Contents

1 Introduction

- Regression: Vector $\vec{x} \in \mathbb{R}^d \to y \in \mathbb{R}^m$
- Example: quadratic function fitting
- Training Set: $S = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N\}$
- $f(\vec{x})$ quadratic function: $x_2 = y_1 x_1^2 + y_2 x_1 + y_3$

Estimation of $\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$

Objective function for $\{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N\}$:

$$J = \sum_{i=1}^{N} \left(x_{i,2} - (y_1 x_{i,1}^2 + y_2 x_{i,1} + y_3)\right)^2$$

$$\hat{y}_1, \hat{y}_2, \hat{y}_3 = \underset{y_1, y_2, y_3}{\arg\min}\, J$$

(Distance in squared L2 norm.)

$$\frac{\partial J}{\partial y_1} = \sum_{i=1}^{N} 2\left(x_{i,2} - (y_1 x_{i,1}^2 + y_2 x_{i,1} + y_3)\right) \cdot x_{i,1}^2 \overset{!}{=} 0$$

$$\frac{\partial J}{\partial y_2} = \sum_{i=1}^{N} 2\left(x_{i,2} - (y_1 x_{i,1}^2 + y_2 x_{i,1} + y_3)\right) \cdot x_{i,1} \overset{!}{=} 0$$

$$\frac{\partial J}{\partial y_3} = \sum_{i=1}^{N} 2\left(x_{i,2} - (y_1 x_{i,1}^2 + y_2 x_{i,1} + y_3)\right) \cdot 1 \overset{!}{=} 0$$

(Factor -1 does not matter due to $\overset{!}{=} 0$)
System of linear equations:

$$A \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \vec{b}$$

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \underbrace{(A^T A)^{-1} A^T}_{\text{Pseudo-Inverse}} \vec{b}$$

$$A = \begin{pmatrix} \sum_{i=1}^N x_{i,1}^4 & \sum_{i=1}^N x_{i,1}^3 & \sum_{i=1}^N x_{i,1}^2 \\ \sum_{i=1}^N x_{i,1}^3 & \sum_{i=1}^N x_{i,1}^2 & \sum_{i=1}^N x_{i,1} \\ \sum_{i=1}^N x_{i,1}^2 & \sum_{i=1}^N x_{i,1} & N \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} \sum_{i=1}^N x_{i,2} \cdot x_{i,1}^2 \\ \sum_{i=1}^N x_{i,2} \cdot x_{i,1} \\ \sum_{i=1}^N x_{i,2} \end{pmatrix}$$

(This is "homework", so it's not checked!)

Classification (pattern recognition)

- feature vector $\vec{x} \in \mathbb{R}^d$
- class number $y \in \{1, 2, \dots, K\}$
- classification: $\zeta(\vec{x}) = y$

PR:

- d is constant
- $y \in \{\pm 1\}$

Example: Bayes classifier

- Loss function: compute the classifier ζ that minimizes the average loss.
- Common loss function: $(0, 1)$-loss function
- Optimal decision rule (Bayes classifier):

$$\begin{aligned} y = \zeta(\vec{x}) &= \arg\max_y \frac{p(y)p(\vec{x}|y)}{p(\vec{x})} \\ &= \arg\max_y p(y)p(\vec{x}|y) \\ &= \arg\max_y p(\vec{x}, y) \end{aligned}$$

$p(y\|\vec{x})$	posterior probability
$p(\vec{x})$	evidence
$p(y)$	prior (probability)
$p(\vec{x}\|y)$	class conditional
$p(\vec{x}, y)$	joint density

2 Clustering (unsupervised learning)

Properties of distance meassures (4)

- non-negativity: $d(\vec{x}_1, \vec{x}_2) \geq 0$
- $d(\vec{x}_1, \vec{x}_2) = 0 \iff \vec{x}_1 = \vec{x}_2$
- symmetry: $d(\vec{x}_1, \vec{x}_2) = d(\vec{x}_2, \vec{x}_1)$
- triangle equation: $d(\vec{x}_1, \vec{x}_3) \leq d(\vec{x}_1, \vec{x}_2) + d(\vec{x}_2, \vec{x}_3)$

$$\hat{\vec{\mu}}_1, \dots, \hat{\vec{\mu}}_K, \hat{C} = \arg\min_{\vec{\mu}_1, \dots, \vec{\mu}_K, C} \sum_{i=1}^{N} \sum_{j=1}^{K} c_{ij}^q d(\vec{x}_i, \vec{\mu}_j) \text{ s.t. } \forall i : \sum_{j=1}^{K} c_{ij} = 1$$

- Hard Clustering
  - $q = 1$
  - $c_{ij} \in \{0, 1\}$
  - Optimize μ_j by setting the derivative of the Lagrangian to zero
  - k-Means Algo.: L_2^2-norm $\Rightarrow c_{ij} = 1$ for closest μ_j, $\mu_j =$ mean $\{\vec{x}_i | c_{ij} = 1\}$
  - Generalized Hard Clustering (arbitrary norm)
- Soft Clustering
  - $q > 1$, larger $q \to c_{ij}$ "fuzzier" / closer to a uniform distribution
  - Optimize both μ_j and c_{ij} by setting the derivative of the Lagrangian to zero
  - Using $1 = \sum_{j=1}^{K} c_{ij} = \dots$ allows direct computation of λ_i, used in c_{ij}
  - Soft Clustering Algo.: Repeatedly compute c_{ij} and μ_j until μ_j converged
- Variations of Clustering / additional regularizers (added to the lagragians)
  - Include a feature transform (PR)
  - μ controls the ratio between the regularizer and the original optimization
  - *Maximize* distance between clusters $-\mu \sum_{k,l=1}^{K} ||\vec{\mu}_k - \vec{\mu}_l||$ ($-$ due to max.!)
  - Covariance matrix of cluster j: $\Sigma_j = \frac{1}{\sum_{i=1}^{N} c_{ij}} \sum_{i=1}^{N} c_{ij} (\vec{x}_i - \vec{\mu}_j)(\vec{x}_i - \vec{\mu}_j)^T$
  - Σ_js shall be close to C': $+\sum_{j=1} \mu_j ||\Sigma_j - C'||_F$, e.g. $C' = I \cdot \sigma^2$ (diagonal)

3 PDF Estimation

Core assumption: The data points $\vec{x}_i$ are sampled from the underlying PDF.

