density Estimation

The task of density estimation is to obtain a continous representation of the underlying pdf from a set of discrete samples (massumants). Note: that if we have the pdf we can do statistical analysis.

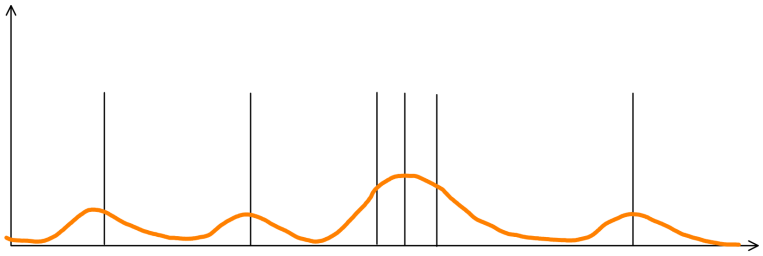


Figure 1: A PDF describing the distribution of measurements

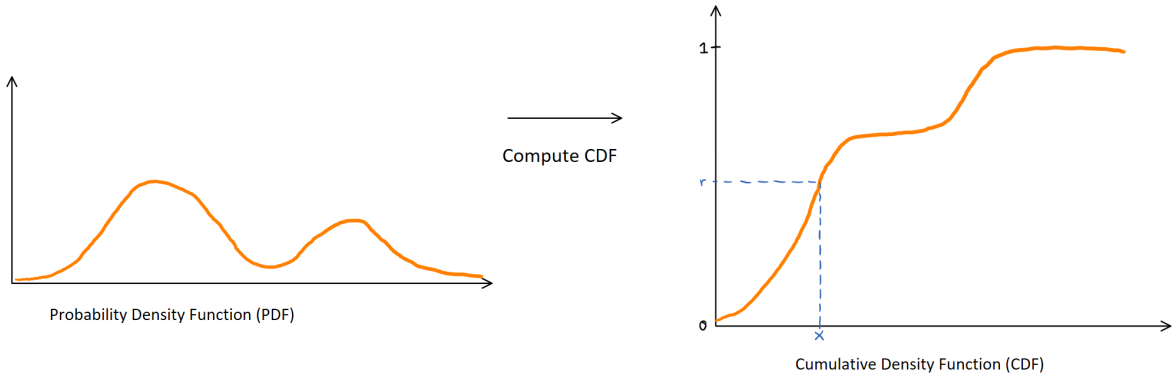
Let $p(\vec{x})$ denote a probability density function pdf then:

1. $p(\vec{x}) \geq 0$
2. $\int_{-\infty}^{\infty} p(\vec{x})\, d\vec{x} = 1$
3. $p(\vec{a} \leq \vec{x} \leq \vec{b}) = \int_{\vec{a}}^{\vec{b}} p(\vec{x})\, d\vec{x}$

**Parametric density estimation** (mostly Pattern Recognition)
Make an assumption about the underlying distribution (e.g. Gausian, GMM) and determine the best fitting distribution parameters from the data. (ML estimation, MAP estimation)

**Non-parametric density estimation** We make no assumption of the underlying Model. (Example Parzen-Rosenblatt estimator)

**Q:** How can we (practically) sample from a pdf?




Compute through discretisation of the pdf $cdf[i] = cdf[i-1] + pdf[i]$. Then draw a uniformly distributed number $(r)$ between 0 and 1. The sampled value is $x$ where $cdf[x] = r$

# Parzen-Rosenblatt estimator

Idea: Quantify the number of samples with a window
The Parzen window estimator interpolates the pdf from the observations in the neighbourhood of a position $\vec{x}$, using an appropriate kernel/window function.

**Short derivtion:** Let $p_R$ denote the probability that $\vec{x}$ lies within region $R$:

$$p_R = \int_R p(\vec{x})\, d\vec{x}$$

Now assume that $p(\vec{x})$ is approximately constant in R.

$$p_R \approx p(\vec{x}) \int_R d\vec{x}$$

For example, let $R$ be a d-dimensional hypercube with side length $h$, then its volume[1][2] is $h^d$

$$p_R \approx p(\vec{x}) V_R$$

Let $p_R = \frac{k_R}{N}$, we determine the probability of making observations in region $R$ by counting the samples in $R$ $(= k_R)$ and dividing by the total number of samples. Note: $p_R$ is also called the "relative frequency"

$$p(\vec{x}) = \frac{p_R}{V_R} = \frac{k_R}{V_R N}$$

Let's write the parzen window estimator as a function of a kernel[3] $k(\vec{x}; \vec{x_i})$, then

$$p(\vec{x}) = \frac{1}{h^d N} \sum_{i=1}^{N} k(\vec{x}; \vec{x_i})$$

where[4]

$$k(\vec{x}; \vec{x_i}) = \begin{cases} 1 & \text{when } \frac{|x_{i,k} - x_k|}{h} \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

equivalently, if we use a (multivariate) gaussian kernel:

$$k(\vec{x}; \vec{x_i}) = \frac{1}{(2\pi)^d |\Sigma|} e^{-(\vec{x}-\vec{x_i})^T \Sigma^{-1} (\vec{x}-\vec{x_i})}$$

**A note on applications**

- General remark: We obtain a continuous pdf, i.e. desity estimation converts a list of measurments to a statistical model
- Specific example: We can sample from a pdf. This means that we have a princeple way of generating new / more / ... data that behaves / looks / ... similary to the observations.

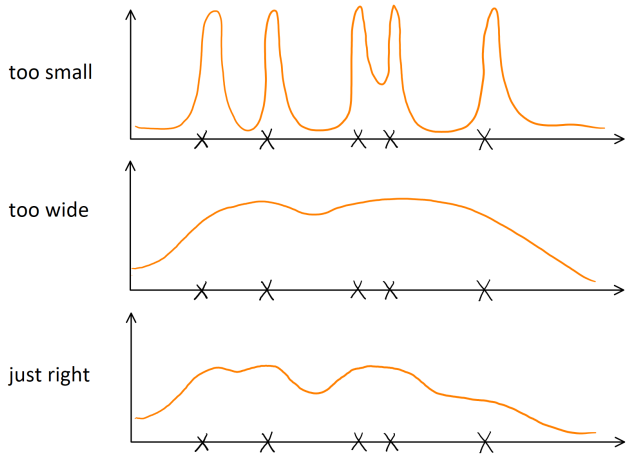
[1] $\int_R d\vec{x}$ is just the volume of $R$
[2] We also write $V_R$ for the volume
[3] Omit $h^d$ if the kernel is gaussian
[4] $\vec{x_i}$ and $\vec{x}$ are not father apart then $0.5h$ in any dimension $k$

**Q:** How can we determine a good window / kernel width h?
Let's do ML est. with a cross-vaslidation (cv) (e.g. leave-one-sample-out cv)

$$p^j_{h,N-1}(\vec{x}) = \frac{1}{h^d N} \sum_{i=1(i\neq j)}^{N} k(\vec{x};\vec{x_i})$$

We estimate the pdf from all samples except $\vec{x_j}$. $\vec{x_i}$ will be used to evaluate the quality of the pdf using window size h.

**Q:** How do the results change with varing window size?




$$\hat{h} = \arg\max_h L(h) = \arg\max_h \prod_{j=1}^{N} p^j_{h,N-1}(\vec{x_j}) = \arg\max_h \sum_{j=1}^{N} \log p^j_{h,N-1}(\vec{x_j})$$

The position of the maximum does not change (when using log-likelihood), because the logarithm is a strictly monotonic function.

