Pattern Analysis ("PA"): 5 ECTS Master's class on feature representations

Lecturer: Christian Riess, https://www1.cs1.tf.fau.de/christian-riess
Exercises: Lina Felsner, Dalia Rodriguez, Mathias Seuret

Communication Platform: StudOn

The class consists of 3 components:

(A) Pre-recorded lectures + literature sources (read, study independently!)

(B) Practical Python Exercises (play with data!)

(C) Joint meeting to "Realize Additional Connections" -> RAC[coon] (transfer & apply!)

(A) Lectures & Literature:
Update every week, you will receive an email notification.

The material consists of
- New videos
- Old videos
- Textbook literature sources
- Scientific papers

Textbooks:

(1) Bishop: "Pattern Recognition and Machine Learning", Springer, 2006
Download:
https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/

Important for the PDF download: check the Errata on Bishop's web page!

(2) Hastie, Tibshirani, Friedman: "The Elements of Statistical Learning", Springer, 2009.
Download:
https://web.stanford.edu/~hastie/ElemStatLearn/

Both textbooks are excellent standard works.

(B) Practical Python Exercises:
Purpose: test the mathematical theory on actual data

New programming tasks every 1-2 weeks
Join a studOn exercise group (up to 5 persons/group)
Each exercise group has its own etherpad to share code, also with supervisors

Use the studOn class forum to ask questions about the exercises!

Groups can also talk to supervisors in the exercise time slots, ask questions, show results
--> zoom meeting links will be posted in StudOn

Additional feedback loop:
post output to the solution thread in the studOn forum (one post/group)

Please feel totally free to communicate with your group partners via
other communication channels, like IRC, whatsapp, or land line!

(C) Joint Meeting to 'Realize Additional Connections' ("RACcoon session"):

Similar to a traditional blackboard exercise, but content is essentially driven by you

60 minutes zoom meeting (Meeting links will be posted in studOn)

During the regular lecture slots: Wednesday 14:15h, Thursday 16:15h

NO THURSDAY LECTURE IN FIRST WEEK!

Content is identical on Wednesday and Thursday - only attent one of them!

There will be a worksheet with questions that aim to deepen your understanding.
You use knowledge from (A) and (B) to address these questions.

You discuss the solution via chat, voice, camera - be prepared to actively participate!

Examination: Written exam of 60 minutes.
Sorry for not continuing the orals, but PA grows each semester, last year: ~150 PA orals.
Some sample questions will be provided to support exam preparation (at end of the summer term)

What will the first week look like?

Before Wednesday, April 22, 14:15h:
0. First videos & transcripts will be posted in StudOn
1. First exercise worksheet will be posted in StudOn
2. Exercise groups will be opened in StudOn
3. Zoom meeting links will be posted in StudOn

On Wednesday, April 22, 14:15h:
Plenary session - please all come to this meeting.
Content: zoom system check, Q&A, sample RACcoon task

NO THURSDAY LECTURE IN FIRST WEEK!

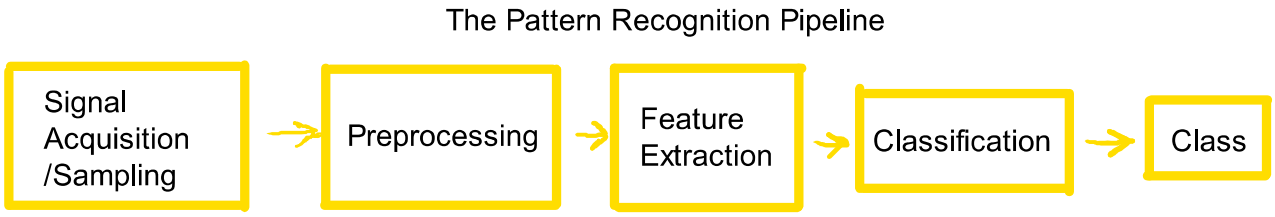
Wednesday, April 22, 16:00h - Friday, April 24
Python system check, get your first PA python code running

What will the weeks 2-14 look like?
We iterate lecture components (A), (B), (C)
If necessary, we will dynamically adjust the format as we go.

Questions? -> please post them to the studOn forum.

Most important remark in the end: Have a lot of fun!

