Summary Pattern Analysis
SS 2020

Zhaoshun Hu

March 27, 2021

This page is empty on purpose.

Topics of SS2020

Representations of the Feature Space

1. Kernel-based Representations
2. Dynamic Feature Space Partitions Via Trees & Forests

Simplifications of the Feature Space

1. Hard and Soft Clustering
2. Dimensionality Reduction & Manifold Learning

Representations of Dependent Variables

1. Hidden Markov Models
2. Markov Random Fields

Spotlight on Model Selection

1 Representation of Feature Space

1.1 Kernel-based Representations

1.1.1 Probabilities and Probability Density Functions (PDFs)

Probability Equations

- Sum rule:$P(x) = \sum_Y P(X, Y)$
- Product rule: $P(x, y) = P(Y|X) * P(X)$
- Bayes theorem: $P(Y|X) = \frac{P(X|Y)*P(Y)}{P(X)}$

Probability Density Function(PDF)

- Non-negativity: $P(x) \geq 0$
- Unit area $\int_{-\infty}^{+\infty} p(x)\partial x = 1$

Algorithm for sampling form an arbitrary PDF $P(x)$

1. Discretize the domain of $P(x)$, normalize if between 0 and 1
2. if x is multivariate, linearize it in arbitrary order. Call the resulting density $P`(x)$
3. Calculate the cumulative density function $P`(z) = \int_{-\infty}^{2} p`(x)\partial x$
4. Draw a uniformly distributed number u, and find $z^* = argmin_z\ u > P`(z)$

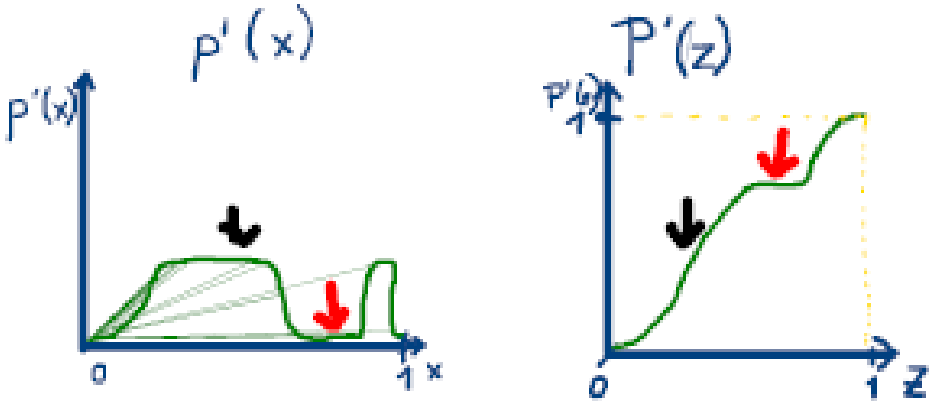

Figure 1: PDF to CDF

1.1.2 Parametric Density Estimation

Definition Determine best fitting function parameters for PDF
There is two method:

1. Maximum Likelihood(ML)
$$\theta^* = argmax_\theta P(x_1, x_2, ...x_N|\theta)$$
2. Maximum A Posterior (MAP) estimator
$$\theta^* = argmax_\theta P(\theta|x_1, x_2, ...x_N) = \frac{P(x_1, x_2, ...x_N|\theta * P(\theta)}{P(x_1, x_2, ...x_N)}$$

1.1.3 Non-Parametric Density Estimation

Feature Space Subdivision Let $P(x)$ be a PDF in D-dim. space, and x is contained in a small region R. The probability mass in R is $P = \int_R P(x)\partial x$

- Assumption 1: Large number of points $\rightarrow$ P is relative frequency

$$P = \frac{K}{N}$$

  which:

  - $K \leftarrow$ # points in R
  - $N \leftarrow$ total # points

- Assumption 2: R is so small that $P(x)$ is approximately constant:

$$P = \int_R P(x)\partial x = P(x) \int_R \partial x = P(x) * V$$

  which: V Volume of R

- Both assumption together give:

$$P(x) = \frac{K}{N * V}$$

  - Parzen window estimator: Variable $\Rightarrow \frac{K}{N}$
  - k-Nearest Neighbors: Variable $\Rightarrow V$

1.1.4 Parzen Window Estimator

Kernel Function

$$k(u) = \begin{cases} 1, & if|u_i| \leq \frac{1}{2}.i = 1....D \\ 0, & otherewise \end{cases} \tag{1}$$

Calculate K with this kernel function:

$$K = \sum_{n=1}^{N} k(\frac{x - x_n}{h})$$

which: h is the seize of the window
Whole density:

$$P(x) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} k(\frac{x - x_n}{h})$$

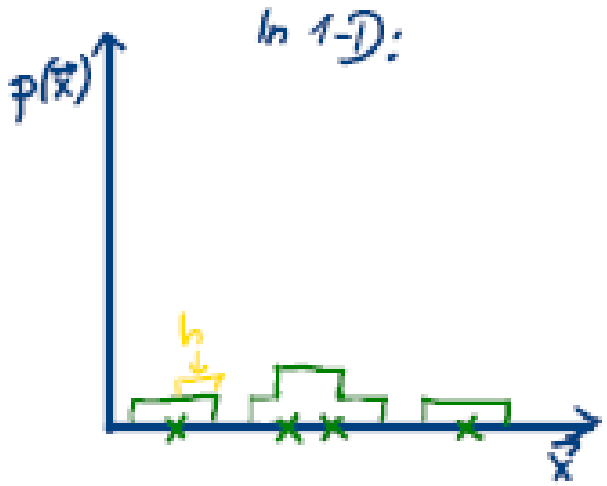

Figure 2: Parzen Window in 1-D

If we use Gaussian kernel:

$$P(x) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{\frac{D}{2}}} * e^{-(\frac{||x-x_n||_2^2}{2h^2})}$$

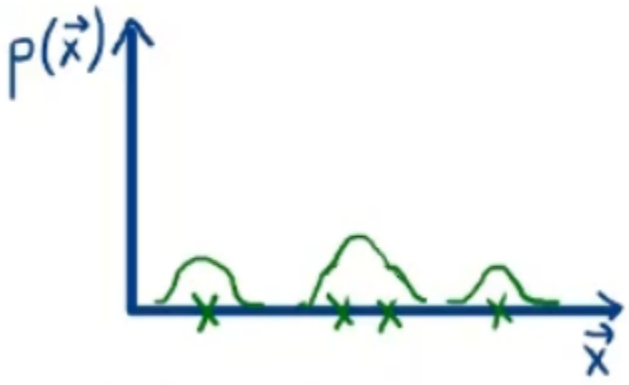

Figure 3: Parzen Window with Gaussian kernel

1.1.5 k-Nearest Neighbors(kNN)

Derived density relation ship $P(x) = \frac{K}{N*V}$

Algorithm

1. fix value for K
2. change Neighbor size V
3. until find the k nearest neighbors