# Mean Shift Algorithm

**Purpose:** Find maxima in pdf without actually performing a full density estimation [5].

**Potential applications:** Clustering (maximum is cluster center), segmentation, ...

**Idea:** Maxima can be found, where the gradient of the pdf is zero. (Assume that we have a full density estimator.)

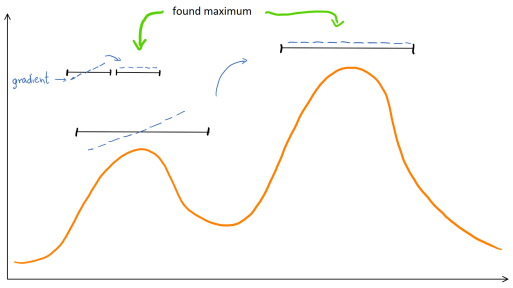



Figure 2: The kernel size indirectly controls the number of indentified maxima

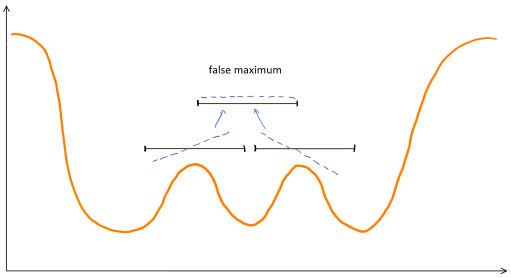



Figure 3: One of the issues is, the case when a zero gradient is just between two finer maxima

Let

$$p(\vec{x}) = \frac{1}{N}\sum_{i=1}^{N} k_h(\vec{x};\vec{x_i})$$

denote the multivariate kernel density estimation. A local maxium of the pdf can be assumed where the gradient vanishes $\nabla p(\vec{x}) = 0$.

$$\nabla p(\vec{x}) = \nabla(\frac{1}{N}\sum_{i=1}^{N} k(\vec{x};\vec{x_i})) = \frac{1}{N}\sum_{i=1}^{N} \nabla k(\vec{x};\vec{x_i})$$

Let's assume that $k_h$ is a radially symmetric kernel, i.e.

$$k(\vec{x};\vec{x_i}) = c_d k_h(||\vec{x_i} - \vec{x}||^2)$$

$$\frac{\partial k_h(S)}{\partial S} = k_h'(S)$$

$$\frac{\partial S}{\partial \vec{x}} = \frac{\partial (\vec{x_i} - \vec{x})^T(\vec{x_i} - \vec{x})}{\partial \vec{x}} = -2(\vec{x_i} - \vec{x})$$

$$\nabla p(\vec{x}) = \frac{1}{N}\sum_{i=1}^{N} c_d k_h(||\vec{x_i} - \vec{x}||^2)(-2(\vec{x_i} - \vec{x})) \doteq \vec{0}$$

$\frac{1}{N}$ and $c_d$ can be dropped then multiply out

$$\sum_{i=1}^{N} k_h'(||\vec{x_i} - \vec{x}||^2)\vec{x_i} - \sum_{i=1}^{N} k_h'(||\vec{x_i} - \vec{x}||^2)\vec{x} = \vec{0}$$

[5] This applies only in some cases, e.g. quickly finding clusters through particle tracing or with a downsampled PDF (see section )

Then we get the mean shift vector

$$\frac{\sum_{i=1}^{N} k_h'(||\vec{x_i} - \vec{x}||^2)\vec{x_i}}{\sum_{i=1}^{N} k_h'(||\vec{x_i} - \vec{x}||^2)} - \vec{x} = \vec{0} \tag{1}$$

To perform a gradient acsend, compute the gradient, walk one step, re-compute the gradient, walk a step, ...

**Mean shift algorithm (formalized)**

1. Compute the mean shift vector $m(\vec{x}^{(t)})$ (see 1)
2. Update $\vec{x} : \vec{x}^{(t+1)} = \vec{x}^{(t)} + m(\vec{x}^{(t)}) = \vec{x}^{(t)} + \frac{\sum_{i=1}^{N} k_h'(||\vec{x_i} - \vec{x}||^2)\vec{x_i}}{\sum_{i=1}^{N} k_h'(||\vec{x_i} - \vec{x}||^2)} - \vec{x}^{(t)} = \frac{\sum_{i=1}^{N} k_h'(||\vec{x_i} - \vec{x}||^2)\vec{x_i}}{\sum_{i=1}^{N} k_h'(||\vec{x_i} - \vec{x}||^2)}$

**Q:** Why is it called "mean shift"?
If we plug in for $k_h$ the Epanechnikov kernel. Then the computation breaks down to the mean of the samples in a circular (hyperspherical) around $\vec{x}^{(t)}$

**Epanechnikov kernel**

$$k_E(\vec{x}) = \begin{cases} c \cdot (1 - \vec{x}^T\vec{x}) & \text{when } \vec{x}^T\vec{x} \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

**Abstract example:** Assume we have a 2-D feature space

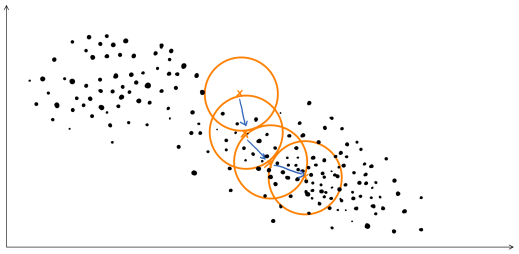


Figure 4: Mean Shift iterations with an Epanechnikov kernel

**Specific example:**

- (Color) quantization
  - Note: the RGB colorspace is not perceptually uniform (Lab or Luv are used in practice)
- (Color) segmentation
  - Similar in result to a super pixel segmentation: operate locally in the image
  - Incorporate the position of each pixel
  - Properly scale each feature dimension, such that distances are comparable (e.g. $\vec{x}$ =(x,y,r,g,b))

**Remarks on the found maxima:**

- Different trajectories typically coverage only to **almost** the same peak, thus, we will have to post process the peaks and somehow reduce them.
- We don't have a guarentee to sit on top of a maximum when reaching a 0-gradient. This is due to the finite window size and the discrete representation of our density (see figure 9).
- If the amount of data is large, then it may become extremly costly to iteratively evaluate the "neighbourhood finder". In that case we have to help ourself, either with a smart data structure (oct-tree or a generalisation for many dimensions) or locality sensitve hashing (LSH).

# Clustering [6]

Grouping or segmenting a collection of objects into subsets or "clusters", such that those within each cluster are more closely related to one another than objects assigned to different clusters. Sometimes the goal also is to arrange the clusters into a natural hirarchy.
All clustering methods attempt to group the objects based on the definition of similarity supplied to it → choice of distance or dissimilarity measure between two objects.

## Proximity Matrices

Sometimes the data is represented directly in terms of the proximity (alikeness or affinity) between pairs of objects. These can either be *similarities* or *dissimilarities*. This type of data can be represented by an $N \times N$ matrix $D$, where $N$ is the number of objects, and each element $d_{ii'}$ records the proximity between the $i$th and $i'$th objects.
Most algorithms presume a symmetric matrix of dissimilarities with nonnegative entries and zero diagonal elements. If the original data were collected as similiarities, a suitable monotone-decreasing function can be used to convert them to dissimilarities.

## Dissimilarities Based on Attributes

Most often we have measurements $x_{ij}$ for $i = 1, \ldots, N$, on variables $j = 1, \ldots, p$ (also called *attributes*. We first have to construct pairwise dissimilarities between the observations. In the common case, we define a dissimilarity $d_j(x_{ij}, x_{i'j})$ (e.g. squared distance $(x_{ij} - x_{i'j})^2$, even though not appropriate for nonquantitive/categorial attributes) between values of the $j$th attribute, and then define

$$D(x_i, x_i') = \sum_{j=1}^{p} d_j(x_{ij}, x_{i'j})$$

- Ordinal variables: Convert to numbers on a scale
- Categorial(nominal) variables: Dissimialrity has to be described explicitly

## Object Dissimilarity

For comparing objects, often a weighted sum of distances is computed:

$$D(x_i, x_i') = \sum_{j=1}^{p} w_j d_j(x_{ij}, x_{i'j}); \sum_{j=1}^{p} w_j = 1$$

Note that giving every variable the same weight $w_j$ doesn't necessarily give all attributes equal influence. The relative influence of the $j$th variable is given by $w_j \cdot \bar{d}_j$, where $\bar{d}_j = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} d_j(x_{ij}, x_{i'j})$ is the average dissimilarity on the $j$th attribute. To achieve this the weights have to be set to $w_j = \frac{1}{\bar{d}_j}$, which doesn't always make sense.
Specifying an appropriate dissimilarity measure is far more important in obtaining success with clustering than choice of clustering algorithm.