...previously, in IntroPR or PR:



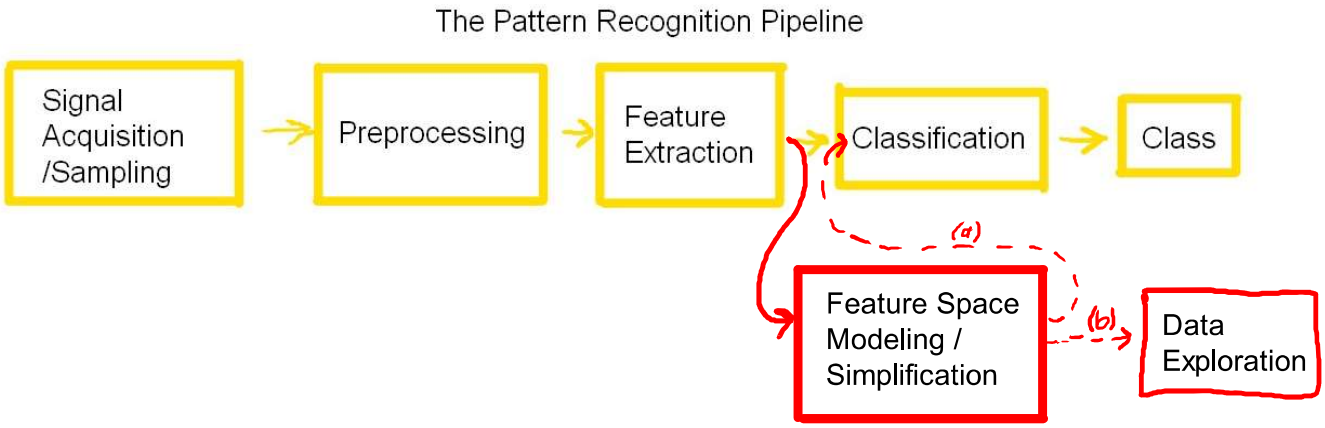
IntroPR is about "signal acquisition/sampling", "preprocessing", "feature extraction",
PR is mostly about "classification".

So where does PA fit in?

--> the PR pipeline is certainly a simplified scheme.
Many data processing tasks need a little bit more work than just these four steps to success

Long story short:
Pattern Analysis discusses methods to feature space modeling and simplification

Thus, we could introduce a processing step after feature extraction - I'll do this on the next slide



We can use our feature space analysis to improve classification.
This is visualized as path (a).

However, in real life there are also many more data analysis tasks than just classification.

For example, what if we are searching connections in Corona virus data. This is not a classification task, but rather an explorative task similar to data mining.

The tools and techniques from the Pattern Analysis lecture can also be used in that direction, visualized as path (b).

Pattern Analysis: Topics of this semester

1. Representations of the Feature Space
   (a) Kernel-based Representations
   (b) Dynamic Feature Space Partitions via Trees & Forests
2. Simplifications of the Feature Space
   (a) Hard and Soft Clustering
   (b) Dimensionality Reduction & Manifold Learning
3. Representations of Dependent Variables
   (a) Hidden Markov Models
   (b) Markov Random Fields
4. Spotlight on Model Selection

The topics of the lecture have A LOT of interconnections

We will "connect the dots" as we go, and in particular during our joint RACcoon sessions on Wednesdays and Thursdays.

Before we close this slide deck - let me ask one last question:

"How does PA relate to deep learning?!"

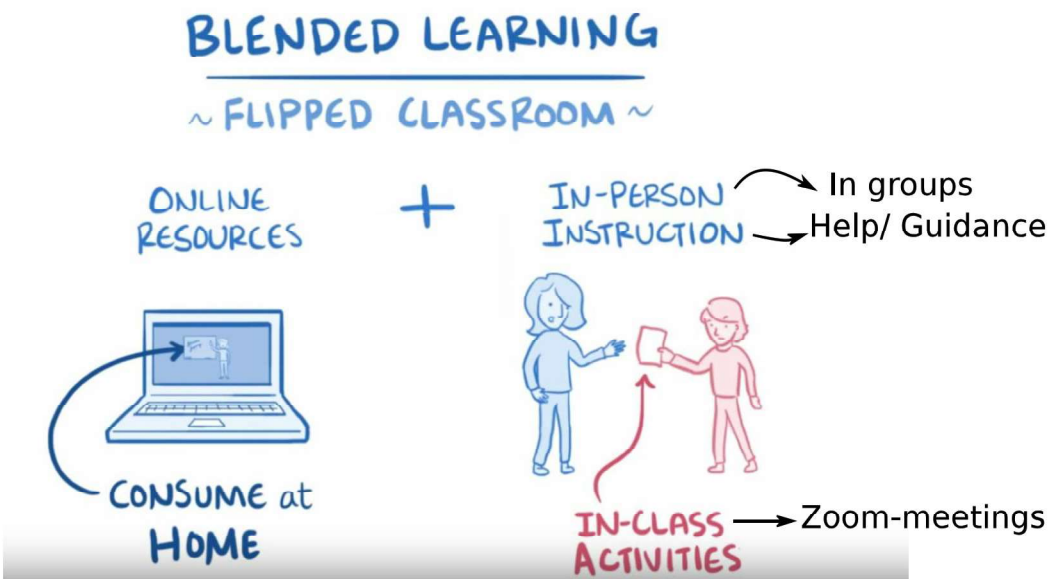
or in a more hostile phrasing:
"Who needs PA if everyone in the whole world is doing deep learning now?"