Properties of PDFs (3)

- non-negativity: $p(\vec{x}) \geq 0$ (possibly > 1!)
- integration to 1: $\int_{-\infty}^{\infty} p(\vec{x})\, \mathrm{d}\vec{x} = 1$
- probability of intervals: $p(\vec{a} \leq \vec{x} \leq \vec{b}) = \int_{\vec{a}}^{\vec{b}} p(\vec{x})\, \mathrm{d}\vec{x} \in [0, 1]$

3.1 Parametric Density Estimation

Justification of parametric PDE

- PDF is known by the construction of the samples (result of PCA)
- Statistical testing, e.g. Kolmogorov Smirnov test for Gaussianity (neg. Kurtosis)
- Engineering approach (choose parametric family, estimate parameters and test)

Maximum Likelihood Estimation (MLE)

- assumption 1: independency
- assumption 2: equal priors (MLE = MAP with $p(\vec{\theta})$ = uniform distrib. / const.)
- example: assume normal distrib., derive params. μ and σ^2 by $\frac{\partial}{\partial \square} \sum_{i=1}^N \log \mathcal{N} \overset{!}{=} 0$

$$\hat{\vec{\theta}} = \arg\max_{\vec{\theta}} p(\vec{\theta}|\vec{x}_1, \ldots, \vec{x}_N) = \ldots = \arg\max_{\vec{\theta}} \prod_{i=1}^N p(\vec{x}_i|\vec{\theta}) = \arg\max_{\vec{\theta}} \sum_{i=1}^N \log p(\vec{x}_i|\vec{\theta})$$

Maximum a-posteriori estimation (MAP)

- assumption 1: independency

$$\hat{\vec{\theta}} = \arg\max_{\vec{\theta}} p(\vec{\theta}|\vec{x}_1, \ldots, \vec{x}_N) = \ldots = \arg\max_{\vec{\theta}} p(\vec{\theta}) \cdot \prod_{i=1}^N p(\vec{x}_i|\vec{\theta})$$

$$= \arg\max_{\vec{\theta}} \log p(\vec{\theta}) + \sum_{i=1}^N \log p(\vec{x}_i|\vec{\theta})$$

3.2 Non-parametric Density Estimation

For a fixed #pixels and #intensities, there are #intensities$^{\text{\#pixels}}$ different images.

Histograms

- Piecewise constant approximation of a pdf
- K bins, bin j represents the region R_j
- Approximate probability of samples falling in R_j by relative frequencies: $P_{R_j} = \frac{k_j}{N}$

Problems

1. How many bins? (K)
2. Can we compute (the size of) R_j in an optimal and non-uniform manner?
3. What do we do with sparse estimates?

Parzen Window Estimation

- Assumption (1): PDF $p(\vec{x})$ is constant inside the regions R_j:

$$P_R = \int_R p(\vec{x})\, \mathrm{d}\vec{x} \overset{(1)}{=} \hat{p}(\vec{x}) \int_R \mathrm{d}\vec{x} = \underbrace{\hat{p}(\vec{x}) \cdot V_R \qquad \frac{k_R}{N}}_{\hat{p}(\vec{x}) \overset{(1)}{=} \frac{k_R}{N \cdot V_R}} = P_R$$

- R is a d-dim. hyper cube, edge length $h \Rightarrow V_R = h^d$

$$K_\lambda(\vec{x}_i, \vec{x}) = \begin{cases} \frac{1}{h^d} & ||\vec{x}_i - \vec{x}||_\infty \leq \frac{h}{2} \\ 0 & \text{o.w.} \end{cases} \qquad \lambda = h \qquad \text{[DIFF] } (h^{-d} \text{ inside kernel})$$

$$K_\lambda(\vec{x}_i, \vec{x}) = \frac{1}{\sqrt{|2\pi\Sigma|}} e^{-\frac{1}{2}(\vec{x}_i - \vec{x})^T \Sigma^{-1} (\vec{x}_i - \vec{x})} \qquad \lambda = \Sigma$$

$$\hat{p}(\vec{x}) = \frac{1}{N} \sum_{i=1}^N K_\lambda(\vec{x}_i, \vec{x})$$

Platziere den Kernel um jedes $\vec{x}_i$, summiere an der Stelle $\vec{x}$ auf.

- Convolution and Parzen Kernels [DIFF]
  - K_λ needs to depend on the difference of $\vec{x}_i$ and $\vec{x}$

  (1) When integrating over all possible $\vec{x}_i$, each single $\vec{x}_i$ occurs with probability $p(\vec{x}_i)$, as the samples are drawn from the exact distribution.

  (2) Definition of convolution

  - The convolution shows that $\hat{p}(\vec{x})$, with sufficent samples, is a low-pass filtered version of the exact density $p(\vec{x})$.

$$\lim_{N\to\infty} \hat{p}(\vec{x}) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^N K_\lambda(\vec{x}_i - \vec{x}) \overset{(1)}{=} \int K_\lambda(\vec{x}_i - \vec{x}) p(\vec{x}_i)\, \mathrm{d}\vec{x}_i \,(= E[p(\vec{x})]) \overset{(2)}{=} K_\lambda(\vec{x}) * p(\vec{x})$$

- Training: Parameter estimation with Leave-one-out cross validation
  - For each test "set" $\{\vec{x}_j\}$
  - Do a parzen estimate with the remaining $N-1$ training samples

$$P^j_{\lambda,N-1}(\vec{x}) = \frac{1}{N-1} \sum_{\substack{i=1 \\ i \neq j}} K_\lambda(\vec{x}_i, \vec{x}_j)$$

  - Optimize λ such that the probability of drawing the $\vec{x}_j$s out of the estimated pdfs is maximized (indep. samples (MLE) $\Rightarrow$ maximize the product)

$$\hat{\lambda} = \arg\max_\lambda \mathcal{L}(\lambda) = \arg\max_\lambda \prod_{j=1}^N P^j_{\lambda,N-1}(\vec{x}_j) = \arg\max_\lambda \sum_{j=1}^N \log P^j_{\lambda,N-1}(\vec{x}_j)$$

  - cont. optimization problem $\Rightarrow$ compute zero crossings of gradient, $\mathcal{O}(n^2)$!
  - Estimation of specific λ_i for each sample possible, drawbacks (high dim.)