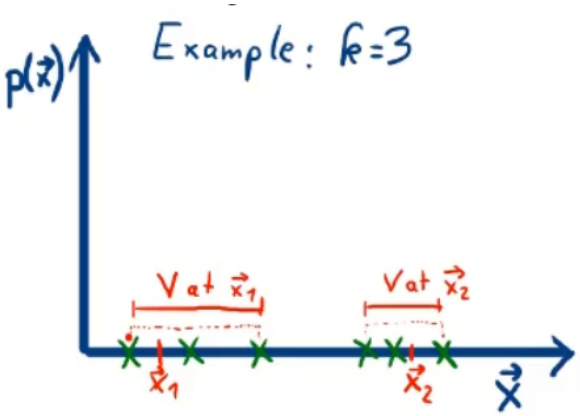

Figure 4: kNN with k=3

1.1.6 Model Selection Problem

Then kernel density estimator and the kNN density estimator are called "non-parametric", but they are not "parameter-free"

- the parzen window size h for kernel Density estimation
- the k in kNN

This parameter are so-called "hyperparameters" The task of hyperparameter optimization is call **model selection problem**

k-fold Cross-Validation Assume a data split into J folds:

$$s^j_{train} = \frac{s}{\{x_{|\frac{N}{J}|j}, ..., x_{|\frac{N}{J}|(j+1)-1}\}}, s^j_{test} = \frac{s}{s^j_{train}}$$

Let

- α denote the unknown hyperparameters
- $P_j(X|\alpha)$ the density estimate for fold S^j_{train} on hyperparameters α
- The Maximum Likelihood estimate is now obtained by evaluation

$$\alpha^* = argmax_\alpha \prod_{j=1}^{J} \prod_{x_i \in S^j_{train}} P_j(x_i|\alpha)$$

In practice, we take a logarithm of that equation to mitigate numerical issues. (because the products become sums)

1.1.7 Bias & Variance

- Bias: Estimation error due to model undercomplexity
- Variance. : Estimation error due to model overcomplexity

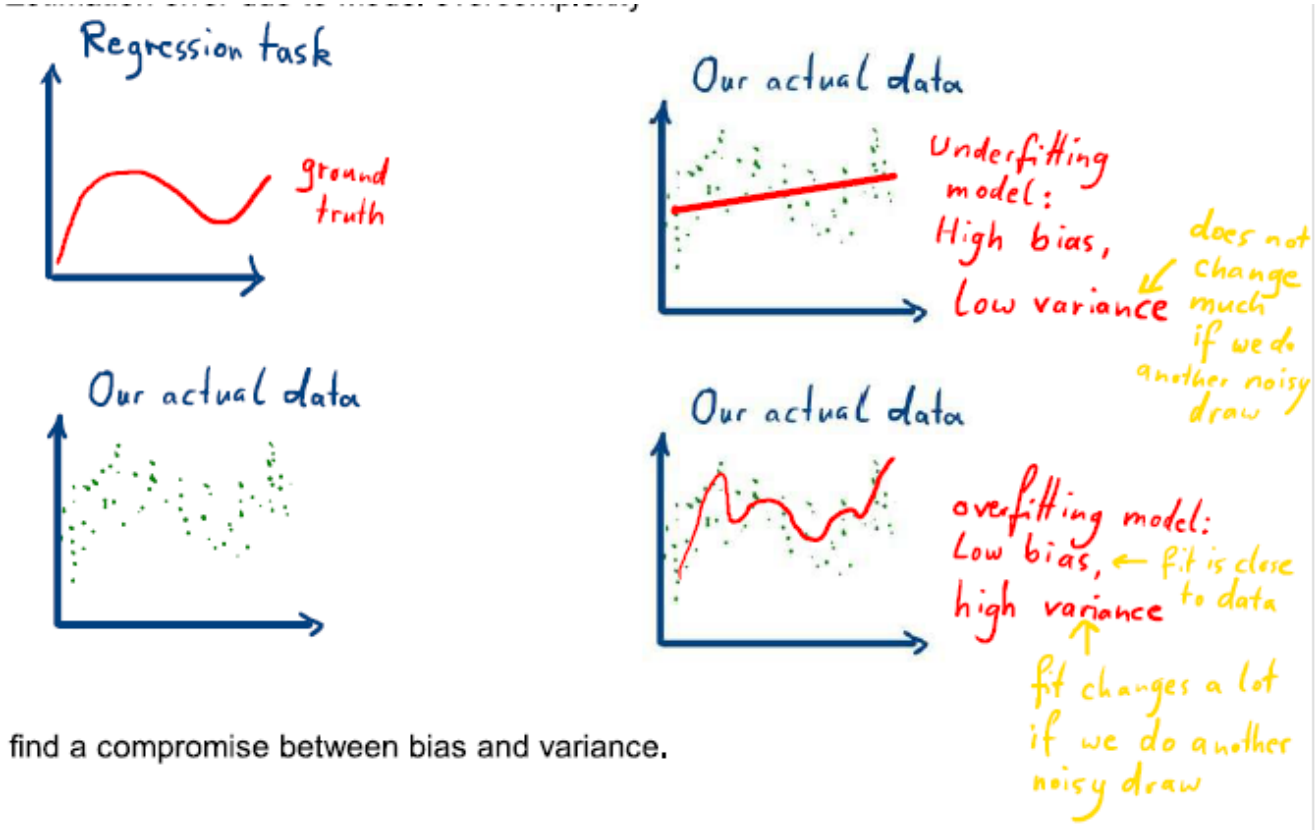

Figure 5: underfitting and overfitting

Goal: find a compromise between bias and variance.

1.1.8 Kernel Smoothing

Our kernel estimator is a memory-based methods require in principle little or no training
If we use NN kernel the line is not smoothing. But if we use the Epanechnikov Kernel, the line is smoothing

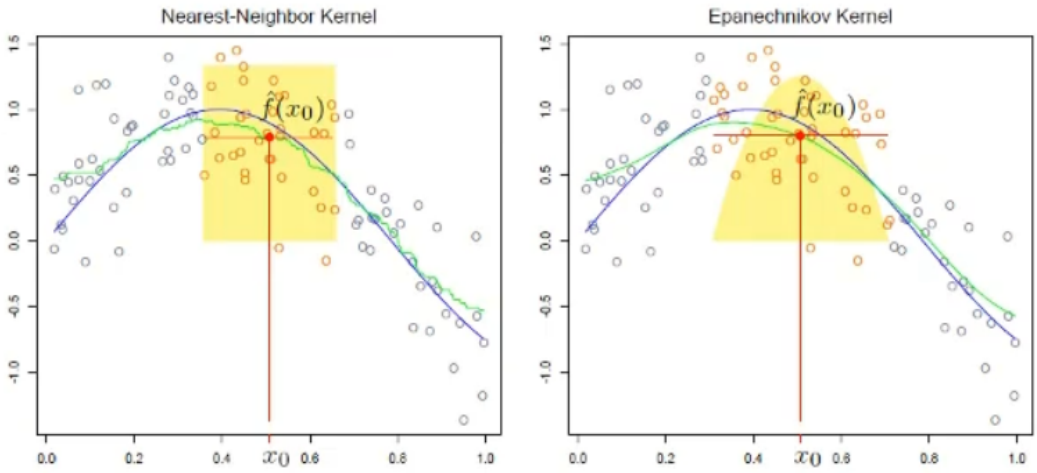

Figure 6: example for 2 kernels

Kernel Selection

Nadaraya-Watson kernel In short: weighted average

$$\hat{f}(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}$$

Epanechnikov quadratic kernel

$$K_\lambda(x_0, x) = D(\frac{|x - x_0|}{\lambda}),$$

with

$$D(t) = \begin{cases} \frac{3}{4}(1 - t^2), & if|t| \leq 1; \\ 0, & otherewise \end{cases} \tag{2}$$