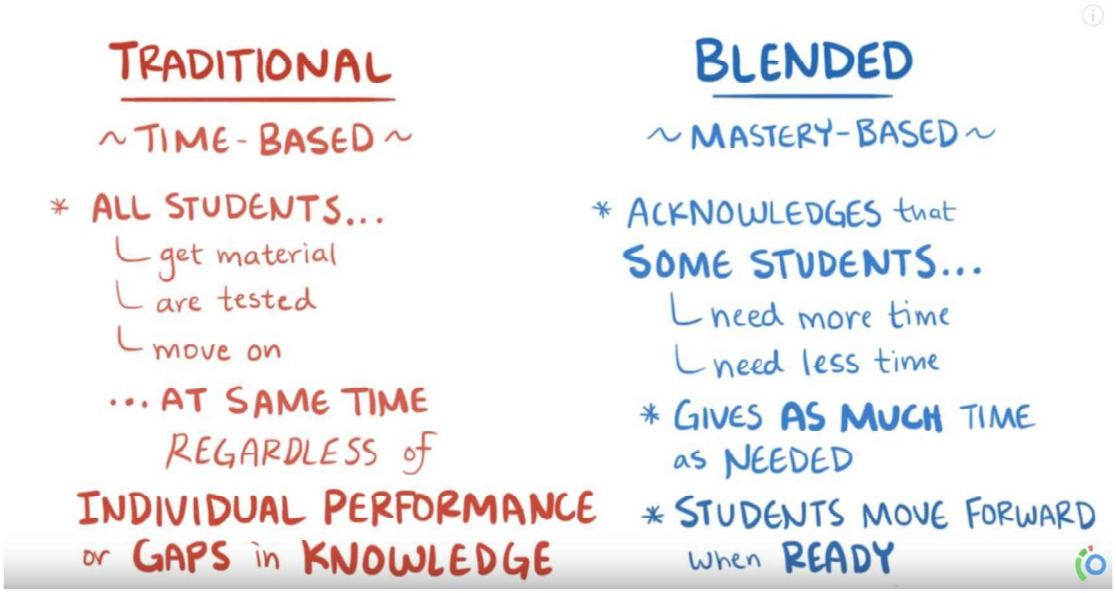
The answer is actually quite straightforward:
"Nothing changes with deep learning.
We need to operate on our feature representations today as much as we ever did before"

Example application:
Fine-tuning of deep neural network training via mining of hard training samples [1].
This technique operates with the tools from PA to realize path (a) on slide 2 in a fancy way.

[1] A. Iscen, G. Tolias, Y. Avrithis, O. Chum: "Mining on Manifolds: Metric Learning Without Labels", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 7642-7651.

These slides are copied and adapted from "Blended learning & flipped classroom"
The video is great, check it out: https://www.youtube.com/watch?v=paQCE58334M





*How to teach online?
*Communication between Teacher and Students?
*Focus and Understanding of "Blackboard-exercises"?

*Students work in groups to help each other.
*Groups gets help with exercise (if needed).
*Joint meetings in larger group.

ONLINE CONTENT

A: Lecture Material & Transcripts

* MOVE AT OWN PACE
  - ~ FAMILIAR → FASTER
  - ~ UNFAMILIAR
    - ↳ PAUSE
    - ↳ REWATCH
    - ↳ OTHER RESOURCES
* ENGAGE WHEN MOST ALERT
  - ~ EARLY or LATE
* TAKE BREAKS WHEN NEEDED

WORK ASSIGNMENTS ~~IN CLASS~~

* ACTIVE WORK > PASSIVE LISTENING
* TEACHERS AVAILABLE WHEN APPLYING KNOWLEDGE
* GROUPWORK
  - ~ LOGISTICALLY EASIER
  - ~ INSTRUCTOR can WATCH | HELP

Happy Student

WORK ASSIGNMENTS ~~in CLASS~~

* ACTIVE WORK > PASSIVE LISTENING
* TEACHERS AVAILABLE WHEN APPLYING KNOWLEDGE

B: Exercise Worksheets

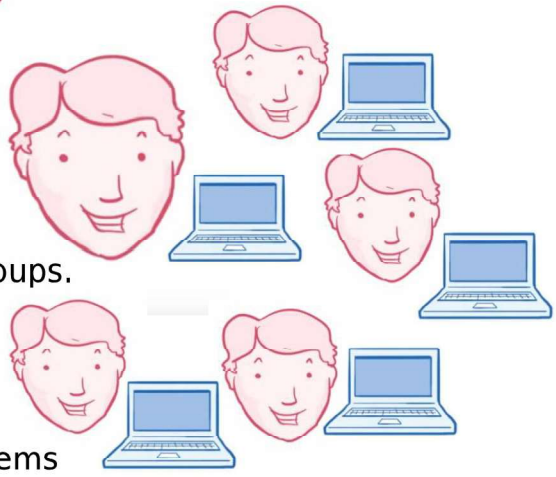
If you (as a group) need help with the exercises you can
- ask in the forum
- join a exercise meeting