Discretization of the Parzen Window Estimator with Histograms

- How many bins?
- Hot to compute adaptive bin sizes?

[DIFF] renamed b to K, $h_N(\vec{x}_i)$ to $h_N(j)$, x_i to μ_j (it is a *bin center*, NOT a sample)

- K: #bins
- $h_N(j)$: relative frequency / fraction of samples that fall into bin j.
- $\vec{\mu}_j$: center of bin j.
- Caution: This is a sum over the bins / the feature space, not over samples

$$\hat{P}_{\lambda,N}(\vec{x}) = \sum_{j=1}^{K} h_N(j) \cdot K_\lambda(\vec{x} - \vec{\mu}_j)$$

- For infinitesimal small bins, this converges to the parzen estimate: $\vec{\mu}_j = \vec{x}_i$ because the "bin" only contains a single value, $h_N(j)$ becomes the probability of that single value ($p(\vec{x}_i)$):

$$\int p(\vec{x}_i) \cdot K_\lambda(\vec{x}_i - \vec{x}) \, \mathrm{d}\vec{x}_i = K_\lambda(\vec{x}) * p(\vec{x}) = \hat{p}(\vec{x})$$

Objective function for minimizing the approximation error of the histogram
- **1d case!**, use parzen window estimation to estimate $p(x)$

$$J = \sum_{j=1}^{K} \int_{l_{j-1}}^{l_j} (x - \mu_j)^2 \cdot p(x) \, \mathrm{d}x$$

- set derivatives to zero
- for $\frac{\partial}{\partial \mu_j}$: integration is linear $\Rightarrow$ pull the derivation into the integral
- for $\frac{\partial}{\partial l_j}$:
  - derivative of integration borders: put them into the antiderivative (Stammfunktion) and derive these, which is zero for $F(l_{j\pm1})$
  - use $p(l_j) > 0$, $l_0 = \min$, $l_K = \max$, different signs in the 2 resulting cases.
- HW: if $p(x)$ is the uniform distribution, this approach chooses $l_j^{(i+1)}$ as the middle of the middles of l_{j-1}, l_j and l_j, l_{j+1}. Thereby, the bin sizes are averaged. Note: For a uniform distribution, the actual bin sizes are completely irrelephant :-)

3.3 Histogram Layout by Regression Trees

[DIFF] rename c_j to h_j as it is just the relative frequency of the prev. chapters

$$\hat{p}(\vec{x}) = \sum_{j=1}^{K} h_j I(\vec{x} \in R_j) \qquad (I \text{ just selects } h_j \text{ of the correct bin})$$

Problems

- Estimate the number of bins (K)
- Compute the K regions (R_j)
- Estimate probability for each bin (h_j)

$$\hat{h}_j = \arg\min_{h_j} \int_{R_j} (p(\vec{x}) - \hat{p}(\vec{x})) \, \mathrm{d}\vec{x} = \arg\min_{h_j} \int_{R_j} (p(\vec{x}) - h_j) \, \mathrm{d}\vec{x}$$

Optimization problem

$$\min_{k,s} \left\{ \min_{h_1} \sum_{\vec{x} \in R_1(k,s)} (p(\vec{x}) - h_1)^2 + \min_{c_2} \sum_{\vec{x} \in R_2(k,s)} (p(\vec{x}) - c_2)^2 \right\} \quad \text{where } \begin{array}{l} R_1(k,s) = \{\vec{x} | x_k \leq s\} \\ R_1(k,s) = \{\vec{x} | x_k > s\} \end{array}$$

1. Stop splits if number of required bins (pre defined "engineering constant") is reached
2. Apply a regularizer that penalizes a large number K of bins: $+\alpha K$

4 Mean Shift Algorithm

Set derivative of pdf to zero until convergence (at the maximum)

$$p(\vec{x}) = \frac{1}{N} \sum_{i=1}^{N} K_\lambda(\vec{x}_i, \vec{x})$$

$$\nabla p(\vec{x}) = \frac{1}{N} \sum_{i=1}^{N} \nabla K_\lambda(\vec{x}_i, \vec{x}) \overset{!}{=} 0$$

$$K_E(\vec{x}, \vec{x}_i) = \begin{cases} c \cdot (1 - ||\vec{x} - \vec{x}_i||_2^2) & ||\vec{x} - \vec{x}_i||_2^2 \leq 1 \\ 0 & \text{o.w.} \end{cases} \qquad \begin{array}{l} \text{Epanechnikov kernel} \\ \text{leads to mean of sphere} \end{array}$$

Don't miss the conditional when deriving the Epanechnikov kernel!
Hill climber $\rightarrow$ local optima, depends on initialization
The gradient and therefore the step size gets smaller near the optimum
Use adaptive window size / kNN, or sparse regions could cause problems
Edge Preserving Smoothing:
- Use $(x, y, L, u, v)^T \in \mathbb{R}^5$, replace the color part by that of the found maximum
- Intuition: Meanshift walks along the pdf of *colors* to maxima
- Edges are preserved, because at edges, the near gradients of the color space point *away from the edge*, therefore the mean shift procedure walks away from the edge, the edge is preserved and forms the cluster boundary (colors on the other side of the edge are not even taken into account for gradient computation, as they do not fall into the kernel-sphere due to the higher distance in the color components)

- the x and y coordinate assures, that for this estimation process, only *local pixels* are evaluated, so the color pdf is not distracted by other parts of the image (where the edge color could have a high probability and therefore would distract the gradient)

5 Manifold Learning

5.1 Curse of Dimensionality

"Human intuition breaks down in high dimensional spaces."