## Clustering Algorithm

Three types of clustering algorithms:

- **Combinatorial:** work directly on the observed data with no direct reference to an unterlying probability model.

[6] For clustering see section 14 of "The Elements of Statistical Learning" (Hastie, Tibshirani, Friedman - 2009). Section 14.3 "Cluster Analysis" introduces different flavours of k-means.

- **Mixture modeling:** suposes that the data is a sample from some population describe by an probability density function. This density function is characterized by a parameterized model taken to be a mixture of component density functions; each component density describes one of the clusters.
- **Mode seeking("bump hunting"):** take a nonparamteric perspective, attempting to directly estimate distinct modes of the pdf. Observations "closest"to each respective mode then define the individual clusters.

## Combinatorial Algorithms

Each observation is uniquely labeled by an integer $i \in \{1, \dots N\}$. A prespecified number of clusters $K < N$ is postulated, and each one is labeled by an integer $k \in \{1, \dots, K\}$. Each observation is assigned to (only) one cluster by an "encoder" $C(i) = k$, that assigned the $i$th observation to the $k$th cluster. One seeks the particular encoder $C^*(i)$ that achieves the required goal, based on the dissimilarities between every pair of observations.
One approach is to directly specify a mathematical loss function and attempt to minimize it through some combinatorial optimization algorithm. Since the goal is to assign close points to the same cluster, a natural loss function would be $W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d(x_i, x_{i'})$ the *within-cluster scatter*.
One can equivalently maximize the *between-cluster scatter* $B(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')\neq k} d(x_i, x_{i'})$
One simple way to solve this is to minimizes $W$ or maximized $B$ over all possible assignments of the $N$ data points to $K$ clusters. Unfortunally, such optimization by complete enumeration is feasible only for very small data sets.
For this reason, practical clustering algorithms are able to maximize only a very small fraction of all possible encoders $C$. The goal is to identify a small subset that is likely to contain the optimal one, or at least a good suboptimal partition. Such feasible stratgies are based on iterative greedy descent (initial partition is specified, at each step cluster assignments are changed such that value of criteroen is improved, stop when no more improvement possible).

## K-means

Squared Euclidean distance:

$$d(x_i, x_{i'}) = \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2 = ||x_i - x_{i'}||^2$$

The within-point scatter can be written as

$$W(C) = \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - \bar{x_k}||^2$$

where $\hat{x_k}$ is the mean vector associated with the $k$th cluster, and $N_k$ is the number of observations assigned to a cluster. Thus, the critereon is minimized by assigning the $N$ observations to the $K$ clusters in such a way that within each cluster the average dissimilarity of the observations from the cluster mean, as defined by the points in that cluster, is minimized.
**Algorithm:**

1. For a given cluster assignment $C$, compute each clusters mean
2. Assign each observation to the closest cluster mean
3. Iterate 1 and 2 until the assignments do not change

Each iteration reduces the value of the critereon, so that convergence is assured, but the solution may be suboptimal. In addition, on should start the algorithm with many different random choices for the starting means, and choose the solution having smallest value of the objective function.

## Gaussian Mixtures as Soft K-means Clustering

The $K$-means clustering procedure is closely related to the EM algorithm for estimating a certain Gaussian mixture model. Suppose we specify $K$ mixture components, each with a Gaussian density having a scalar covariance matrix $\sigma^2 I$. Then the relative density under each mixture component is a monotone function of the Euclidean distance between the data point and the mixture center. Hence in this setup EM is a "soft" version of $K$-means clustering, making probabilistic assignments of points to cluster centers. As the variance $\sigma^2 \to 0$, these probabilities become 0 and 1, and the two methods coincide (responsibility of mixture component $i$ : $\frac{g_i(x)}{\sum_m g_m(x)}$).

## Vector Quantization

The $K$-means clustering algorithm represents a key tool in the apparently unrelated area of image and signal compression, particular in *vector quantization* or $VQ$(Gersho and Gray, 1992).
**Algorithm**:

1. Convert to grayscale
2. Break image into small blocks, e.g. $2 \times 2$ blocks of pixels
3. Each block is regarded as a vector in $\mathcal{R}^4$
4. Apply $K$-means
5. Each of the blocks is approximated by its closest cluster centroid, known as codeword. The clustering process is called the *encoding* step, and the collection of centroids is called the *codebook*.

## $K$-medioids

$K-$means stuggels with outliers, because using *squared* Euclidean distance places the highest influence on the larger distances.
Instead of taking means as cluster center, one can identify the sample inside each cluster, which is nearest to all other samples inside it. One then assigns this sample to be the new cluster center.
The downside is that $K$-medioids is far more computationally intensive.

## Practical Issues

In order to use $K$-means or -medioids one must select the number of clusters $K^*$ as initialization. A choice for the number of clusters $K$ depends on the goal. For data segmentation $K$ is usually defined a part of the problem.
Data-based methods for estimating $K^*$ typically examine the within cluster dissimilarity $W_k$ as a function of the number of clusters $K$. Seperate solutions are obtained for $K \in \{1, 2, \ldots, K_{max}\}$. The corresponding values generally decrease with increasing $K$.

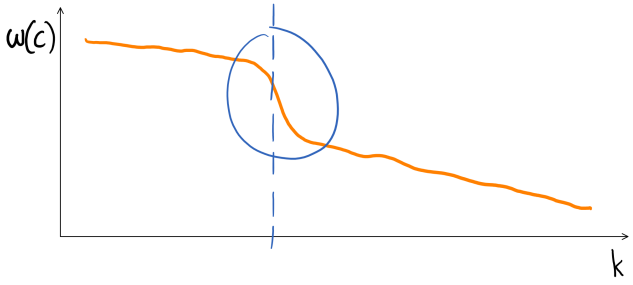



Figure 5: Approach 1: Track the rate of change of a quality metric (like $w(c)$). Proposed in "Pattern Classification" (Duda, Hart, Stork).

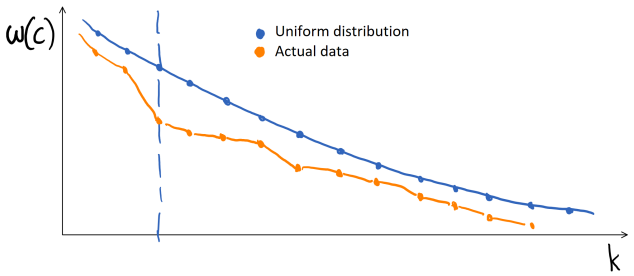



Figure 6: Approach 2: Let $w'(c)$ be a metric on a uniform distribution of samples. Relate change of $w(c)$ to change of $w'(c)$. Proposed in TEoSL.

**Intuition:**

- $K < K^*$ will partition actual groups into subsets. The solution criterion value will tend to decrease substantially with each successive increase in the number of specified clusters, $W_{K+1} \ll W_K$, as the natural groups are successively assigned to seperate clusters.
- $K > K^*$ will fuse natural groups into one. This will tend to provide a smaller decrease in the criterion as $K$ is further increased.

$\Rightarrow$ Splitting a natural group, within which the observations are quite close to each other, reduces the criterion less than partitioning the union of two well-separated groups into their proper constituents.
To the extent this scenario is realized, there will be a sharp decrease in successive differences in critereon value, $W_K - W_{K+1}$, at $K = K^*$. An estimate for $K^*$ is then obtained by identifying a "kink" in the plot of $W_K$ as a function of $K$. As with outher aspects of clustering procedures, this approach is somewhat heuristic.
Recently proposed *Gap statistic* (Tibshirani et al., 2001b):

- Compare the curve $logW_K$ to the curve obtained from data uniformily distributed over a rectangle containing the data.
- Optimal $K$ is where the gap between the two curves is largest

## Hirarchical Clustering

Hirarchical clustering methods do not require specifications like for $K$-means. Instead, the require the user to specify a measure of dissimilarity between (disjoint) *groups* of observations, based on the pairwise dissimilarities among the observartions in the two groups.
At the lowest level, each cluster contains a single observation. At the highest level there is only one cluster containing all of the data.
Two basic paradigms: *aglomerative* (bottom-up, every step up there is one less cluster) and *devisive* (top-down, every step down there is one new cluster). There are $N - 1$ levels in the hirarchy.