C: RACcoon session

Joint meetings where we apply the knowledge and discuss
- learned content
- interesting questions

WORK ASSIGNMENTS ~~in CLASS~~

* ACTIVE WORK > PASSIVE LISTENING
* TEACHERS AVAILABLE WHEN APPLYING KNOWLEDGE
* GROUPWORK
  - ~ LOGISTICALLY EASIER
  - ~ INSTRUCTOR can WATCH | HELP

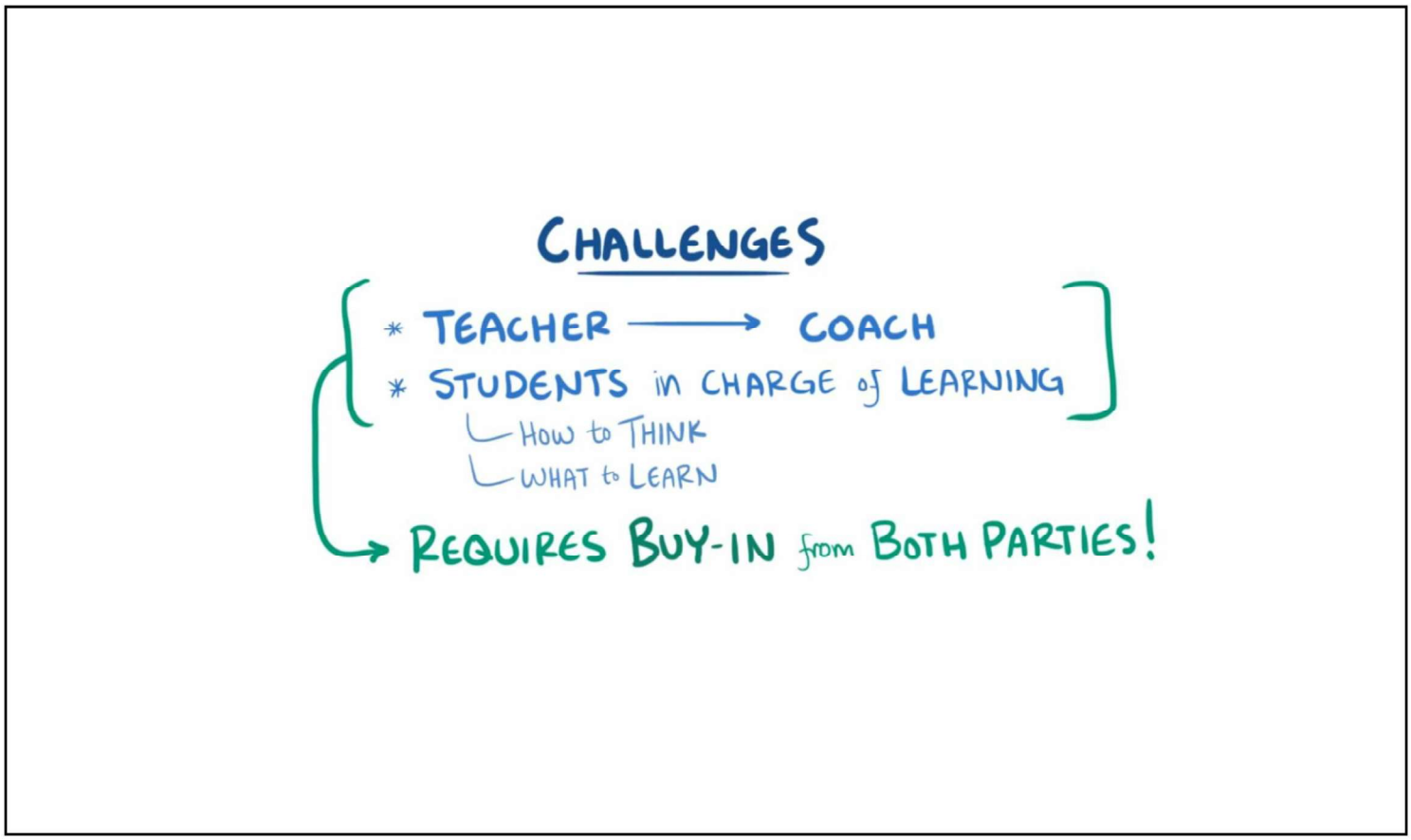


B: Exercise Worksheets

Solve the Exercises in Groups.
Help each other and share code if needed.