- uniformly distributed samples in a volume, capture fraction r of samples
- required volume: $r \cdot V$
- For a cubic fraction inside a unit cube: $e^d = r \cdot 1 \Rightarrow e = r^{\frac{1}{d}}$
- Let N samples be uniformly distributed within a sphere ($||\vec{x}_i|| \leq 1$)
  The expected *median* distance of the nearest neighbor to the origin is

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

  $\Rightarrow$ Most of the data points rae close to the boundary

- Exponential explosion of samples: To achieve the same density of samples as with N samples in $d = 1$, N^d samples are necessary

5.2 Principal Component Analysis (PCA) [DIFF]

- $\vec{x}_i$ are assumed to be zero mean, if not, replace with $\vec{x}_i - \mu$
- Objective function with Φ is wrong (result is a constant, constrain does not enforce $||\vec{n}||_2 = 1$, orthogonality constraints for 2nd, 3rd, ... components missing)

Objective function for the first PC:

$$\arg\max_{\vec{n}} = \sum_{i=1}^{N} \underbrace{(\vec{n}^T \vec{x}_i)^2}_{\substack{\text{as } \vec{n} \text{ is unit length,} \\ \text{this is the length of} \\ \vec{x}_i\text{s projection}}} + \lambda(1 - ||\vec{n}||_2^2)$$

$$\text{Note: } \sum_{i=1}^{N} ((\vec{n}^T \vec{x}_i)\vec{n})^T (\vec{n}^T \vec{x}_i)\vec{n} = \sum_{i=1}^{N} (\vec{n}^T \vec{x}_i)^2 \underbrace{(\vec{n}^T \vec{n})}_{1} = \vec{n}^T (\sum_{i=1}^{N} \vec{x}_i \vec{x}_i^T) \vec{n} = \vec{n}^T \Sigma \vec{n}$$

5.3 Multidimensional Scaling (MDS)

- **linear** - projections in lower dimensions very similar to PCA (with its problems)!
- $\vec{x}_i$ are assumed to be zero mean (1), if not, replace with $\vec{x}_i - \mu$
- Problem: Reconstruct a set of points out of differences
- Differences don't include translation, rotation and reflection
- Sample matrix: $X = [\vec{x}_1, \dots, \vec{x}_N] \in \mathbb{R}^{d \times N}$
- Distance matrix: $D^2 = [d_{ij}^2] = [||\vec{x}_i - \vec{x}_j||_2^2]$
  $= \text{diag}(X^T X) \cdot \vec{1}^T + \vec{1} \cdot \text{diag}^T(X^T X) - 2X^T X$, where diag() is a column vector
- Centering matrix: $C = (I - \frac{1}{N}\vec{1} \cdot \vec{1}^T)$
- Formal Problem: Reconstruct X from D^2 by applying SVD on $-\frac{1}{2}CD^2C$
- (1) $X \cdot \vec{1} = \vec{0}$, B is a (2) symmetric, (3) positive matrix, (4) Σ is symmetric

$$-\frac{1}{2}CD^2C \overset{(1)}{=} X^T X =: B \quad B \overset{(2)}{=} U^T \Sigma U \overset{(3)}{=} U^T \Sigma^{\frac{1}{2}} \Sigma^{\frac{1}{2}} U \overset{(4)}{=} (\Sigma^{\frac{1}{2}} U)^T \underbrace{(\Sigma^{\frac{1}{2}} U)}_{=:X} = X^T X$$

5.4 Sammon Transform

$$d_{ij} = ||\vec{x}_i - \vec{x}_j||_2^2 \qquad d'_{ij} = ||\vec{x}'_i - \vec{x}'_j||_2^2$$

$$\min \frac{1}{\sum_{\substack{i,j \\ i<j}} d'_{ij}} \sum_{\substack{i,j \\ i<j}} \frac{(d'_{ij} - d_{ij})^2}{d'_{ij}}$$

- "Optimization w.r.t. $\vec{x}'_i$ is awkward."

5.5 Local(ly) Linear Embedding (LLE)

1. Define a local neighborhood (sphere or kNN)
2. Find w_{ij}:

$$\min_W \sum_{i=1}^{N} ||\vec{x}_i - \sum_{j=1}^{N} w_{ij}\vec{x}_j||_2^2 \quad \text{s.t.} \quad \sum_{j=1}^{N} w_{ij} = 1 \text{ where } w_{ij} = 0 \text{ if } \vec{x}_j \notin \mathcal{N}(\vec{x}_i)$$

[DIFF] $\vec{x}_i = \sum_{j=1}^{N} w_{ij}\vec{x}_i$ as $\sum_{j=1}^{N} w_{ij} = 1$ (same result, i, j are just swapped, symmetric)

$$= \min_W \sum_{i=1}^{N} || \sum_{j=1}^{N} w_{ij}(\vec{x}_i - \vec{x}_j)||_2^2 = \min_W \sum_{i=1}^{N} ||M_i \vec{w}_i||_2^2 \text{ where } M_i = (\vec{x}_i - \vec{x}_1, \dots, \vec{x}_i - \vec{x}_N)$$

Each summand is independent of the others when optimizing $\vec{w}_i$.
The columns of M_i' are set to $\vec{0}$, if i and j are no neighbours.

$$\Rightarrow \forall i : \min_{\vec{w}_i} ||M_i'\vec{w}_i||_2^2 + \lambda(1 - ||\vec{w}||_2^2) = \min_{\vec{w}_i} \vec{w}_i^T(M_i'^T M_i')\vec{w}_i + \lambda(1 - \vec{w}^T\vec{w})$$

Apply SVD like for PCA, but as we minimize, take the eigenvectors that belong to the lowest eigenvalues.