**Agglomerative Clustering** Let $G, H$ represent two groups. The dissimilarity $d(G, H)$ is computed from the set of pairwise observation dissimilarities $d_{ii'}$.*Single linkage* (SL) agglomerative clustering takes the intergroup dissimilarity to be that of the closest (least dissimilar) pair. This is also often called the *nearest-neighbor* technique. *Complete linkage* (CL) agglomerative clustering (*furthest-neighbor* technique) takes the intergroup dissimilarity to be that of the furthest pair. *Group average* (GA) clutering uses the average dissimilarity between the groups (trade-off between compactness/diameter of clusters).

**Note:** We have several options for clustering data. K-means and its variants are straight forward and easy to understand, but if the proper number of clusters is part of the unknowns, it may be a suboptimal choice. One alternative is mean shift clustering. Here is the number of clusters determined implicitly by the kernel type and size plus the type of bump post process.

# Dirichlet Process

The Dirichlet Process to model infinite gaussian mixtures.

**Short reminder: Gaussian mixture models**

1. What is a GMM? $\Rightarrow \sum_{i=1}^{k} \beta_i \mathcal{N}(\mu_i, \Sigma_i)$
2. What parameters? $\Rightarrow \beta_i, \mu_i, \Sigma_i$
3. How do I determine these parameters? $\Rightarrow EM - Algorithm$

   Start with random initialization, expectation (how much does each component contribute to each point), maximization (update component parameters), repeat until convergence

   From a more abstract perspective, EM performs two tasks simultaneously: segmentation/clustering assignment, and model parameters estimation/fitting

**The idea for infinite mixture models:** Instead of fitting a specific distribution to the data (the GMM), we define a meta-distribution from which the actual distribution is drawn. specifically we can draw a GMM from a dirichlet process[7]. However this is a top-down perspective, i.e. if we randomly draw a GMM, we will certainly not draw one that fits nicely to our data. From a bottom up perspective, we need a fitting algorithm that works with the available data points and finds a GMM in such a way, that it could also have been drawn from a distribution of distributions (the dirichlet process). To illustrate the model behind the fitting algorithm, we usually talk about the "**Chinese restaurant process**" (CRP).

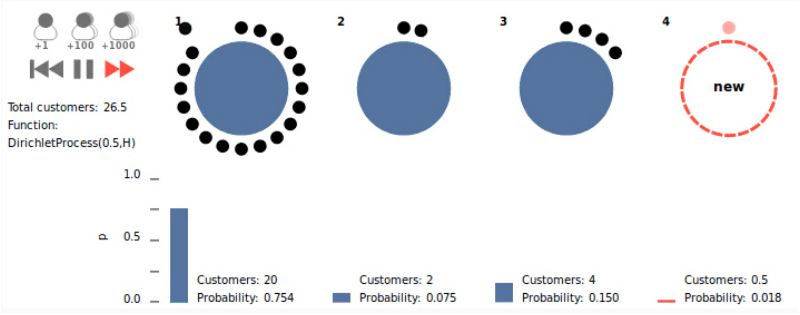



Figure 7: There are tables in a restaurant (our mixture components). Whenever a new customer comes in (a sample), it chooses the table it most relates to. When the customer is unsatisfied with current offering, he can open up a new table (Check the explaination form wikipedia).

**Extensions to a "normal" GMM:**

1. If a customer prefers to sit at a new table, this is possible we can add arbitrary many tables
2. The more people sit on a table the more attractive it is to the new customers (rich get richer)

The chinese restaurant process (CRP) provides a constructive way of sampling from a dirichlet process.

[7]Because uniform ??? parameters for drawing the GMM

**Gibbs sampling:** A straight forward (probailistic[8]) clustering algorithm based on the CRP

1. Init: assign each sample to a cluster (e.g. randomly, or do $k-$means)
2. Randomly select a sample $x_i$ from the data
3. Compute a affinity of $x_i$ to each table: $t_i = \frac{N_i}{N+\alpha}\mathcal{N}(x_i, \mu_i, \Sigma_i)$ (Gaussian distance of $x_i$ to $(\mu_i, \Sigma_i)$), where $N_i$ is the number of customers on the table (number of samples assigned to $\mathcal{N}(\mu_i, \Sigma_i)$ and $N$ is the total number of samples, $\alpha$ is an "expansion parameter"
4. Compute affinity to a new table $t_0 = \frac{\alpha}{N+\alpha} \cdot \mathcal{N}(x_i, \mu_0, \Sigma_0)$
5. Collect all affinities $t_0, \ldots, t_T$ in a list, normalize sum to one
6. Sample from that list a table assignment (all affinities are interpreted as a PDF from which we sample)
7. Recompute $\mu_i, \Sigma_i$ for the assigned table
8. Goto 2, stop after certain number of iterations

**Back to the top-down view:** Using the CRP, we are effectivly drawing a GMM using a dirichlet process. Implicitly, the mixture weihts $\beta_i$ match a $Beta(\alpha)$ distribution and the normal distributions are drawn from a hyper-distribution, i.e. $\vartheta_i \sim H(\lambda)$

**Stick breaking process:** (check the wiki-page)

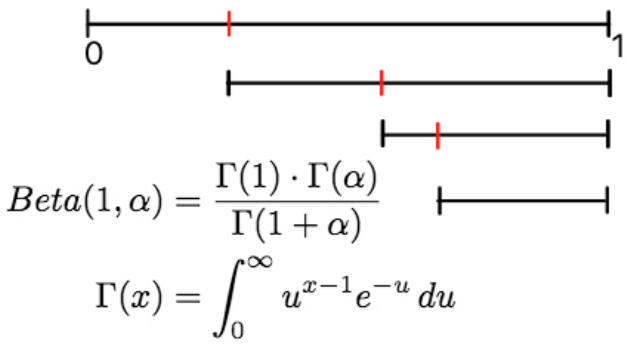



Figure 8: Illustration of a $Beta$ distribution

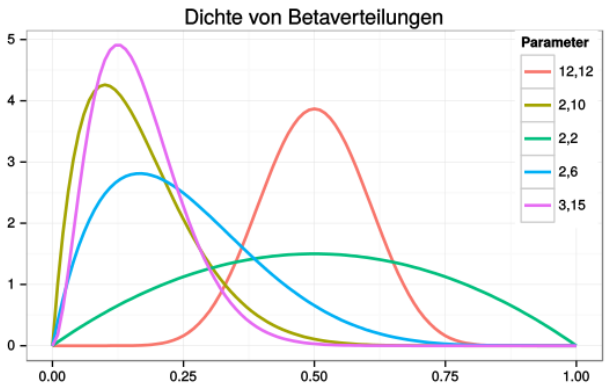



Figure 9: Qualitatively, this is what the $Beta$ distribution looks like

The expansion parameter $\alpha$ is a prior that influences the number of clusters that will be created.

**Observation 1:** The larger the expansion parameter $\alpha$ is, the more likely it is to draw a number that is close to 0 form $Beta(1, \alpha)$
The number that has been drawn form $Beta(1, \alpha)$ is used in the stick breaking process to determine the weight of the i-th mixture component $\beta_i$

**Observation 2:** The number that has been drawn from $Beta(1, \alpha)$ is used in the stick breaking process to determine the weight of the $i$-th mixture component $\beta_i$.

[8] because we sample from the list ..???

**Stick breaking process (formalised):**

$$b_i \sim Beta(1, \alpha)$$

$$\beta_i = b_i \prod_{i=1}^{i-1} (1 - b_i) = b_i (1 - \sum_{i=1}^{i-1} \beta_i)$$

# Manifold Learning

**Principal idea:** Reduce the dimensions of the Data, but preserve the structure of the data / underling manifold.

## Curse of Dimensionality

"Human intuition breaks in high dimensional spaces" - (Bellman, 1961)

**Let's illustrate this:** Consider a d-dimensional feature vector $\vec{x_1}, \vec{x_2}, \ldots, \vec{x_N} \in \mathbb{R}^d$ where $0 \leq x_{i,k} \leq 1$ (??? wo kommt das k her) uniformly distributed in a d-dimensional hyperspace of volume 1. Let's say we would like to cover enough volume of the cube to collect 1% of the data. Let's say we also use a cube for this task. What is the required edge length $s$ of the cube to obtain that 1% of space?