The more generally we have:

$$K_\lambda(x_0, x) = D(\frac{x - x_0}{h_\lambda(x_0)})$$

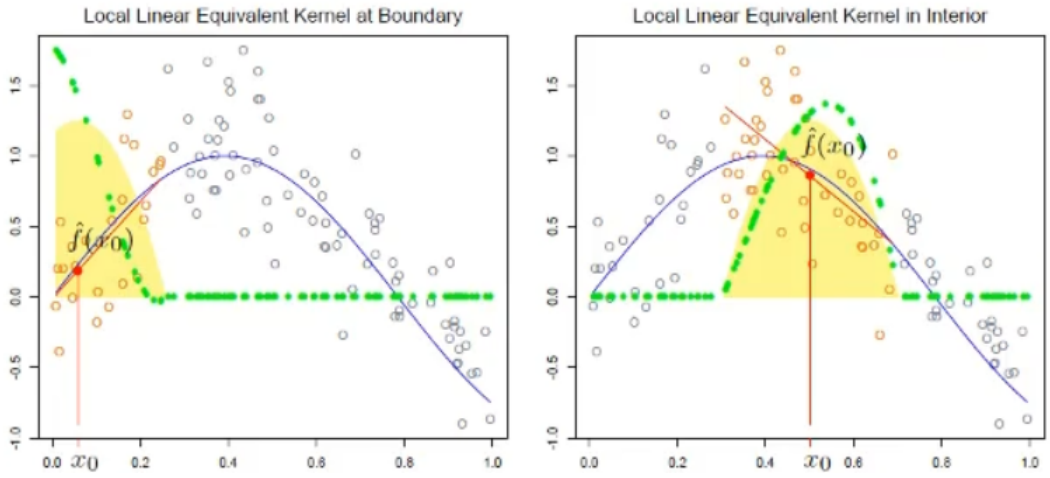

Figure 7: edge problem

Edge Problem

Boundary issues arise The metric neighborhoods tend to contain less points on the boundaries, while the NNs get wider.

Solution Do local linear regression

1.2 Dynamic Feature Space Partitions Via Trees & Forests

1.2.1 Classification and Regression Tree

Why We want more flexibility on density estimation.

Idea Hierarchical subdivision of the feature domain

Object Function Min the Object Function

1. Option 1: Negative cross-entropy:
$$Q_\tau(T) = \sum_{k=1}^{K} P_{\tau k} * ln\ P_{\tau k}$$
2. Option 2: Gini index:
$$Q_\tau(T) = \sum_{k=1}^{K} P_{\tau k} * (1 - P_{\tau k})$$

   which:

   - k: class label
   - K: total number of class
   - $P_{\tau k}$: relative frequency of samples from class k in region R_τ

both of them are better for tree growing than misclassification rate and they are differentiable

Algorithm

- Testing:

  Regression of the data in a leaf node is just the sample average

$$y_\tau = \frac{1}{N_\tau} \sum_{x_n \in R_\tau} t_n$$

- Training:

  - Greedy tree growth from root to leaves - At each node: exhaustive search for best split dimension & position
  - Objective function for regression:

$$Q_\tau(T) = \sum_{x_n \in R_\tau} (t_n - y_\tau)^2$$

  which:

  - $\tau : regionindex$
  - $R_\tau : region/leafw/inder\tau$
  - $t_n : samplelabel$
  - y_τ: sample average in R_τ

  In other words: find a partitioning into regions("a split"), such that this error is minimized.

- Terminate

  Grow each node until only few sample are left, the prune

  Pruning criterion is tradeoff between model accuracy & model complicity

$$C(T) = \sum_{\tau=1}^{|\tau|} Q_\tau(T) + \lambda |T|$$

  which λ is regularization parameter

Issues higg variance = tendency to overfit the training data

1.2.2 Random Forests

Why CART can really easy become overfitting.

Idea A Random Forest is CART + bagging + extra randomization
random forest additionally randomly draw at each node a subset of m feature dimensions for finding the next split. This further decorrelates the trees.

Object Function

$$\hat{f}(x) = \frac{1}{B} * \sum_{b=1}^{B} T(x; \theta_b)$$

which θ_b is the parameter of tree b

Algorithm The prediction of the whole RF is just the average of the individual tree predictions:
see idea

Remarks

- "escalation: of the weak learner idea:
  1. replace the exhaustive search for the best split by a random draw of candidate splits
  2. the candidates with the optimum value of the objective function is used.
- Out of Bag(OOB) Samples

  You can perform cross-validation on the fly by using a sample on those trees for validation where it was not part of the training bag.
- Overfitting behavior
  - larger numbers of trees effectively counter overfitting
  - extremely deep trees may overfit - hence a weak explicit or implicit depth limit may make sense
- Adaptive nearest neighbors:

  The combination of
  - the partitioning of the feature space into small tiles with few samples
  - the variability of these partitions across tress
  - the averaging of the individual tree responses effectively creates a data-driven version of our nearest neighbors

1.3 Shot Summary

We learn 2 method for DE in this Section

- kernel smoothing (non-parametric DE) (local operators from (memory-based) fixed neighborhood relationships)
- tree and tree-based ensembles (local operators from learning-based feature space partitioning)

2 Simplifications of Feature Space

2.1 Introduction

Properties of Clustering grouping of data into meaningful units
Goal: assign labels to local agglomerations of samples
Assumption: data somehow "belongs to groups" (without mathematically rigorous definition)
Applications: data exploration, quantization of bags of features, unsupervised classification
Method & Question:

- K-means
- Gaussian Mixture Models
- Mean-Shift
- Model Selection Problem: how can we estimate the hyperparameters?

Properties of Manifold Learning represent information in a lower-dimensional space
Applications: classification, regression, visualization, data exploration
Method & Question:

- PCA
- Multi-dimensional Scaling (MDS)
- A ride along notable non-linear methods to t-SNE:
  - ISOMAP
  - Laplacian Eigenmaps(LE)
- Model Selection Problem: what target dimension is appropriate?