3. Use w_{ij} to optimize $\vec{x}_i'$ in lower dim. space:

$$\min_{X'} \sum_{i=1}^N ||\vec{x}_i' - \sum_{j=1}^N w_{ij}\vec{x}_j'||_2^2 \text{ s.t. } \sum_{i=1}^N \vec{x}_i' = \vec{0}, \frac{1}{N}\sum_{i=1}^N \vec{x}_i'\vec{x}_i'^T = I$$

We (in the exercise) (and the paper) just proofed the **1D case**:

$$\sum_{i=1}^N (x_i' - \sum_{j=1}^N w_{ij}x_j')^2 = ||(I - W)\vec{x}'||_2^2$$
$$\Rightarrow \frac{\partial}{\partial \vec{x}'}\vec{x}'^T M\vec{x}' + \lambda(1 - \frac{1}{N}\vec{x}'^T\vec{x}') \overset{!}{=} 0$$

NOTES: If we don't use the zero-mean constrain, the smallest eigenvector is always a vector with all-equal entries corresponding to the eigenvalue zero. This eigenvector translates all samples equally, which is what we want to prevent with the constrain.

$$(I - W)\vec{1} = I\vec{1} - W\vec{1} = \vec{1} - \vec{1} = \vec{0} = \underbrace{0}_{=\lambda} \cdot \underbrace{\vec{1}}_{\vec{x}'}$$

The other constrain fixes the scaling (which shall be uniform and of order 1 in each dimension (1s in the diagonal), therefore the error is measured in the same scale) and rotation (to the solution where all axes are uncorrelated).

5.6 Isometric Feature Mapping (ISOMAP)

- Target: Preserve geodesic manifold interpoint distance (graphs)
- Problem 1: How to measure geodesic distances on the unknown manifold?
- Problem 2: How to map feature vectors on the Euclidean space in lower dim.?
- ISOMAP = Classical MDS with D based on geodesic distances

1. Construct a neighborhood graph (sphere or kNN):

$$d_{ij} = \begin{cases} ||\vec{x}_i - \vec{x}_j||_2 & \vec{x}_j \in \mathcal{N}(\vec{x}_i) \\ 0 & \text{o.w.} \end{cases}$$

2. Compute the shortest path between all pairs of points (Floyd Warshall) $\rightarrow D$
3. Perform MDS on D (SVD on $-\frac{1}{2}CDC$)

+ non-linear (in contrast to MDS distances esp. of non-neighbors are not preserved)
+ non-iterative polynomial algorithm
+ globally optimal
+ manifolds of arbitrary dimension

− very sensitive to noise

5.7 Laplacian Eigenmaps (LEM)

+ Noisy Data

1. Build the adjacency graph on $\mathcal{S}$ (guess what? sphere or kNN!)
2. Choose weights $w_{ij} \in [0, 1]$ for the edges in the graph. 2 examples:

$$w_{ij} = \begin{cases} e^{-\frac{1}{t}||\vec{x}_i - \vec{x}_j||_2^2} & \vec{x}_j \in \mathcal{N}(\vec{x}_i) \\ 0 & \text{o.w.} \end{cases}$$

$$w_{ij} = \begin{cases} 1 & \vec{x}_j \in \mathcal{N}(\vec{x}_i) \\ 0 & \text{o.w.} \end{cases}$$ (actually the upper weights with $t \rightarrow \infty$)

3. Eigendecomposition of the graph's Laplacian
(There are special routines for $L\vec{v} = \lambda D\vec{v}$ decompositions.)
4. Low dimensional embedding (Each eigenvector (starting at the (second) lowest eigenvalue) contains one dimension of the projected samples $\vec{x}'$.)

1D Case:

$$\min_X \sum_{i=1}^N \sum_{j=1}^N (x_i' - x_j')^2 w_{ij} \quad \text{s.t. } \vec{x}'^T D\vec{x}' = 1 \quad (\vec{x}' D\vec{1} = 0 \text{ removes transl. invar.})$$

$$\vec{x}' = \begin{pmatrix} x_1' \\ \vdots \\ x_N' \end{pmatrix} \qquad D = \begin{pmatrix} \sum_{i=1}^N w_{1i} & & 0 \\ & \ddots & \\ 0 & & \sum_{i=1}^N w_{Ni} \end{pmatrix} \qquad W = [w_{ij}] \qquad L = D - W$$

(Here, D is not a distance matrix, but a $\underline{d}$iagonal matrix!)

$$= \min_X 2\vec{x}'^T L\vec{x}' \quad \text{s.t. } \vec{x}'^T D\vec{x}' = 1 \qquad \Rightarrow \qquad D^{-1}L\vec{x}' = \lambda\vec{x}'$$

[DIFF] Without the additional translation invariance constrain, again remove the eigenvector belonging to the eigenvalue zero. Proof:

$$D^{-1}(D - W)\vec{1} = \lambda\vec{1} \iff D^{-1}(D\vec{1} - W\vec{1}) = \lambda\vec{1} \iff D^{-1}\vec{0} = \lambda\vec{1} \iff \lambda = 0$$

6 Hidden Markov Models (HMMs)

Checklist

- Marginal: Dynamic programming (sums)
- Most probable state sequence: Viterbi (max)
- Training: EM (fancy)

6.1 Orders of dependency

$$p(\langle \vec{x}_1, \dots, \vec{x}_n \rangle | y) = \prod_{i=1}^{n} p(\vec{x}_i | y) \qquad \text{(stat. independency)}$$

$$p(\langle \vec{x}_1, \dots, \vec{x}_n \rangle | y) = p(\vec{x}_1 | y) \prod_{i=2}^{n} p(\vec{x}_i | \vec{x}_{i-1}, y) \qquad \text{(1st order model)}$$

$$p(\langle \vec{x}_1, \dots, \vec{x}_n \rangle | y) = p(\vec{x}_1 | y) p(\vec{x}_2 | \vec{x}_1, y) \prod_{i=2}^{n} p(\vec{x}_i | \vec{x}_{i-2}, \vec{x}_{i-1}, y) \qquad \text{(2nd order model)}$$