**Example:** A 10 dimensional hypercube

$$V = s^d \Rightarrow s = V^{\frac{1}{d}} = 0,01^{\frac{1}{10}} = 0.63$$

Another way of thinking about this is, that in a very high dimensional space, virtually every feature point is located at the boundary of the feature space[9]. This leads to the effect, that common distance measures loose their effectivity. E.g. the *median* distance for the nearest neighbour to the origin.

**Example applications:** ("Notorious" examples for high dimensional data)

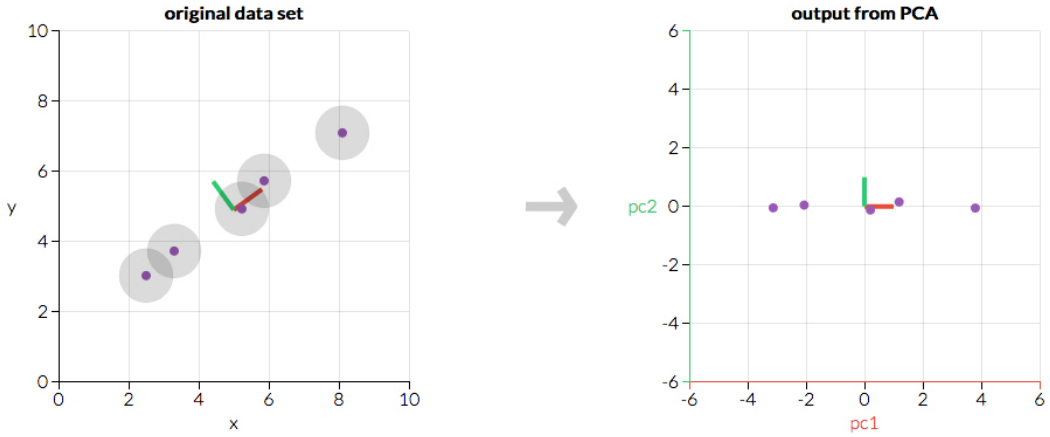
- hyper-spectral remote sensing image classification
- satellite image: perform e.g. agricultural monitoring classify type of vegetation from hyper-spectral (many color channels) image.

In such classification pipelines, dimensionality reduction is often one integral component.

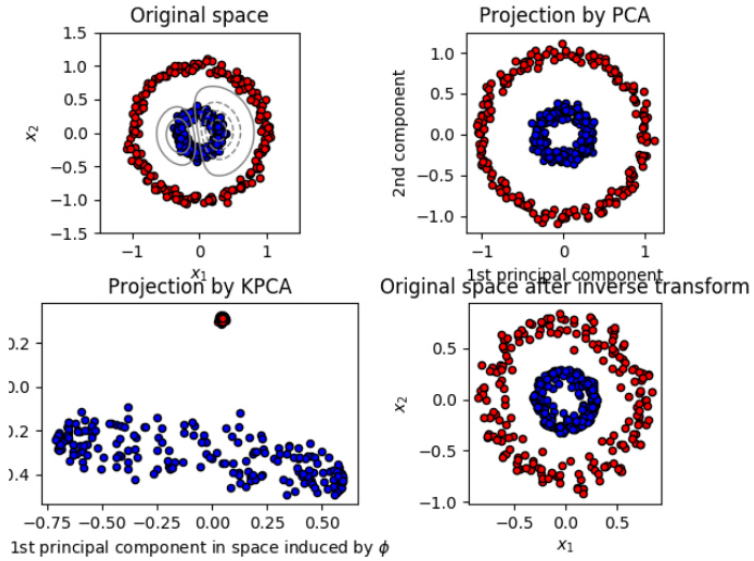
[9] Because in at least on dimension, we draw a very low of very high value

## (k)PCA

Our known approch is the principle component analysis (PCA).




Orthogonal basis that is aligned with the "maximum spread" (w.r.t. the covariance) of the data. It is a global unsupervised method[10]. Consider that PCA os a linear method.




The Kernel PCA performs a non-linear mapping of the data, then perform a standard (linear) PCA on the result. With the kernel-trick we can do that in one step. The Objective function: (the non-linear mapping is part of $\phi$)

$$\delta = \sum_{i,j=1}^{N} (\phi(\vec{x_i}) - \phi(\vec{x_j}))^T (\phi(\vec{x_i}) - \phi(\vec{x_j})) + \lambda(\phi^T \phi - 1)$$

Some offsprings of the Kernel PCA idea are:

1. Perform some preprocessing / mapping that operates non-linearly
2. The dimensionality reduction itself is an operation that operates linearly.

[10]Operating on all data points, no labels, find one global optimal solution

## Multi Dimensional Scaling (MDS)

*Task:* Reconstruct a set of points (potentially in a lower dimension) from their differences. MDS computes the "best" (in a least square sense) coordinates up to rotation, translation and mirroring (axis reversal).
*Mathmatical problem formulation:*

- Let $S = \{x_1, \ldots, x_N\}, x_i \in \mathbb{R}^d$.
- Let $X$ denote a matrix consisting of all samples, $X = [x_1, \ldots, x_N] \in \mathbb{R}^{d \times N}$.
- Let $D^2 = [d_{ij}^2]_{i,j \in \{1,\ldots,N\}}$, where $d_{ij}^2 = (x_i - x_j)^T (x_i - x_j)$.
- Goal: Given $D^2$, compute $X$.
- We assume that $x_1, \ldots, x_N$ have zero mean i.e. $\sum_{i=1}^N x_i = 0$

Let us consider the distance matrix in terms of $x$:

$$d_{ij}^2 = (x_i - x_j)^T (x_i - x_j) = x_i^T x_i + x_j^T x_j - 2 x_i x_j$$

In matrix notation,

$$D^2 = diag(X^T X) \cdot \vec{1}^T + \vec{1} \cdot diag(X^T X)^T - 2 X^T X$$

where $diag(A)$ denotes a vector diagonal entries of $A$, i.e. $diag(A) = (a_{11}, a_{22}, \ldots, a_{NN})^2$, and $\vec{1}$ denotes a vector of ones, i.e. $\vec{1} = (1, 1, \ldots, 1)^T \in \mathbb{R}^d$.
Multiply $D^2$ from left and right with a centering matrix $C = (I - \frac{1}{N} \vec{1}\, \vec{1}^T)$, and weight the result by $-\frac{1}{2}$:

$$\begin{aligned} -\frac{1}{2} C D^2 C &= -\frac{1}{2} (I - \frac{1}{N} \vec{1}\, \vec{1}^T)(diag(X^T X) \cdot \vec{1}^T + \vec{1} \cdot diag(X^T X)^T - 2 X^T X)(I - \frac{1}{N} \vec{1}\, \vec{1}^T) \\ &= \ldots \text{(see lecture, too lazy)} \\ &= X^T X \end{aligned}$$

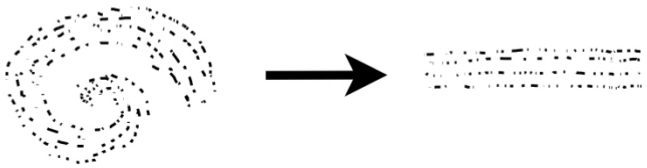
This is a matrix factorization problem. If we compute the eigenvector-eigenvalue decomposition (we can do this because its an square matrix) of $X^T X$ we get $U \Sigma U^T \Rightarrow X = \Sigma^{\frac{1}{2}} U^T$.
MDS is essentially the same thing as PCA, but it operates on distances. We can reduce the dimensionality of our data by:

- Determining the $m$ largest eigennvalues and their corresponding eigenvectors
- Dropping the remaining eigenvalue and eigennvectors inside the multiplication

We can calculate $\frac{\sum_{i=1}^p \lambda_i}{\sum_{i=1}^{n-1} \lambda_i}$, which expresses the proportioin of variance explained by $p$ dimensions.