Spotlight on applications of the Graph Laplacian,

- Graph Laplacian + k-means = spectral clustering (not talked in ss20?)
- LE + forest-based density estimation = forest-based manifold learning

2.2 Clustering

2.2.1 Mean Shift

Idea Straightforward extension of kernel smoothing: hill-climbing on the PDF to a mode($\rightarrow$ maximum)
Let us denote the multivariate kernel density estimate as

$$p(x) = \frac{1}{N}\sum_{k=1}^{N} K_\lambda(x_i, x)$$

The optima(and saddle points) of $p(x)$ are characterized by the points where its gradient $\nabla p(x)$ is equal to 0

$$\nabla p(x) = \frac{1}{N}\sum_{k=1}^{N} K_\lambda(x_i, x) = \frac{1}{N}\sum_{k=1}^{N} \nabla K_\lambda(x_i, x)$$

trick: they use a radial symmetric kernel

$$K_\lambda(x_i, x) = c_d * k_\lambda(||x_i - x||^2)$$

which: $||x_i - x||^2 = s$
First derivative: $\frac{\partial k_\lambda(x)}{\partial s} = k_\lambda^,(s)$
gradient ascent: compute the normalized (scaled gradient and follow the gradient direction)

Object Function

$$\frac{\partial k_\lambda(x)}{\partial s} = k_\lambda^,(s)$$

$$\Rightarrow \nabla(x) = \frac{1}{N}\sum_{i=1}^{N} c_d k_\lambda^,(||x_i - x||^2)(-2x_i - x) == 0$$

finally:

$$m^{(t)}(x) = \frac{\sum_{k=1}^{N} k_\lambda^,(||x_i - x||^2) * x_i}{\sum_{k=1}^{N} k_\lambda^,(||x_i - x||^2)} - x^{(t)}$$

Algorithm

1. Calculate the mean shift vector:

$$m^{(t)}(x) = \frac{\sum_{k=1}^{N} k_\lambda^,(||x_i - x||^2) * x_i}{\sum_{k=1}^{N} k_\lambda^,(||x_i - x||^2)} - x^{(t)}$$

2. Update position $x^{(t)}$:

$$x^{(t+1)} = x^{(t)} + m^{(t)}(x) = \frac{\sum_{k=1}^{N} k_\lambda^,(||x_i - x||^2) * x_i}{\sum_{k=1}^{N} k_\lambda^,(||x_i - x||^2)} - x^{(t)} + x^{(t)}$$

3. goto 1. unless convergence

Issues

- For gaussian we get better result but slowly to be converge.
- With Epanechnikov kernel: Mean shift vector is mean in λ-sphere, but Gauss kernel also possible.
- Numerical caveat: merge multiple nearby bumps if necessary

For gaussian we get better result but slowly to be converge.
With Epanechnikov kernel: Mean shift vector is mean in λ-sphere, but Gauss kernel also possible. Numerical caveat: merge multiple nearby bumps if necessary

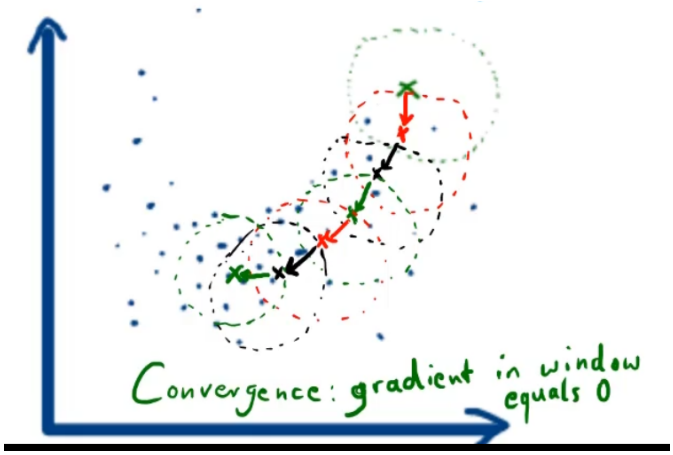

Figure 8: example of mean shift

Model Selection

Problem kernel size("bandwidth") impacts the result

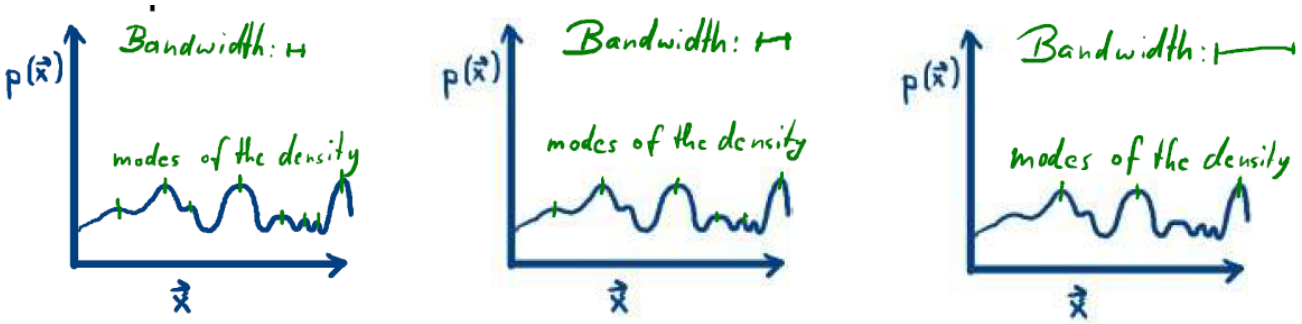

Figure 9: model selection problem for mean shift

Possible Solution For supervised tasks, cross-validation is a universal model selector.
However, using cross-validation(CV) for clustering is complicated by two issues:

1. clustering is unsupervised i.e., there are no "ground truth" labels as CV targets
2. also a transfer of the maximum likelihood estimator that we did for the bandwidth selection in kernel density estimation appears impossible:
    - there, we used a insertion trick: take one sample out, estimate density, check sample prediction. It is not clear how such a trick could be transferred to the clustering task

2.2.2 K-means

Arguably the most well-know hard clustering algorithm

Idea Optimizes k vectors that are place holders for these clusters, and the clusters shows in that way that they minimize the within-cluster distance $w(c)$ the squared Euclidean distance

$$W(c) = \frac{1}{2}\sum_{k=1}^{K}\sum_{c(i)=k}\sum_{c(i')=k}||x_i - x_{i'}||^2$$

$$= \sum_{l=1}^{K} N_k * \sum_{c(i)=k}||x_i - \mu_k||^2$$

which:

- $c(i) = k$: check whether x_i belongs to cluster k
- N_k: # points in cluster k
- μ_k: mean of all points in cluster k