6.2 Mistures / Gaussian Mixture Models (GMMs)

- GMM = HMM mit $\forall S_i, S_j : \pi_{S_j} = a_{S_i S_j}$

$$p(\langle \vec{x}_1, \dots \vec{x}_n \rangle | \langle s_1, \dots, s_n \rangle) = \prod_{i=1}^{n} p(\vec{x}_i | S_i)$$

$$p(\langle \vec{x}_1, \dots \vec{x}_n \rangle, \langle s_1, \dots, s_n \rangle) = \prod_{i=1}^{n} \pi_{s_i} p(\vec{x}_i | s_i)$$

$$p(\langle \vec{x}_1, \dots \vec{x}_n \rangle) = \underbrace{\sum_{s_1, \dots, s_n}}_{\in \mathcal{O}(m^n)} p(\langle \vec{x}_1, \dots \vec{x}_n \rangle, \langle s_1, \dots, s_n \rangle) = \underbrace{\prod_{i=1}^{n} \sum_{s_i}}_{\in \mathcal{O}(n \cdot m)} \pi_{s_i} \cdot p(\vec{x}_i | s_i)$$

6.3 Hiden Markov Models (HMMs)

- When drawing HMMs, don't miss the self loops $a_{s_i s_i}$!
- If feature sequence and state sequence are known
- the first = is correct, as we don't start with the posterior but the joint density!

$$p(\langle \vec{x}_1, \dots \vec{x}_n \rangle, \langle s_1, \dots, s_n \rangle) = p(\langle s_1, \dots, s_n \rangle) \cdot p(\langle \vec{x}_1, \dots \vec{x}_n \rangle | \langle s_1, \dots, s_n \rangle)$$
$$= \pi_{s_1} \prod_{i=2}^{n} a_{s_{i-1} s_i} \cdot \prod_{i=1}^{n} p(\vec{x}_i | s_i)$$

$$p(\langle \vec{x}_1, \dots \vec{x}_n \rangle) = \sum_{s_1, \dots s_n} \pi_{s_1} \prod_{i=2}^{n} a_{s_{i-1} s_i} \cdot \prod_{i=1}^{n} p(\vec{x}_i | s_i) \qquad \text{(marginalization over the states)}$$

$$= \sum_{s_1} \pi_{s_1} p(\vec{x}_1 | s_1) \cdot \left(\dots \left[\sum_{s_{n-1}} a_{s_{n-2} s_{n-1}} p(\vec{x}_{n-1} | s_{n-1}) \cdot \left\{ \sum_{s_n} a_{s_{n-1} s_n} p(\vec{x}_n | s_n) \right\} \right] \dots \right)$$

dynamic programming $\curvearrowright$ time $\in \mathcal{O}(n \cdot m^2)$, space $\in \mathcal{O}(m)$

Computation of the most probable state sequence / VITERBI algorithm

- Reminder: $y^* = \arg\max_y p(y | \vec{x}) = \arg\max_y \frac{p(\vec{x}, y)}{\cancel{p(\vec{x})}}$
- Instead of marginalizing over the states, we maximize

$$\langle \hat{s}_1, \dots, \hat{s}_n \rangle = \arg\max_{\langle s_1, \dots, s_n \rangle} p(\langle \vec{x}_1, \dots \vec{x}_n \rangle, \langle s_1, \dots, s_n \rangle)$$
$$= \arg\max_{s_1, \dots s_n} \pi_{s_1} \prod_{i=2}^{n} a_{s_{i-1} s_i} \cdot \prod_{i=1}^{n} p(\vec{x}_i | s_i)$$

$$\curvearrowright \max_{s_1} \pi_{s_1} p(\vec{x}_1 | s_1) \cdot \left(\dots \left[\max_{s_{n-1}} a_{s_{n-2} s_{n-1}} p(\vec{x}_{n-1} | s_{n-1}) \cdot \left\{ \max_{s_n} a_{s_{n-1} s_n} p(\vec{x}_n | s_n) \right\} \right] \dots \right)$$

dynamic programming $\curvearrowright$ time $\in \mathcal{O}(n \cdot m^2)$, space $\in \mathcal{O}(m)$

6.4 HMM training

- $\lambda = (\vec{\pi}, A, B)$
- π: input probabilities
- $A = [a_{S_i S_j}]$: transition probabilities
- B: output pdfs (one for each state)
- NOTE: Simple alternative: Viterbi training
  - Do some model initialization, then repeat:
  - Compute most probable state sequence with Viterbi
  - Now we can estimate π_{s_i}, $a_{s_i s_j}$ and $p(\vec{x}_i | s_i)$ using relative frequencies (engineering: add one to each probability)
- Reminder EM:
  - X: observable random variable, $\langle s_1, \dots, s_n \rangle$
  - Y: hidden random variable, $\langle \vec{x}_1, \dots, \vec{x}_n \rangle$

$$p(X) = \frac{p(X, Y)}{p(Y|X)}$$

$$\underbrace{-\log p(X)}_{\text{observable info.}} = \underbrace{-\log p(X, Y)}_{\text{complete info.}} - \underbrace{(-\log p(Y|X))}_{\text{hidden info.}}$$

Assumption: parametric pdfs, multiply $p(Y|X, \lambda^{(i)})$ (index!) and integrate over Y

$$LS = -\log p(X|\lambda^{(i+1)}) = -\int p(Y|X, \lambda^{(i)}) \log p(X|\lambda^{(i+1)})\, \mathrm{d}Y = LS \cdot 1$$

$$RS = \underbrace{-\int p(Y|X, \lambda^{(i)}) \cdot \log p(X, Y|\lambda^{(i+1)})\, \mathrm{d}Y}_{Q}$$

$$- \underbrace{(-\int p(Y|X, \lambda^{(i)}) \cdot \log p(Y|X, \lambda^{(i+1)}))}_{H}$$

$$Q(\lambda^{(i)}, \lambda^{(i+1)}) = \int p(Y|X, \lambda^{(i)}) \cdot \log p(X, Y|\lambda^{(i+1)})\, \mathrm{d}Y$$
$$(= E[\log p(X, Y|\lambda^{(i+1)})|X, \lambda^{(i)}])$$