## ISOMAP Algorithm

A non-linearity "patch" to MDS



**Idea:** Nearby points have their "usual" euclidian distance. If a pair of points are not within a local neigbourhood, then the distance between these points is a **graph distance**[11]. Then run MDS on the resulting distance Matrix. ISOMAP operates on geodesic distances (like a distance on earths surface/distances on the manifold).

### Additional notes

- Manifold learning algorithms are based on the idea that the dimensionality of many data sets is only artificially high. Although the data points may consist of thousands of features, they may be described as a function of only a few underlying parameters. That is, the data points are actually sampled from a low-dimensional manifold that is embedded in a high dimensional space. Manifold learning algorithms attempt to uncover these parameters in order to find a low-dimensional representation of the data.
- Error function:

$$E = ||\text{inner product distances in graph} - \text{inner product distances in new coordinate system}||_{L_2}$$

- Finding optimal dimensionality: Look at residual variance (or error) (depending on $d$) and search for "elbow" at which the curve ceases to decrease significanlty with added dimensions. (PCA/MDS tend to overestimate dimensionality)
- PCA and MDS are guaranteed, given sufficient data, to recover the true structure of linear manifolds
- ISOMAP is guaranteed asymptotically to recover the true dimensionality and geometric structure of a strictly larger class of nonlinear manifolds, which intrinsic geometry is that of a convex region of Euclidean space
- ISOMAP is a polynomial time, noniterative procedure which guarantees global optimality
- Possible problems:
  - If $k$ for $k$-NN is to large, or noise in the data moves the points slightly off the manifold, ISOMAP is vulnerable to "short-circuiut error", where even single errors can alter the low-dim embedding drastically
  - If $k$ is to small, the neighborhood graph may become too sparse to approximate geodesic paths accuratly

[11] Compute the all pairs shortest path (Dejkstra, A*, DFS, Floyd–Warshall, ...)

## Locally Linear Embedding (LLE)

**Idea:** Treat the local neighborhood of a point $x$ as linear. This makes the mapping an overall non-linear mapping consisting of small linear patches and we can linearily interpolate a point $x_i$ from its neighbors,

$$x_i = \sum_{x_j \in N(x_i), j \neq i} w_{ij} x_j, \text{ subject to } \sum_j w_{ij} = 1$$

and search points $x'$ in a lower dimensional space, such that $x'_i = \sum_{j \in N(x'_i)} w_{ij} x'_j$

**Algorithm:**

1. Define the neighborhood ($k$-nearest neighbors, distance thresholding)
2. Solve for $w_{ij}$ in the high dimensional space

$$min \sum_i ||x_i - \sum_{j \in N(x_i)} w_{ij} x_j||_2^2 \text{ subject to } \sum_{j \in N(x_i)} w_{ij} = 1$$

3. Solve for $x'_i \in \mathbb{R}^{d'} (d' \ll d)$:

$$\sum_i ||x'_i - \sum_{j \in N(x_i)} w_{ij} x'_j||_2^2, \text{ subject to } \frac{1}{N} \cdot \sum_i x'_i \cdot x_i'^T = I, \sum_i x'_i = \vec{0}$$

   The first constraint says that the covariance is the identity. Which basically fixes the volume.

Nearer points in the original space will result in higher weights.

**Caveats:**

1. The idea of LLE is really nice, but the original formulation is sometimes a bit unstable if applied on real data. Therefore there exists a number of variants to LLE in the literature that aims to make it more robust.
2. We will now deviate from the original formulation, and modify the constraint in step 2 to $\sum_{jinN(x_i)} w_{ij}^2 = 1$, to obtain a closed solution.

*Modified step 2* (weight computation): Note tha the objective function is <u>invariant</u> to translation:

$$\sum_{i=1}^N ||x_i - \sum_j w_{ij} x_j||_2^2 = \sum_{i=1}^N ||(x_i - \vec{t}) - \sum_j w_{ij}(x_j - \vec{t})||_2^2$$

$$\Rightarrow \text{set } x_i = \vec{t} \Rightarrow \sum_{i=1}^N || \sum_j w_{ij}(x_j - x_i)||_2^2$$

$$= ||M_i \vec{w_i}||_2^2, \text{ where } \vec{w_i} = \begin{pmatrix} w_{i1} \\ w_{i2} \\ \dots \\ w_{iN} \end{pmatrix}, M_i = \begin{pmatrix} x_1 - x_i & x_2 - x_i & \dots & x_N - x_i \end{pmatrix}$$

Note: differences $x_j - x_i = 0$ for $x_j$ outside the neighborhood of $x_i$

$$\Rightarrow minimize (M_i w_i)^T (M_i w_i) + \lambda (1 - w_i^T w_i)$$

where the latter is the constraint that the squared weights should sum up to one. We see why we have to take the squaered weights critereon: if we would compute the derivative of $\lambda(1 - w_i)$, the constraint would vanish to 0. We would have to use linear programming or something.
Compute

$$\frac{\partial}{\partial w_i}(w_i^T(M_i^T M_i)w_i + \lambda(1 - w_i^T w_i) = 2 \cdot M_i^T M_i w_i - 2\lambda \cdot w_i = 0$$

$$\Rightarrow M_i^T M_i w_i = \lambda w_i$$

This is an eigenvector eigenvalue problem and can solve this easily.

## Laplacian Eigenmaps

**Idea:**

1. Build an adjacency graph on the set of feature points $S = \{x_1, \ldots, x_n\}$
2. Choose weights $w_{ij}$ for the edges of the graph
3. Perform eigendecomposition of the graph Laplacian
4. Obtain low-dimensional embedding

Choose the weights as affinities, meaning that closer point pairs have higher weights → points that are closely together in high-dimensional space shall remain close in the lower-dimensional embedding.
One common choice for the weights is the so-called heat kernel $w_{ij} = e^{-||x_i - x_j||_2^2}$, or just binary affinity $w_{ij} = 1$ if $||x - x_j||_2^2 \leq t$ and 0 otherwise.

**Ojective function:**

$$minimize \sum_{i=1}^{N} \sum_{j=1}^{N} ||x'_i - x'_j||_2^2 w_{ij} \qquad \text{where} \quad x' \in \mathbb{R}^{d'}, d' \ll d$$

Contraint for optimization: $x'^T D x' = 1$ (squared distances should add up to 1).
Relationship of the objective function to the graph Laplacian:

$$\begin{aligned}
\sum_{i,j} ||x'_i - x'_j||_2^2 w_{ij} &= \sum_{i,j} (x_i'^T x'_i + x_j'^T x'_j - 2 x_i'^T x'_j) w_{ij} = (2 \cdot \sum_{i,j=1}^{N} (x_i'^T x'_i) \cdot w_{ij}) - (2 \cdot \sum_{i,j} x_i'^T x'_j w_{ij}) \\
&= (2 \cdot \sum_{i=1}^{N} x_i'^T x'_i \cdot (\sum_{j=1}^{N} w_{ij})) - (2 \cdot \sum_{i,j} x_i'^T x'_j w_{ij}) \\
&= 2 x'^T D x' - x'^T W x' = 2 \cdot x'^T (D - W) x'
\end{aligned}$$

$$\begin{aligned}
&\text{minimize } x'^T L x' \text{ subject to } x'^T D x' = 1 : \\
&\frac{\partial}{\partial x'} (x'^T L x + \lambda (1 - x'^T D x')) \\
&= 22 L x' - \lambda D x' = 0 \\
&= 22 L x' - \lambda D x' = \vec{0} \\
&L x' = \lambda D x' \\
\Rightarrow\ &D^{-1} L x' = \lambda x'
\end{aligned}$$

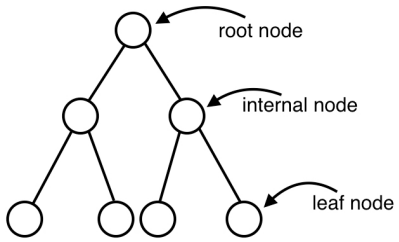
On 4.) The lower-dimensional embedding is obtained by selecting a ordered subset of eigenvectors from that decomposition.