Object Function The Optimization task is

$$min_{c\ \{m_k\}_1^k}\sum_{k=1}^{k} N_x \sum_{c(i)=k}||x(i) * m_k||^2$$

which $m_k = \mu_k$, other wise we are not optimal

Algorithm k-means algorithm: init: distribute cluster centers in sample space

1. (re-)assign each sample to nearest cluster center
2. calculate mean of each cluster from its assigned samples
3. goto 1. until convergence

Issues the solution is only locally optimal. hence different initializations can lead to different results
The iteration partitions the space, this is called a Voronoi tessellation

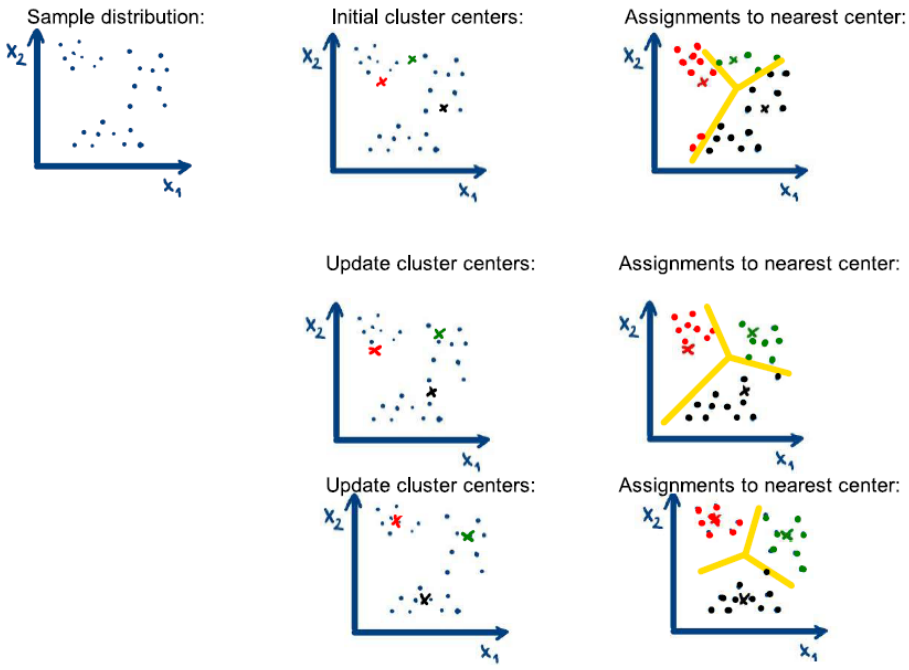

Figure 10: example of k-mean

Model Selection

Problem k impacts the result

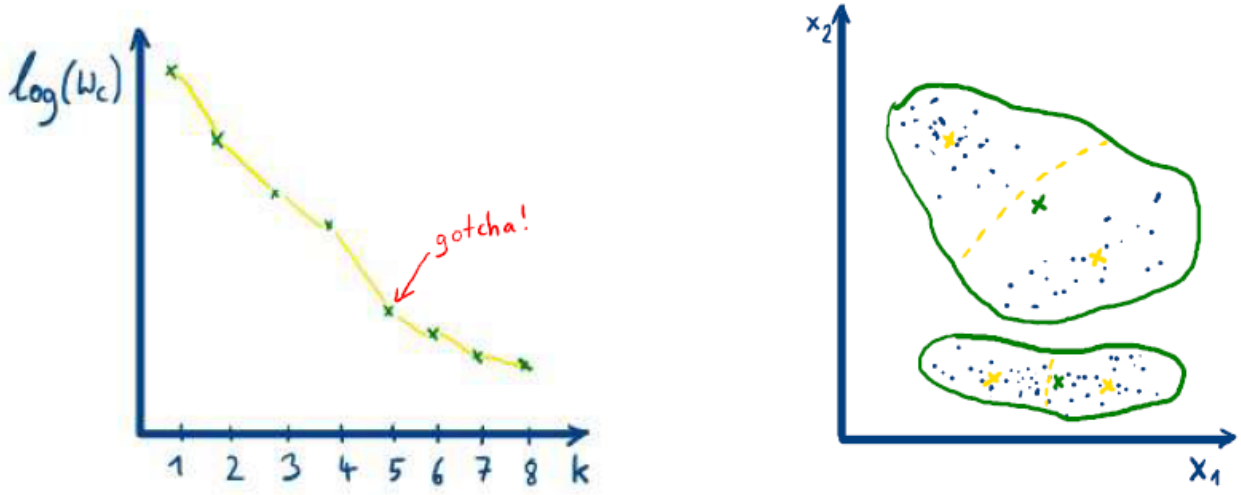

Figure 11: ellbow method

Possible Solution - ellbow method choosing a concrete threshold on the decrease is difficult: curve segments for smaller k have a "natural" larger decrease than curve segments for a larger k

Possible Solution - gap statistic normalize the curve of the within-cluster distance of the actual data with a curve of the within-cluster distances of data drawn from a uniform distribution.
The second curve is created from $B = 20$ Monte Carlo draws, and provides a model of some average "natural" decrease of the curve
The selection criterion for the number of clusters K^* is then

$$K^* = argmin_k\{K|G(K) \geq G(K+1) - s'_{K+1}\}$$

which:

- $G(K) = log(w_k^{unif} - log(w_k^{data}))$
- $s'_{K+1} = s_k\sqrt{1 + \frac{1}{20}}$
- s_k = stand deviation of $log(w_k^{unif})$ access 20 runs

2.2.3 Gaussian Mixture Models

Idea Gaussian Mixture Model:

$$p(x) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)$$

which:

- K: # of components
- π_k: weight of k-th component, $0 \leq \pi_k \leq 1, \sum_{k=1}^{k} \pi_k = 1$

To fit the Model, we use a K-dim. Binary indicator $z, z_k = \{0, 1\}, \sum_{k=1}^{k} z_k = 1$ As a result, the GMM can also be written as a marginalization over the hidden variables:

$$p(x) = \sum_z p(z)p(x|z) = \sum_{k=1}^{k} \pi_k * N(x|\mu_k, \Sigma_k)$$

Furthermore, we note that

$$p(z_k = 1|x) = \frac{p(z_k = 1)p(x|z_k = 1)}{p(x)} = \frac{\pi_k N(x|\mu_k \Sigma_k)}{\sum_{j=1}^{k} \pi_j N(x|\mu_j, \Sigma_j)}$$

Object Function EM iteratively optimizes the responsibilities of the hidden variables and the actual GMM parameters.
Given the responsibilities of the current iteration, the updated parameter are calculated from the log likelihood of the GMM. the log likelihood is:

$$ln\ p(x|\pi, \mu, \Sigma) = \sum_{n=1}^{N} ln(\sum_{k=1}^{k} \pi_k N(x_m|\mu_k, \Sigma_k))$$

As usual, the optimum values for μ_k, Σ_k, π_k are found by setting the derivative to 0.
The responsibilities $\gamma(z_k)$ are part of that optimum

$$\frac{\partial ln\ p(x|\pi, \mu, \Sigma)}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\pi_k N(x_m|\mu_k, \Sigma_k))}{\sum_{j=1}^{m} \pi_j N(x_m|\mu_j, \Sigma_j))} * \sum_{k}^{-1}(x_n - \mu_k)$$

Algorithm The final update equations:

- "Expectation" updates the responsibilities $\gamma(z_k)$
- "Maximization" updates the GMM params:
  - $\mu_k^{new} = \frac{1/N_k}{\sum}_{n=1}^{N} \gamma(z_{nk}) x_n$
  - $\Sigma_k^{new} = \frac{1/N_k}{\sum}_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k^{new})(x_n - \mu_k^{new})^T$
  - $\pi_k^{new} = \frac{N_k}{N}$
  - $N_k = \sum_{n=1}^{N} \gamma(z_{nk})$

Issues ???