1. Step: $p(Y|X, \lambda^{(i)})$

$$p(Y|X, \lambda^{(i)}) = \frac{p(X, Y|\lambda^{(i)})}{p(X|\lambda^{(i)})} = \frac{p(X, Y|\lambda^{(i)})}{\int p(X, Y|\lambda^{(i)})\, \mathrm{d}Y}$$

2. Step (<u>Expectation</u>): $Q(\lambda^{(i)}, \lambda^{(i+1)})$ (integrals $\rightarrow$ sums over $s_1, \dots, s_n$, discrete)

$$Q(\lambda^{(i)}, \lambda^{(i+1)}) = \int \underbrace{p(Y|X, \lambda^{(i)})}_{\text{independent of } \lambda^{(i+1)}!} \cdot \log p(X, Y)\lambda^{(i+1)}\, \mathrm{d}Y$$

3. Step: $\log p(X, Y|\lambda^{(i+1)}$

$$\log p(X, Y|\lambda^{(i+1)}) = \log \pi_{s_1}^{(i+1)} \cdot \prod_{i=2}^{n} a_{s_{i-1}s_i}^{(i+1)} \cdot \prod_{i=1}^{n} p^{(i+1)}(\vec{x}_i|s_i)$$
$$= \log \pi_{s_1}^{(i+1)} + \sum_{i=2}^{n} \log a_{s_{i-1}s_i}^{(i+1)} + \sum_{i=1}^{n} \log p^{(i+1)}(\vec{x}_i|s_i)$$

4. Step (<u>Maximization</u>):

DIFF: I think we need $\nabla_{\pi_{S_k}}$ (capital S). My notes are very chaotic / wrong from here on, so what follows is only similar to the lecture.

$$\nabla_{\lambda^{(i+1)}} Q(\lambda^{(i)}, \lambda^{(i+1)}) \overset{!}{=} 0 \quad \text{s.t.} \sum_{S_j} \pi_{S_j} = 1, \quad \forall S_j : \sum_{k=1}^{m} a_{S_j S_k} = 1, \quad \forall S_k : \int p(x|S_k)\, \mathrm{d}x = 1$$

$$\nabla_{\pi_{S_k}^{(i+1)}} \left[Q(\lambda^{(i)}, \lambda^{(i+1)}) + \mu(1 - \sum_{S_j} \pi_{S_j}) \right] = \underbrace{\sum_{s_2, \dots, s_n} p(\langle S_k, s_2, \dots, s_n \rangle | X, \lambda^{(i)})}_{=:a_k} \cdot \underbrace{\frac{1}{\pi_{S_k}^{(i+1)}}}_{=:p_k} - \mu = 0$$

$$a_k = \mu p_k \quad \Rightarrow \quad \sum_k a_k = \sum_k \mu p_k \quad \Leftrightarrow \quad \sum_k a_k = \mu \underbrace{\sum_k p_k}_{=1}$$

$$a_k = \sum_k a_k p_k \quad \Leftrightarrow \quad p_k = \frac{a_k}{\sum_{k'} a_{k'}}$$

Derivative for $a_{S_j S_k}$ and $p(\vec{x}|S_k)$: HW! ;-)
Spoiler:

$$\pi_{S_k}^{(i+1)} = \frac{p(\langle S_k, s_2, \dots, s_n \rangle | X, \lambda^{(i)})}{\sum_{s'_1, \dots, s'_n} p(\langle s'_1, s'_2, \dots, s'_n \rangle | X, \lambda^{(i)})} = \frac{p(\langle S_k, s_2, \dots, s_n \rangle | X, \lambda^{(i)})}{1}$$

$$a_{S_j S_k}^{(i+1)} = \frac{\sum_{i=2}^{n} p(\langle s_1, \dots, s_{i-2}, S_j, S_k, s_{i+1}, \dots, s_n \rangle | X, \lambda^{(i)})}{\sum_{i=2}^{n} p(\langle s_1, \dots, s_{i-2}, S_j, s_i, s_{i+1}, \dots, s_n \rangle | X, \lambda^{(i)})}$$

$$p^{(i+1)}(\vec{x}|S_k) = \frac{\sum_{i=2}^{n} p(\langle s_1, \dots, s_{i-1}, S_k, s_{i+1}, \dots, s_n \rangle | X, \lambda^{(i)}) \cdot I(\vec{x}_i = \vec{x})}{\sum_{i=2}^{n} p(\langle s_1, \dots, s_{i-1}, S_k, s_{i+1}, \dots, s_n \rangle | X, \lambda^{(i)})}$$

- What these formulas do is basically just computing relative frequencies with the training data, where the probability of being in a state S_k is estimated with the previous parameter set $\lambda^{(i)}$.
- These formulas only consider one training sequence, you will have to average between multiple ones, this is especially obvious for computing π_{S_k}s.
- In contrast to Viterbi training, we do not only take the most probable state assignment into consideration, but all of them.

6.5 Different Types of HMMs

1. discrete vs. continuous HMMs (depends on the pdfs $p(\vec{x}|s)$)
2. left-right-HMMs: $\forall j < i : a_{S_i S_j} = 0$
3. ergodic HMMs: $\forall i, j : a_{S_i S_j} > 0$
4. higher order HMM: n-th order: $a_{s_{i-(n-1)} \cdots s_{i-1} s_i}$

7 Markov Random Fields (MRFs) and Gibbs Random Fields (GRFs)

Definition of MRF

1. Positivity: $p(\vec{x}_1, \dots, \vec{x}_N) > 0$ (non-zero!)
2. Markov Property: $p(\vec{x}_k | \vec{x}_1, \dots, \vec{x}_N) = p(\vec{x}_k | \mathcal{N}(\vec{x}_k))$