- Sort eigenvectors by the magnitude of their associated eigenvalues
- Discard the eigenvalue/eigenvector pair for the lowest eigenvalues (the 0 eigenvalue) (number of values depends on connected components in graph)
- The $d'$ next smallest eigenvalue/eigenvector pairs $(e_1, e_2, \ldots)$ form the lower -dimensionanl embedding: $i$th row of $(e_1, e_2, ...e_d)$ represents the lower dimensioanl embedding of $x_i^T$.

*Addendum* to our Section on clustering: "Spectral clustering" is essentially executing Laplacian Eigenmaps + $k$-Means on the lower dimensional space.
How to choose a appropriate $d'$? The distribution of the eigenvalues gives an indication about the "intrinsic" dimensionality of the data. If there is a certain gap between eigenvalue, this is a good indication. We don't have this available when using LLE.

# Random Forest

*What is a random forest?* Is a learing based approach for analysing the feature space, with an ensemble of decision trees.
*What is a decision tree?* A binary tree with a "decision" at every internal and the root node. It's a learning based approach, because good decision functions at the internal nodes are the result of training.




*What is a decision?* For example the answer to the question: "Is this sample on the right side of a hyperplane?"
*Why an ensemble of trees?* Experience showed that it is complicated to train a single, highly accurate decision tree. The idea of random forests is therefore to train a large number of individually less accurate trees in a ranomized fashion, and to report then an averaged result of this forest. If we have a trained decision tree, we can test it by evaluating at the root node a function

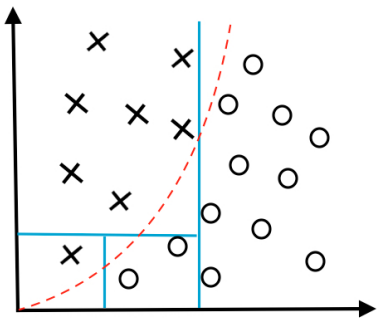
$$h(\vec{x}, \vec{\vartheta}_j) : \mathbb{R}^d \times \underset{\text{tree parameters}}{\mathcal{T}} \rightarrow \{0, 1\}.$$

Depending on the result (0 or 1) we evaluate the test function at either the left successor or the right successor of the root node, and continue down the tree recursively until a terminal/leaf node is reached. The leaf node performs an application-specific action. For example, if the task is to perfom classification, it assigns a label to the sample.

**Training:** compute the tree parameters $\vartheta$, consisting of

- the tree height/depth (trade-off[12])
- the splitting function at each internal node
- if necessary, the "action" in the leaf node

It is important to note that this "binary tree paradigm" essentially performs a partitioning of the feature space. More specifically, each internal node subdivides the incoming samples into two parts.



[12] deeper trees tend to overfitt, can be complemented with increased number of trees,

## Classification Forrests

Training of a single tree in a forest of size T

1. decide for a set of splitting function prototypes, e.g. hyperplanes or conics, ... (simpler functions are typically prefered. Simplest function: "axis aligned split")
2. (decide for randomization)
3. to find the parameters for the split function, select a suitable objective function.

(3.) "Solid choice": Information Gain [13] [14]

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

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

where $c$ denotes the class label, and $p(c)$ the empirical distribution computed from $S_j$.
(2.) The candidate functions, out of which the best on is chosen with the information gain, are randomly drawn. Typically, one decides:

- how many candidate functions are drawn
- if also linear projection of the data shall be drawn (e.g. consider only dimensions $\{d_{i_1}, d_{i_2}, \ldots, d_{i_n}\}$
- how the splitting parameters are sampled [15]

Choice of tree depth:

- set maximum depth ("mandatory")
- optional: set minimum number of samples for split
- depending on the application, stop if for example 99% of features in a node belong to one class

Final classifier output:

- at a leaf node, report the relative frequencies of the class labels in that node (e.g., 15%: class 1, 85% class 2)
- combine al trees by averaging the individual tree outputs. If a single discrete label is required, decide for the class with maximum probability.

Example: Classification (again)

1. Randomly select a number of splitting functions
2. Evaluate the information gain for each splitting function
3. Set the function with maximum information gain as the current nodes' decision function
4. Recursively repeat for the child nodes, until max tree depth (or some other criterium) is reached

[13] maximize the Information Gain for a good split
[14] where $S_j$ is the data that flows into the node, $S_j^L S_j^R$ is the data that flows to the left/right and $H()$ denotes the entropy. (*Note:* $I = H_{\text{before}} - H_{\text{after, weighted}}$)
[15] a sparser sampling leads to more "noise"/less optimal results (might be desired, e.g. prevents overfitting).

## Regression Forests

Goal: predict a continous label $p(y|\vec{x})$. (Remark: choice of the model complexity is related to the bias/variance trade off)
"Leaf prediction model": a base function that is fitted to the samples. The leaf prediction model could be constant (maximise the bias), linear, polynomial, ...
To faithfully represent all of the data with a single function, it would certainly make sense to use a polynomial model, or something even more complex. However, the random idea implies to subdivide/partition the space, and to fit simpler models to the individual partitions. As a specific example, let's split up our input data points:

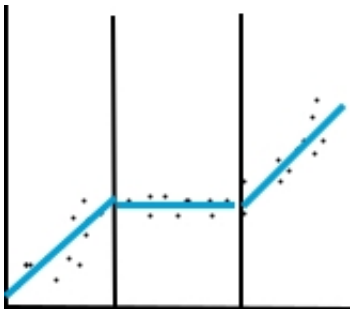


Figure 10: Regression split

The decision criterion for the splitting function works analogously to the classification case. The only difference ist that we need to define the entropy $H(S_j)$ on continous values:

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

where $p(y|x)$ can, e.g. be chosen as a Gaussian distribution $p(y|x) = \mathcal{N}(y; \overline{y}(x), \sigma_y^2(x))$, where $\overline{y}(x)$ is a linear function and $\sigma_y(x)$ is the conditional variance computed from a linear fit.

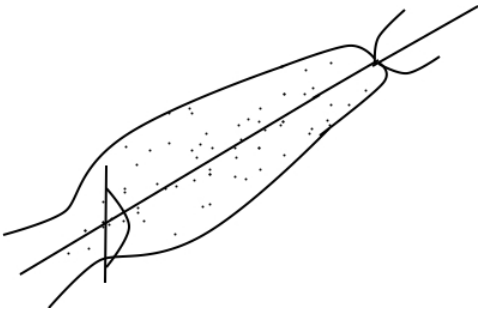


Figure 11: Probabilistic linear fit (left Gaussian, right constraint)

Combining the expression for $p(y|x)$ into $H(S_j)$ yields

$$H(S_j) = \frac{1}{|S_j|} \cdot \sum_{\vec{x} \in S_j} \frac{1}{2} \cdot log((2\pi e)^2 \sigma_y^2(\vec{x}))$$

$$\Rightarrow I(S_j, \vartheta) = \sum_{\vec{x} \in S_j} log(\sigma_y(\vec{x})) - \sum_{i \in \{L,R\}} (\sum_{x \in S_j^i} log(\sigma_y(\vec{x})))$$

## Density Forests

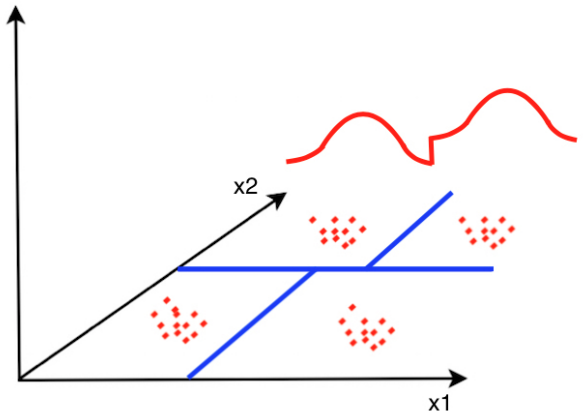
Very same idea, adapted to unlabelled data ⇒ learning-baed density estimator.
Each leaf node is modeled as a multivariate Gaussian distribution. The information gain metric can again be reused, i.e. $I(S_j, \vartheta) = H(S_j) - \sum_{i \in \{L,R\}} \frac{|S_j^i|}{|S_j|} \cdot H(S_j^i)$ but let us choose $H(S_j)$ as [16]

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

Plugging $H(S_j)$ back into $I(S_j, \vartheta)$ yields

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

.
In each leaf, fit a multivariate Gaussian distribution to the data in that leaf using, e.g. MLE.