2.3 Manifold Learning

2.3.1 PCA

Goal Get a projection onto the 1-D subspace that maximizes the variance, and show that this projection coincides with the largest eigenvector of the covariance matrix.

Idea Find a linear mapping $\Phi : R^d -> R^d, d' << d$, that maximizes the variance (spread) of the data along each dimension.

Object Function

$$J = \sum_{i,j=1}^{N} (\Phi_{x_i} - \Phi_{x_j})^T (\Phi_{x_i} - \Phi_{x_j}) - \lambda(\Phi^T \Phi - 1)$$

Algorithm

1. compute variance of data with:
$$\frac{1}{N} \sum_{i=1} N(u^T x_i - u^T \bar{x})^2 = u^T S u$$
2. compute covariance matrix:
$$S = \frac{1}{N} \sum_{i=1} N(x_i - \bar{x})(x_i - \bar{x})^T$$
3. find that u that maximizes the variance:
$$u^T S u + \lambda(1 - u^T u) -> max$$

   Maximization is done by calculating the derivative u and to set the equation equal to 0:

$$\frac{\partial}{\partial u} u^T S u + \lambda(1 - u^T u) \neq 0$$

$$2Su = 2\lambda u$$

$$Su = \lambda u$$

Issues This is just the eigenvector decomposition of S. The maximum covariance is obtained from the eigenvector associated with the largest eigenvalue. This vector is called a "principal component".

2.3.2 MDS

Idea MDS is equivalent to PCA. but it take distances between points as inputs.

Object Function

$$SVD(\frac{1}{2}CD^2C) = SVD(X^TX) = U^T\Sigma V$$

$$\Rightarrow X = \Sigma^{\frac{1}{2}}U$$

Algorithm

1. Given a distance matrix
$$d_{ij}^2 = (x_i - x_j)^T(x_i - x_j) = x_i^Tx_j + x_ix_j^T - 2x_i^Tx_j$$
$$\Rightarrow D^2 = diag(x^Tx) * \mathbb{I} + \mathbb{I} * diag(x^Tx)^T - 2x^Tx$$
2. $$SVD(\frac{1}{2}CD^2C) = SVD(X^TX) = U^T\Sigma V$$
with $C = (I - \frac{1}{N}\mathbb{I}\mathbb{I}^T)$
3. $$\Rightarrow X = \Sigma^{\frac{1}{2}}U$$

Model Selection Issues How many dimensions shall the target space have?
For PCA, one very commonly seen approach is "to preserve x% of variance", by selecting the N largest eigen values, such that

$$d^{,*} = argmin_d\ x\% \leq \frac{\sum_{i=1}^{d}}{\sum_{i=1}^{d^,}}$$

Independent of that, there is for all methods that rely on an eigen decomposition or a singular value decomposition the concept of the **"eigenvalue gap"**
In theory, the eigenvalue distribution exhibits a gap when hitting the data's intrinsic dimensionality.

2.3.3 ISOMAP

Why Straightforward non-linearity hack for MDS: find a mapping that preserves the global non-linear geometry of data by preserving the "geodesic" distances along the manifold

Idea Replace the Euclidean distances in MDS with shortest distances in a graph.

Object Function Same as MDS but with different input Matrix

Algorithm

1. Calculate the edges in the graph via Euclidean distances within a local neighborhood.(k-nearest neighbors or fixed threshold)
2. Calculate the all-pairs shortest paths between all samples to obtain the distances matrix.
3. Perform MDS on that distance Matrix

Issues Sensitive with noise, because compute the shortest path in the Matrix.

2.3.4 LE

Idea LE does not consider such global structure at all.
Instead, if focuses on the preservation of local neighborhoods.
This makes LE intrinsically better suited than ISOMAP to model non-linear manifolds

Object Function

$$\sum_{i=1}^{N}\sum_{j=1}^{N}(x_i^{,}-x_j^{,})^2 w_{ij} \to min$$

Let us rewrite the objective function:

$$\sum_{i,j}(x_i - x_j)^2 * w_{ij} = \sum_{i,j}(x_i^2 + x_j^2 + 2x_i x_j) * w_{ij}$$

$$= 2 * \sum_i x_i^2 (\sum_j w_{ij}) - 2\sum_{i,j} x_i x_j * w_{ij}$$

$$= 2(x^T D x - x^T W x)$$

$$= 2(x^T (D - W) * x)$$

The matrix $L = D - W$ is also known as the graph Laplacian.
Minimize $x^T : x$ subject to $xDx = 1$

$$\Rightarrow \frac{\partial}{\partial x}(x^T L x + \lambda(1 - xDx))$$

$$= 2Lx - \lambda Dx \neq= 0$$

$$Lx = \lambda DX \Rightarrow D^{-1}Lx = \lambda x$$

Algorithm

1. Build adjacency graph from the samples (e.g., kNN or fixed distance threshold)
2. Weight the edges in the graph
3. Calculate an eigen decomposition of the graph Laplacian
4. Perform low-dimensional embedding

2.4 Laplacian Eigenmaps and Random Forests

2.4.1 Density Forest

Information Gain Information gain:

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

Can not work on unsupervised learning.