C: RACcoon session

Discussions/solving problems
1. in small groups (breakout rooms)
2. in large group


CHALLENGES
* TEACHER → COACH
* STUDENTS in CHARGE of LEARNING
└ How to THINK
└ WHAT to LEARN
→ REQUIRES BUY-IN from BOTH PARTIES!

Specifically on PA's joint meeting: How does it work?

Fictuous example schedule for Wednesday (times may vary, latest end after 90 minutes,
Thursday schedule analogously)

1) ~14:00h - 14:15h: Pre-meeting admission to the classroom.
2) ~14:15h - 14:25h: Welcome, greetings, general questions & remarks
3) 14:25h - 14:55h: Task 1 - "Think-Pair-Share"

~5 minutes: introduction to problem statement, task link to overleaf distributed
-> random assignment to breakout rooms in small groups

~15 minutes: "think & pair" without supervisors
research solution, exchange in small group

-> return to plenary session
~10 minutes: hosted discussion to collect & discuss solutions
need 1 student host for the discussion
(ad hoc determined, needs microphone)
need 1 student writer for solution proposals
(ad hoc determined, shares screen while writing down
solution proposals, e.g., in gimp, overleaf, or google docs)
need N-2 students who throw in solution hypotheses

4) 14:55h - 15:15h: Task 2 - General Discussion
-> same as "Think-Pair-Share", but we directly stay within the plenary session
5) 15:15h - 15:25h: Outlook on the upcoming week and closing
6) Meeting is closed, people disconnect. C. Riess will stay a few minutes for individual questions

Probabilities and Probability Density Functions (PDFs)

Bishop Sec. 1.2 through Sec. 1.2.3 (page 12-24):
very nice introduction to probabilities and probability density functions
Make sure that you are familiar with the terms and equations in these Sections!

Double-check your professional vocabulary:
- "joint distribution",
- "conditional distribution"
- "probability of x given y"
- "marginalization"
- "posterior"
- "likelihood"
- "prior"

Double-check your elementary probability equations:

Sum rule: $p(X) = \sum_Y p(X,Y)$ (1.10)

Product rule: $p(X,Y) = p(Y|X) \cdot p(X)$ (1.11)

Bayes theorem: $p(Y|X) = \frac{p(X|Y) \cdot p(Y)}{p(X)}$ (1.12)

Probability Density Function (PDF) $p(\vec{x})$

A propability density function satisfies two conditions:

Non-negativity: $p(\vec{x}) \geq 0$ (1.25)

Unit area: $\int_{-\infty}^{+\infty} p(\vec{x}) \, d\vec{x} = 1$ (1.26)

These equations remain valid when $\vec{x}$ is univariate or multivariate (i.e., a scalar or a vector).

Practical work with PDFs:

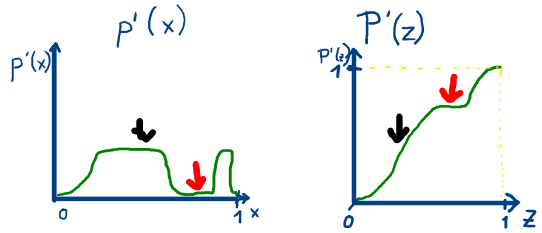
Assume you are given a PDF $p(\vec{x})$

Assume also that you use a programming language that only allows to draw random numbers from a uniform distribution

Question:
How can you computationally generate (draw) samples from that PDF?

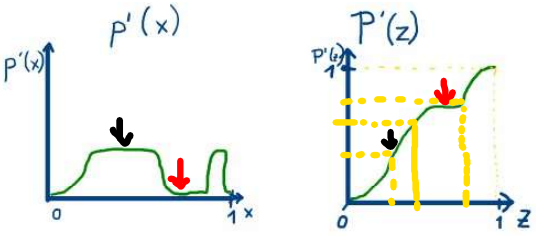
Algorithm for sampling from an (almost) arbitrary PDF $p(\vec{x})$

1. Discretize the domain of $p(\vec{x})$, normalize it between 0 and 1
2. If $\vec{x}$ is multivariate, linearize it in arbitrary order. Call the resulting density $p'(x)$
3. Calculate the cumulative density function $P'(z) = \int_{-\infty}^{z} p'(x)\,dx$
   (confer Bishop Eqn. (1.28) if necessary)



4. Draw a uniformly distributed number u, and find $z^* = \underset{z}{\operatorname{argmin}}\ u > P'(z)$

Copy of our sketch:



--> why does the algorithm work?
--> why does the caption say "almost arbitrary"?