**Note** that the fitted densities have discontinuities at the splitting boundaries (this is the same type of discontinuity that we observed for regression forests or classification forests). However, remember that the result of a random forest is averaged over all individual trees. Because of randomization, each tree splits at a slightly different location, and thus the discontinuities are "averaged out" in the forest.

[16] $\Lambda$ is the determinant of the covariance matrix of the data, "volume of a cluster"

## Manifold Forest (Manifold Learning with Random Forests)

**Idea:** Learn a partitioning of the feature space by training a density forest.[17] These partitions define a local neighborhood of the samples, which can be used to compute affinities[18], and then to apply e.g. Laplacian Eigenmaps on them.

**Task** Define affinities on a readily trained density forest.
Let $w_{ij} = e^{-Q(x_i,x_j)}$ denote the affinity between samples $x_i, x_j$, where $Q(x_i, x_j)$ denotes a distance function.
The "speciaility" of a Manifold Forest (in contrast, e.g. to standard Laplacian Eigenmaps) is that these distances are defined w.r.t. the leafs of the trees (i.e. the partitions).

Example choices of $Q$: Let $d_{ij} = x_i - x_j$, then

$$\text{Mahalanobis: } Q(x_i, x_j) = \begin{cases} d_{ij}^T (\Lambda_{l(x_i)})^{-1} d_{ij} & \text{if in the same leaf } l(x_i) \\ \infty & otherwise \end{cases}$$

$$\text{Binary: } Q(x_i, x_j) = \begin{cases} 0 & \text{if in the same leaf} \\ \infty & otherwise \end{cases}$$

Gaussian:...

In a single tree, only the samples within the same leaf-node have a non-zero affinity. Thus, a single tree produces a number of disconnected neighborhoods. However if the affinities are averaged over the whole forest, all points become connected.

$\Rightarrow$ We have a full adjacency matrix $A = \frac{1}{T} \sum_{t=1}^{T} w_i j^t$, where $T$ denotes the total number of trees.

$\Rightarrow$ Compute LE on $A$.

**Note** that the graph Laplacian in Ciminisi/Shotton are differently normalized, but it is the same algorithm:

$$L = I - \Gamma^{-\frac{1}{2}} A \Gamma^{-\frac{1}{2}}$$

where $\Gamma = D = \sum A_{ij}$ denotes the sum of the weights for sample $x_i$.

$\Rightarrow$: analogous to LE eigenvalue decomposition, arrange $d'$ eigenvectors that are associated with the lowest eigenvalues in a matrix $E$.

$$E = (e_1, e_2, \ldots, e_{d'})$$

The projection of $x_i$ onto a $d'$ dimensional space just corresponds to the $i$-th row of $E$.

[17] Note that we don't explicitly use the density
[18] Affinity intuition: Decreases with increasing distance

# Hidden Markov Models HMM

TODO:

## Remakrs on HMM

3 algorithmes

1. How to train the HMM (determain the parameters $\lambda(A, B, \pi)$)
2. Determine the propbaility of a symbole being produced
3. Recover the most likly state sequence

- the directed edge in a HMM graph can be understood as a statistical dependency $p(S_2|S_1)$ (more section 8 in bishops book on pattern recognition)
- generative approch
- For many tasks including speach processing, we often only allow state transitions $a_{ij}$ with $i \leq j$ (now backward links). So called "left-right-HMMs".

# Markov Random Fields (MRF)

Geman / Geman introduced MRF to image processing (1985). Example use cases are denoiseing, segmentation or stero matching.
- Consider the pixel gird as a lattice of random variables.



More specifically let us asume, that the image $F$ is given by the random matrix $[f_{i,j}]$.
Assumption: limited statisical depenency (between the pixels)

$$p(f_{i,j}|f_{i-1,j}f_{i,j-1}f_{i-1,j-1})$$

where $f_{i-1,j}f_{i,j-1}f_{i-1,j-1}$ form a dependency of the neigbours to $f_{i,j}$ (This will later be our Markov property)

$$p([f_{i,j}] = \prod_{i,j} p(f_{i,j}|f_{i-1,j}f_{i,j-1}f_{i-1,j-1})$$

**Definition of MRF:**
Let us consider the features / observations $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N$

1. Positivity: $p(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N) > 0$
2. Markokv property: $p(\vec{x}_k|\vec{x}_1, \ldots, \vec{x}_{k-1}, \vec{x}_{k+1}, \ldots, \vec{x}_N) = p(\vec{x}_k)|\mathcal{N}(\vec{x}_k))$, where $\mathcal{N}(\vec{x}_k))$ denotes the neighborhood of $\vec{x}_k$ ($\vec{x}_k$ only depends on its neighbors)

Definition of the neighborhood:

1. $\vec{x}_k \notin \mathcal{N}(\vec{x}_k))$
2. $\vec{x}_i \in \mathcal{N}(\vec{x}_k)) \rightarrow \vec{x}_k \in \mathcal{N}(\vec{x}_i))$
3. $\mathcal{N}(\vec{x}_k) = \{x_i|0 < dist(x_i, x_k) \leq t\}$

**Example:** Pixel gird

$$\mathcal{N}(\vec{x}_{i,j}) = \{x_{k,l}|(i-k)^2 + (j-l)^2 \leq c^2, \quad i \neq k \text{ or } j \neq l\}$$

Figure 12: 4-Neighborhood ($c = 1$), 8-Neighborhood ($c = \sqrt{2}$) and dinamic-Neighborhood



Figure 13: Image that explains the idea of MRF on images

**Graph Clique:** is a Complete subgraph:

**Gibbs Random Field (GRF):** is given by the PDF ($Z = \sum_{x'} H(x')$ is called a partition function and $H(x)$ is an energy function, i.e. a sum of potential functions.)

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

**Remark:** For a given PDF $p(x)$, the choice of the energy function is not unique. Consider for example

$$H(x) = -\log p(x) - \log Z$$

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

$\rightarrow$ we can choose $Z$ arbitarily

The interesting theoretical property is that GRFs and MRFs are equivalent. The proof for this is called Hammersley-Clifford Theorem.

**Example:** Image denoising
Given: The observed noisy image $[g_{i,j}]$
Searched: Hidden variables are the ideal (noiseless) image $[f_{i,j}]$

Assumption 1: The ideal image is spatially smooth

$$p(f_{i,j}]) = \frac{1}{Z} e^{-H([f_{i,j}])}, \quad \text{where } H([f_{i,j}]) = \sum_{i,j} ||\nabla f_{i,j}||_2^2$$

$H([f_{i,j}])$ (sum of squared gradients, computed over a neighborhood)

Assumption 2: $[g_{i,j}]$ is similar to $[f_{i,j}]$, but corrupted by additive Gaussian noise

$$p([g_{i,j}]|f_{i,j}]) = \prod_{i,j} \frac{1}{\sqrt{2\pi} G_{i,j}} \exp(-\frac{1}{2G_{i,j}^2}(f_{i,j} - g_{i,j})^2)$$

where $(f_{i,j} - g_{i,j})^2$ is the energy function $H$.
With these two functions defined, we can solve for a MAP estimate for $f$ (where $[\hat{f}_{i,j}]$ is the estimated idal image).

$$
\begin{aligned}
[\hat{f}_{i,j}] &= \arg\max_{[f_{i,j}]} p([f_{i,j}]|[g_{i,j}]) \\
&= \arg\max_{[f_{i,j}]} p([g_{i,j}]|[f_{i,j}])p([f_{i,j}]) \\
&\dots \\
&\arg\min_{[f_{i,j}]} \{\sum_{i,j} ||\nabla f_{i,j}||_2^2 + \sum_{i,j} \lambda_{i,j}(f_{i,j} - g_{i,j})^2\}
\end{aligned}
$$