"Volume Gain"

$$H(s_j^i(\theta)) = \frac{1}{2} log((2\pi e)^d |\Lambda(s_j^i(\theta))|)$$

$$I(s_{j,i}\theta) = log(|\Lambda(s_j)|) - \sum_{i=\{L,R\}} \frac{|s_j^i(\theta)|}{|s_j|} log(|\Lambda(s_j^i(\theta))|)$$

Now fit for unsupervised learning

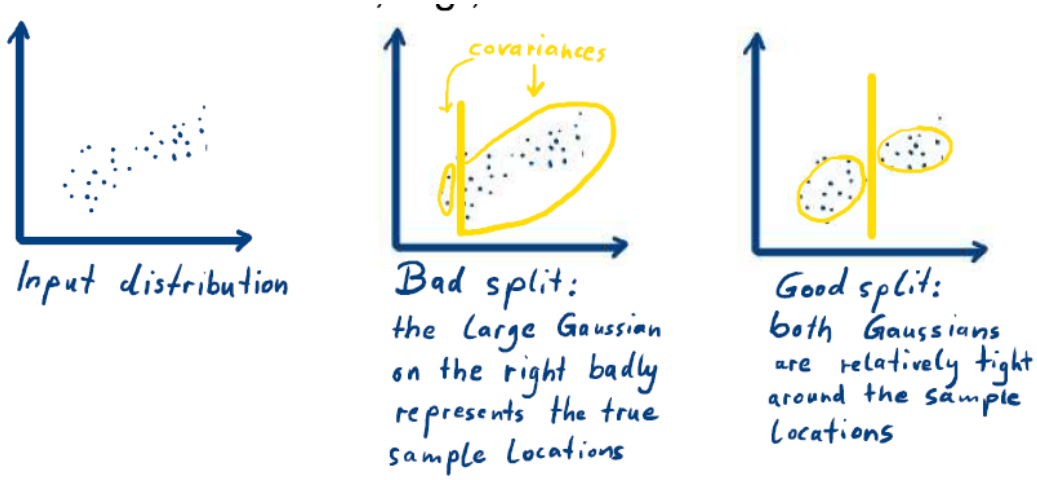

Figure 12: density forests and volume

DF use for DE Each individual tree provides a mixture of truncated Gaussian in 1-D:

Figure 13: DF use for DE

Averaging the output of multiple trees of the forest smoothes the boundaries:
To sample from that density:

1. randomly select a tree,
2. select a leaf w/ probability acc. to the # of leaf samples
3. sample from that truncated Gaussian

2.4.2 Manifold Forests

Idea The Density Forest provides an unsupervised learned partitioning of the feature space

Algorithm - Compute affinities matrix Define affinities on the leaves of the Density Forest.

1. For each tree t, define an affinity matrix $w^t : w^t_{ij} = e^{-d^t(x_i,x_j)}$
   where: $-d^t(x_i, x_j)$ is a distance function,
2. For each tree, w^t has the shape of a block matrix (induced by the leaves)
   $w = \sum_{t=1}^{N} w^t$ is the average over the whole forest/
3. Averaging removes the block structure.

Distance Function

- Gaussian affinity:

$$d^T(x_i, x_j) = \begin{cases} \dfrac{(x_i - x_j)^T(x_i - x_j))}{t}, & if\ leaf(x_i) = leaf(x_j) \\ \infty, & otherwise \end{cases} \tag{3}$$

- Binary affinity:

$$d^T(x_i, x_j) = \begin{cases} 0, & if\ leaf(x_i) = leaf(x_j) \\ \infty, & otherwise \end{cases} \tag{4}$$

Algorithm - implement into LE We can again calculate an eigenvalue decomposition on L, and consider the smallest eigenvalues in increasing order.
To obtain a lower-dimensional embedding:

1. Discard all eigenvectors with associated eigenvalue $\lambda_i = 0$
2. Arrange the eigenvectors associated with the next d' smallest eigenvalues in a matrix: $E = (e_1, ..., e_{d'})$
3. The lower-dimensional embedding of point x_i corresponds to the i-th row of matrix E.

3 Representations of Dependent Variables

3.1 Introduction

- HMM = a probabilistic model aims to express the properties of our data in a probabilistic representation
- MRS = the "graphical" part aims to make these properties better understandable through visualization in a graph of probabilistic relationships.

3.2 Hidden Markov Models(HMMs)

3.2.1 Definition

The joint PDF in the sense of the previous lecture is $p(\underbrace{x_1, .., x_N}_{observations}, \underbrace{y_1, ..., y_N}_{hidden\ state}) = p(x_1, .., x_N|y_1, ..., y_N)p(y_1, ..., y_N)$
We assume that both $x_1, .., x_N$ and $y_1, ..., y_N$ are sequences
We require assumptions to further factorize the equation
We use these assumptions:

1. Observations only depend on a single hidden state $p(x_1, .., x_N|y_1, ..., y_N) = \prod_{i=1}^{N} p(x_i|y_i)$
2. A hidden state depends only on its predecessor ("Markov assumption") $p(y_1, ..., y_N) = p(y_1) \prod_{i=2}^{N} p(y_i|y_{i-1})$

This leads to the factorization

$$p(x_1, .., x_N, y_1, ..., y_N) = p(y_1) \prod_{i=1}^{N} p(x_i|y_i) \prod_{i=2}^{N} p(y_i|y_{i-1})$$

This factorization consists of many small factors, which makes the problem tractable.
The corresponding sketch as a graphical model is:

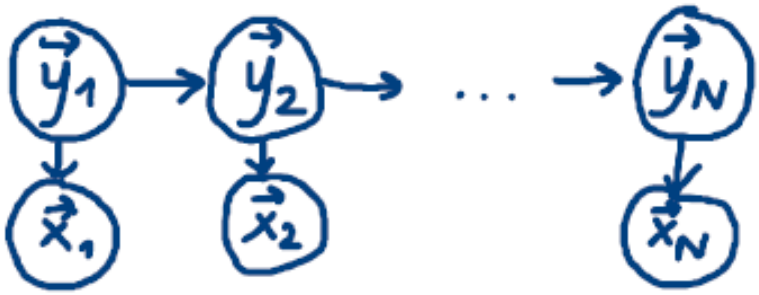

Figure 14: HMM

HMMs with parameters $\lambda = (A, B, \pi)$

1. Matching algorithm: calculates $p(< o_1, ..., o_N > |\lambda)$. e.g. forward or backward algorithm
2. Most likely state sequence: calculate $argmax_{q_1,...,q_N}\ p(< o_1, ..., o_N >, < q_1, ..., q_N > |\lambda)$. e.g. Viterbi algorithm
3. HMM training: calculates $\lambda = (A, B, \pi)$. e.g. Baum-Welch

3.2.2 Three Algorithm - Matching Algorithm

$p(< o_1, ..., o_N >)$ can be calculated by marginalization over all state sequences from the full model

$$p(< o_1, ..., o_N >) = \sum_{<q_1,...,q_N>} p(< o_1, ..., o_N >, < q_1, ..., q_N >) = \sum_{<q_1,...,q_N>} p(< o_1, ..., o_N > | < q_1, ..., q_N >) p(< q_1, ..., q_N >)$$