Quick Recap: Parametric Density Estimation

Parametric estimator: determine best fitting function parameters for PDF.

Example: Estimate Gaussian mean & std.-dev. with a Maximum Likelihood (ML) estimator

$$\vec{\vartheta}^* = \underset{\vec{\vartheta}}{\operatorname{argmax}}\ p(\vec{x}_1, \ldots, \vec{x}_N | \vec{\vartheta})$$

or Maximum A Posteriori (MAP) estimator

$$\vec{\vartheta}^* = \underset{\vec{\vartheta}}{\operatorname{argmax}}\ p(\vec{\vartheta} | \vec{x}_1, \ldots, \vec{x}_N) \overset{\text{Bayes}}{=} \frac{p(\vec{x}_1, \ldots, \vec{x}_N | \vec{\vartheta}) \cdot p(\vec{\vartheta})}{p(\vec{x}_1, \ldots, \vec{x}_N)}$$

--> see PR Lecture, or e.g. Bishop Eqns. (1.55) and (1.59) for ML est.

Limitation: choice of function??
Bishop Sec. 2.1 - 2.4 offers plenty of functions, but still...?

Non-Parametric Density Estimation

Estimator without a closed-form distribution function.

Here:

(0. Histograms) <- Intro of Bishop Sec. 2.5
1. Parzen window estimator
2. k-Nearest Neighbor estimator

Literature source for the whole section: Bishop Sec. 2.5

Feature Space Subdivision

Let $p(\vec{x})$ be a PDF in D-dim. space, and x is contained in a small region R.

The probability mass in R is $P = \int_R p(\vec{x})\, d\vec{x}$

Assumption 1: large number of points --> P is relative frequency

$P = \frac{K}{N}$ ← # points in R; N ← total # points

Assumption 2: R is so small that $p(\vec{x})$ is approximately constant:

$$P = \int_R p(\vec{x})\, d\vec{x} = p(\vec{x}) \int_R dx = p(\vec{x}) \cdot V$$

V: Volume of R

Both (somewhat contradictory!) assumptions together give

$$p(\vec{x}) = \frac{K}{N \cdot V}$$

Parzen window estimator: variable $\frac{K}{N}$

k-Nearest Neighbors: variable V

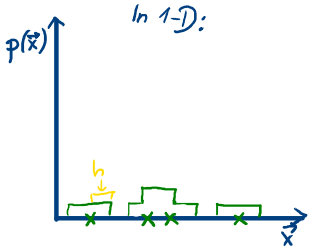
Parzen Window Estimator

"Parzen window": kernel function $k(\vec{u}) = \begin{cases} 1, & \text{if } |u_i| \leq \frac{1}{2}, \; i = 1 \ldots D \\ 0, & \text{otherwise} \end{cases}$ (D: dimension of the feature vector) (<-- a.k.a. "box kernel")

Calculate K with this kernel function: $K = \sum_{n=1}^{N} k\left(\frac{\vec{x} - \vec{x}_n}{h}\right)$ (h: size of the window)

Whole density: $p(\vec{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} k\left(\frac{\vec{x} - \vec{x}_n}{h}\right)$

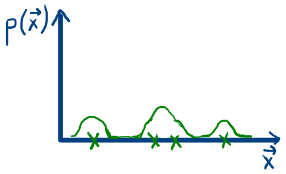
unfortunately not much greater than a plain histogram!



Oftentimes more interesting: a slightly smoothing kernel like the Gaussian kernel

$$p(\vec{x}) = \frac{1}{N} \cdot \sum_{n=1}^{N} \frac{1}{(2\pi h^2)^{D/2}} \cdot e^{-\left(\frac{\|\vec{x} - \vec{x}_n\|_2^2}{2h^2}\right)}$$

(h: standard deviation)



From a mathematical perspective, any kernel function is possible if these conditions hold:

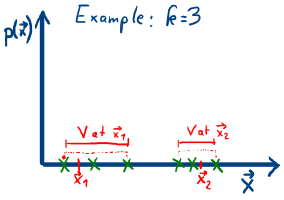
$$k(\vec{u}) \geq 0$$
$$\int k(\vec{u}) \, d\vec{u} = 1$$

k-Nearest Neighbors

Recall our derived density relationship

$$p(\vec{x}) = \frac{K}{N \cdot V}$$

We can also fix K and vary V ---> "k-nearest neighbors": grow V until k nearest points are found



First Glance at the Model Selection Problem

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

---> consider the kernel parameters (in our examples just "h"), and the "k" in kNN!

These parameters are so-called "hyperparameters", i.e., prior knowledge that you bring on the table

Oftentimes, a method may works with poorly chosen hyperparameters.
However, better hyperparameters typically lead to drastically better performance.