- Neighborhoods $\mathcal{N}$ are <u>symmetric</u> and <u>irreflexive</u> ($x_i \notin \mathcal{N}(\vec{x}_i)$)
- A clique is a complete subgraph (edges between all pairs of nodes in the subgraph)
- Unsecure guess: The idea with the cliques is that we can define $p(\vec{x})$ as the product of all $p(x_i|c)$ where $c \subset S$ are all cliques containing the random variable x_i

Definition of GRF

- DIFF: Z is defined in a way that $\sum_x p(x)$ actually sums up to 1

$$p(x) = \frac{1}{Z} e^{-H(x)} \qquad \text{where } Z = \sum_{x'} e^{-H(x')}$$

- Z is called the *partition function*
(calculating it directly is almost always computationally prohibitive)
- $H(x)$ is called the *energy function*, set $H(x) = \sum_{m \subset S} V_m(x)$
- A *potential* is a family $\{V_m, m \subset S\}$, $V_\emptyset = 0$, $V_m(x) \overset{(1)}{=} V_m(y)$ if $x \cap m = y \cap m$
- $H(x)$ is not unique:

$$H(x) = -\log p(x) - \log Z$$
$$p(x) = \frac{1}{Z} e^{-H(x)} = \frac{1}{Z} e^{\log p(x) + \log Z} = \frac{\cancel{1}}{\cancel{Z}} e^{\log p(x)} e^{\log \cancel{Z}} = p(x)$$
$$\text{set } Z = e^0 = 1 \Rightarrow H(x) = -\log p(x) = \sum_{m \subset S} V_m(x)$$

Hammerslay Clifford Theorem

- MRF and GRF are equivalent

<u>GRF $\Rightarrow$ MRF</u>
$p(\vec{x})$ is GRF, $\vec{x}^k$ means all random variables except x_k, C is the set of cliques

DIFF: renamed some variables ($\vec{x}$ are the random variables, c are cliques)

$$p(x_k|\vec{x}^k) = \frac{p(\vec{x})}{p(\vec{x}^k)} = \frac{p(\vec{x})}{\sum_{x_k} p(\vec{x})} = \frac{\cancel{\frac{1}{Z}} \exp(-\sum_{c \in C} V_c(\vec{x}))}{\sum_{x_k} \cancel{\frac{1}{Z}} \exp(-\sum_{c \in C} V_c(\vec{x}))}$$
$$C_1 := \{c \subset C | x_k \in c\} \quad C_2 := \{c \subset C | x_k \notin c\}$$
$$= \frac{\exp(-\sum_{c \in C_1} V_c(\vec{x})) \cdot \exp(-\sum_{c \in C_2} V_c(\vec{x}))}{\sum_{x_k} [\exp(-\sum_{c \in C_1} V_c(\vec{x})) \cdot \exp(-\sum_{c \in C_2} V_c(\vec{x}))]}$$
$$\forall c \subset C_2 : x_k \notin c$$
$$= \frac{\exp(-\sum_{c \in C_1} V_c(\vec{x})) \cdot \cancel{\exp(-\sum_{c \in C_2} V_c(\vec{x}))}}{[\sum_{x_k} \exp(-\sum_{c \in C_1} V_c(\vec{x}))] \cdot \cancel{\exp(-\sum_{c \in C_2} V_c(\vec{x}))}}$$
$$= \frac{\exp(-\sum_{c \in C_1} V_c(\vec{x}))}{[\sum_{x_k} \exp(-\sum_{c \in C_1} V_c(\vec{x}))]}$$
$$\overset{(1)}{=} \frac{\exp(-\sum_{c \in C} V_c(\mathcal{N}(x_k)))}{\sum_{x_k} \exp(-\sum_{c \in C} V_c(\mathcal{N}(x_k)))} = p(x_k|\mathcal{N}(x_k))$$

Image smoothing

- DIFF: In the lecture, f (here: smoothed / output) and g (here: input) where swapped at some point
- The neighborhood for image computations is usually a 4 (cross) or 8 neighborhood.
- Assume independency between the pixels in the input image $g_i j$:

$$p([g_{ij}]) = \prod_{i,j} p(g_{ij})$$

- The pixel of the noisy image depends on its true value (which we want to restore):

$$p(g_{ij}|f_{ij})$$

This dependency becomes the $\mathcal{L}$ part of the MAP estimator

- Smooth images have a higher probability ($\rightarrow$ pixels depend on their neighbors)

$$p([f_{ij}]) = \prod_{i,j} p(f_{ij}|\mathcal{N}(f_{ij})) \overset{\log}{\rightarrow} \sum_{i,j} \log p(f_{ij}|\mathcal{N}(f_{ij}))$$

This dependency (over all cliques) will become the $\mathcal{R}$ part of the MAP estimator

$$
\begin{aligned}
[\hat{f}_{ij}] &= \underset{[f_{ij}]}{\arg\max}\, p([f_{ij}] | [g_{ij}]) \\
&\overset{(*)}{=} \underset{[f_{ij}]}{\arg\max}\, p([f_{ij}]) \cdot p([g_{ij}] | [f_{ij}]) \\
&= \underset{[f_{ij}]}{\arg\max} \overbrace{\underbrace{\log p([f_{ij}])}_{=:\mathcal{R}} + \underbrace{\log p([g_{ij}] | [f_{ij}])}_{=:\mathcal{L}}}^{=:-\mathcal{E} \quad \text{(Energy function)}} \\
&= \underset{[f_{ij}]}{\arg\max} \overbrace{\sum_{i,j} ||\nabla f_{ij}||_2^2} + \lambda \overbrace{\sum_{i,j} (f_{ij} - g_{ij})^2}
\end{aligned}
$$

- $(*)$ is just the bayesian rule in our context
- By choosing $\mathcal{R}$ as the gradient (which is then minimized), smooth images have a higher prior probability / are favored (why not the second order derivative?) With this simple model, the smoothing is not edge preserving.
- By choosing $\mathcal{L}$ to be the difference of the pixel value in the original and smoothed image, we favor images that are close to the original image
- λ defines just which one of our 2 optimization goals should be focused more