It can be directly calculated via summation over all:

$$p(< o_1, ..., o_N >) = \sum_{q_1=1}^{M} \sum_{q_2=1}^{M} ... \sum_{q_N=1}^{M} \pi_{q_1} b_{q_1}(o_1) a_{q_1 q_2} b_{q_2}(o_2) a_{q_2 q_3} ... b_{q_{N-1}}(o_N) a_{q_{N-1} q_N}$$

complexity is $o(N^M)$
But we can use dynamic programming by using markov properly

$$p(< o_1, ..., o_N >) = \sum_{q_1=1}^{M} \pi_{q_1} b_{q_1}(o_1) \sum_{q_2=1}^{M} a_{q_1 q_2} b_{q_2}(o_2) ... \sum_{q_N=1}^{M} a_{q_{N-1} q_N} b_{q_{N-1}}(o_N)$$

complexity is $o(NM^2)$

Forward algorithm

1. $t = 1 : \ \alpha_1(q_1) = \pi_{q_1} * b_{q_q}(o_1) \ \forall 1 \leq q_1 \leq M$
2. $1 \leq t \leq N : \ \alpha_t(q_t) = (\sum_{q_{t-1}}^{M} \alpha_{t-1}(q_{t-1}) a_{q_{t-1} q_t}) b_{q_t}(o_t) \ \forall 1 \leq q_i \leq M$
3. $t = N : \ p(< o_1, ..., o_N >) = \sum_{q_N=1}^{M} \alpha_N(q_N)$

Backward algorithm

1. $t = N : \beta_N(q_N) = 1 \ \forall 1 \leq q_N \leq M$
2. $1 \leq t \leq N : \beta_t(q_t) = \sum_{q_{t+1}=1}^{M} a_{q_t q_{t+1}} b_{q_{t+1}}(o_{t+1}) \beta_{t+1}(q_{t+1}) \ \forall q \leq q_t \leq M$

3.2.3 Three Algorithm - Finding the most likely state sequence

Core:

$$\delta(i) = argmax_{<q_1,...,q_N>} \ p(< o_1, ..., o_t >, < q_1, ..., q_t - i > | \lambda) \ \forall t \leq N$$

where $\delta(i)$ is the most likely subpath of the time $1 \leq t \leq N$ that ends in state q_i
One additional variable $\Psi_t(i)$ that tracks the predecessor state, in order to reconstruct the path in the last algorithm step.

Viterbi algorithm

1. $t = 1 : \ \delta_1(i) = \pi_i b_i(o_1) \ \forall 1 \leq q_1 \leq M \Psi_1(i) = 0$
2. $2 \leq q_1 \leq N : \ \delta_t(i) = max_{1 \leq i \leq M} \ (\delta_{t-1}(i) a_{ij}) b_j(o_i) \ \forall 1 \leq j \leq M \Psi_1(i) = argmax_{1 \leq i \leq M} \ \delta_{t-1}(i) a_{ij}$
3. $p^* = max_{1 \leq i \leq M} \delta_N(i), \ q_N^* = argmax_{1 \leq i \leq M} \ \delta_N(i)$
4. Path reconstruction: $q_t^* = \Psi_{t+1}^*(q_{t+1}^*) \ \forall 1 \leq i \leq M$

3.2.4 Three Algorithm - HMM Training

Baum-Welch-Algorithm

- Probability for transition $s_i -> s_j$ at time t: $\xi_t(i,j) = p(q_t = s_i, q_{t+1} = s_j)$
- Probability of being in state s_i at time t: $\delta_t(i) = \sum_{j=1}^{M} \xi_t(i,j)$
- the expected # of transitions $s_i -> s_j$ at all times: $\sum_{t=1}^{N} \xi_t(i,j)$
- the expected # of times s_j is visited/expected # of transition from state s_i: $\sum_{t=1}^{N} \delta_t(i)$

Maximization:

$$\bar{\pi}_i = \delta_1(i) \Rightarrow Expected\ \#\ of\ times\ in\ s_i\ at\ time\ t = 1$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{N-1} \xi_t(i,j)}{\sum_{t=1}^{N-1} \delta_t(i)} \Rightarrow \frac{Expected\ \#\ of\ transition\ s_i -> s_j}{Expected\ \#\ of\ times\ in\ state\ s_i}$$

$$\bar{o}_j(K) = \frac{\sum_{t=1}^{N-1} \xi_t(j) \mathbb{I}(o_t = v_k)}{\sum_{t=1}^{N-1} \delta_t(i)} \Rightarrow \frac{Expected\ \#\ of\ times\ \in\ s_i\ and\ obserbing\ v_k}{Expected\ \#\ of\ times\ in\ state\ s_i}$$

3.3 Markov random fields(MRFs)

3.3.1 Definition

The conditional independence introduced by the markov blanket factorizes joint probabilities:

$$p(x_i, x_j | x_{k \neq i,j}) = p(x_i | x_{k \neq i,j}) p(x_j | x_{k \neq i,j)}$$

The Markov Random Field is typically written in terms of such potential functions:

$$p(x_1, ..., x_N) = \frac{1}{z} \prod_c \phi_c(x_c)$$

where:

- $z = \sum_{all\ value\ combinations\ for\ x_1,...,x_N} \prod_c \phi_c(x_c)$
- $x_1, ..., x_N$: random variables with discrete state
- c: set of all maximal cliques in the graphical model
- $\phi_c(x_c)$: potential function over a clique with variable x_c.

Affinities with exponentials of energy functions $\phi_c = e^{-E(x_c)}$

3.3.2 Inference via Gibbs Sampling

We can define the clique potentials as

- $e^{-E(x_i,y_i)}$ with the energy term $E(x_i, y_i) = -\eta x_i y_i$ here η is positive, constant weighting factor.
- $e^{-E(y_i,y_j)}$ with the energy term $E(y_i, y_j) = -\beta y_i y_j$

3.3.3 Objective Function for MRF Inference

Consider a joint distribution of observation and hidden variables:

$$p(x_1, ..., x_N, y_1, ..., y_N) = p(x_1, ..., x_N | y_1, ..., y_N) p(y_1, ..., y_N)$$

We seek to find the assignment to variables that maximized the joint probability

$$argmax_{y_1,...,y_N} \; p(x_1, ..., x_N | y_1, ..., y_N) p(y_1, ..., y_N)$$

We make the flowing independence assumptions:

- observations are mutually independent
- each observation depends only on its associated hidden variable
- each hidden variable only depends on its neighboring hidden variables

This allows to further factorize the objective function into:

$$argmax_{y_1,...,y_N} \prod_{i=1}^{N} p(x_i|y_i) \prod_{i=1, y_i \in N(y_i)}^{N} p(y_i|y_j) = argmax_{y_1,...,y_N} \; \frac{1}{z} e^{-\sum E(x_i,y_i) - \sum E(y_i,y_j)}$$

where $N(y_i)$: set of neighboring hidden variables of y_i Larger potentials are closer to the solution

- Potentials with energy terms $E(x_i, y_i)$ called "unary potentials"
- Potentials with energy terms $E(y_i, y_j)$ called "pairwise potentials"

3.3.4 Gibbs sampling

While (infinity):

1. select a node y_i
2. Fix the assignments to all its neighbors
3. Assign a new label to y_i if this yields a lower energy

3.3.5 MRF inference via Graph Cuts

Constraints

- requires binary labels
- maximal clique size is 2
- pairwise energies must satisfy "submodularity condition"

Benefits

- globally optimal labeling
- polynomial-time algorithm