The task of hyperparameter optimization is sometimes called "model selection problem"

The swiss army knife of model selection in supervised learning (known from PR):

k-fold cross-validation!

---> ML estimate for the density estimation hyper parameters via cross-validation.

Assume a data split into J folds: $S^j_{train} = S \setminus \{\vec{x}_{\lfloor \frac{N}{J} \rfloor \cdot j}, \dots, \vec{x}_{\lfloor \frac{N}{J} \rfloor \cdot (j+1)-1}\}$, $S^j_{test} = S \setminus S^j_{train}$

(S: all samples)

Let $\vec{\alpha}$ denote the unknown hyperparameters,

and $p_j(\vec{x}|\vec{\alpha})$ the density estimate for fold S^j_{train} on hyperparameters $\vec{\alpha}$

The Maximum Likelihood estimate is now obtained by evaluating

$$\vec{\alpha}^* = \arg\max_{\vec{\alpha}} \prod_{j=1}^{J} \prod_{\vec{x}_i \in S^j_{test}} p_j(\vec{x}_i|\vec{\alpha})$$

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

Density Estimation Links, Hyperloop to Hastie/Tibshirani/Friedman, Bias & Variance

Note: density estimation naturally links to other machine learning tasks:

With this insight, we can start to work with our second textbook, namely "The Elements of Statistical Learning" by Hastie, Tibshirani, and Friedman.
Abbreviation in this class: HTF

We will use selected parts of Sec. 6 to complement our knowledge (-> see next video!)

One note: before we start:
Style and priorities of HTF are somewhat different from Bishop.

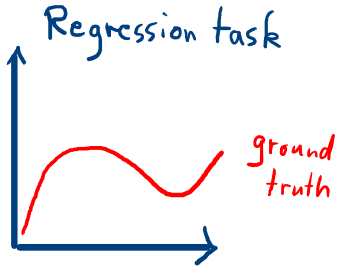
Also, HTF Sec. 6 repeatedly mentions "bias" and "variance", which are briefly introduced below.

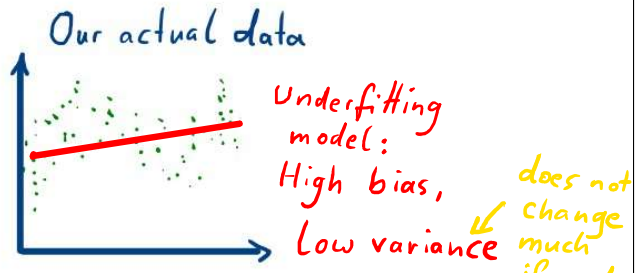
Bias and Variance

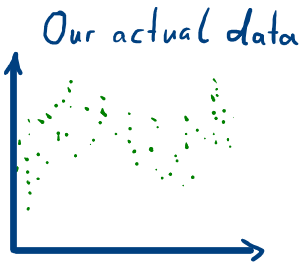
Bias: Estimation error due to model undercomplexity

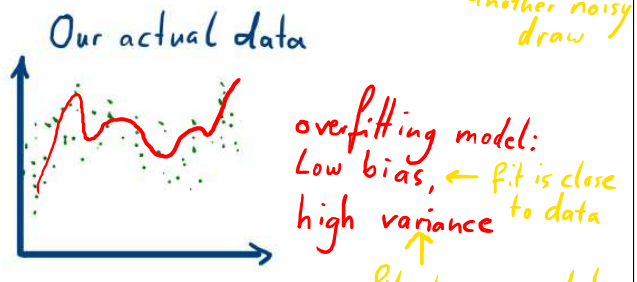
Variance: Estimation error due to model overcomplexity

Example:









Goal: find a compromise between bias and variance.

Further Considerations on Kernel Smoothing

In HTF: kernel density estimation is in Sec. 6.6.1

However, let us start to read a little bit earlier to get some context:
interesting remarks and caveats are scattered throughout Sec. 6,
and with our existing knowledge we can harvest many of those for only little additional cost

Specifically, please study the sections

- 6 until 6.3 (inclusively)

- 6.6.1 and 6.6.2

- 6.8 until 6.9 (inclusively)

--> I will briefly browse through these pages with you now, to highlight some cool locations.
However, please take the time to completely study these sections.

Derivation of HTF Eqn. (6.8) and Eqn. (6.9) (-> Programming Exercise 3)

Line equation: $y_i = \beta(x_0) x_i + \alpha(x_0)$

Eqn. (6.7):

$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} \underbrace{K_\lambda(x_0, x_i)}_{\text{Kernel weight}} \underbrace{[y_i - \alpha(x_0) - \beta(x_0) x_i]^2}_{\text{residual error of line fit wit } \alpha(x_0), \beta(x_0)}$$

Eqn. (6.8): $\hat{f}(x_0) = b(x_0)^T (B^T W(x_0) B)^{-1} B^T W(x_0) \vec{y}$

where $b(x) = \begin{pmatrix} 1 \\ x \end{pmatrix}$ and $B = \begin{pmatrix} b(x_1)^T \\ \vdots \\ b(x_N)^T \end{pmatrix}$ and $W(x_0) = \text{diag}(K_\lambda(x_0,x_1), \ldots, K_\lambda(x_0,x_N))$

To understand where Eqn. (6.8) comes from, you need to backtrack all the way to the beginning of linear regression → HTF Sec. 3.2 (page 44)

Eqn. (3.2): $RSS(\beta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 = \sum_{i=1}^{N} K(x_i) \cdot (y_i - 1 \cdot \beta_0 - x_i \beta_1)^2$

(β: different β than above!; $1 \to \begin{pmatrix} 1 \\ x_i \end{pmatrix}$; $\beta \to \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$)

This is Eqn(3.2) adapted to our needs in Eqn(6.7)

In matrix notation:

Eqn(3.3) $RSS(\beta) = K(x_i) \cdot (\vec{y} - X \cdot \vec{\beta})^T \cdot (\vec{y} - X\vec{\beta})$

Look for minimum:

Eqn.(3.4) (check for 2nd. derivative omitted)

$$\frac{\partial RSS}{\partial \beta} = \cancel{-2} X^T K(x_i) (\vec{y} - X\vec{\beta}) \stackrel{!}{=} 0$$

the minimum

We solve this expression for the regression parameters $\vec{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$

$$X^T K(x_0) \vec{y} - X^T K(x_0) X \vec{\beta} = 0$$

$$X^T K(x_0) X \vec{\beta} = X^T K(x_0) \vec{y}$$

$$\vec{\beta} = \underbrace{(X^T K(x_0) X)^{-1} \cdot X^T K(x_0) \vec{y}}_{= \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}}$$

Line equation: $y_i = \beta(x_0) x_i + \alpha(x_0)$

Eqn. (6.8): $\hat{f}(x_0) = b(x_0)^T (B^T W(x_0) B)^{-1} B^T W(x_0) \vec{y}$ //

$$= \begin{pmatrix} 1 \\ x_0 \end{pmatrix}^T \cdot \begin{pmatrix} \alpha(x_0) \\ \beta(x_0) \end{pmatrix} = \alpha(x_0) + \beta(x_0) \cdot x_0$$

Eqn. (6.9) can be directly obtained from Eqn. (6.8):

Eqn. (6.8): $\hat{f}(x_0) = \underbrace{b(x_0)^T (B^T W(x_0) B)^{-1} B^T W(x_0)}_{\vec{\ell}(x_0)} \vec{y}$

$$= \vec{\ell}_{(x_0)}^T \vec{y} = \sum_{i=1}^{N} \ell_i(x_0) \cdot y_i$$

Classification and Regression Trees (CART)

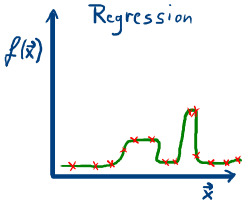
Remember that the kernel smoothing methods are "memory-methods", i.e., in general the whole dataset needs to be fully stored in order to be able to respond to queries.

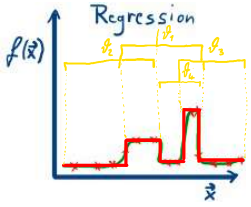
Also remember that we used relatively simple neighborhood operators
(kernel smoothing: fixed domain, k-NN: sample weight averaging)

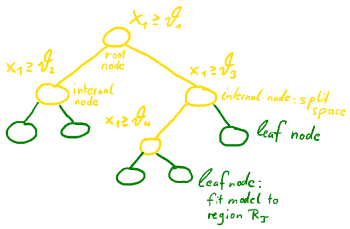
Question: how can we gain more flexibility?

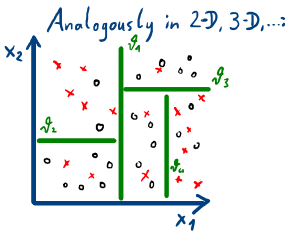
--> Classification and Regression Trees ("CART", Breiman 1984)
References: Bishop Sec. 14.4 (pp. 663-666), HTF Sec. 9.2 (pp. 305-316)

Idea: hierarchical subdivision of the feature domain









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

read "Bishop Eqn. 14.29"

$$y_J = \frac{1}{N_J} \sum_{x_n \in \mathcal{R}_J} t_n \qquad (B\ 14.29)$$

Training:
Greedy tree growth from root to leaves
At each node: exhaustive search for best split dimension & position
Objective function for regression: Minimize sum of squared errors

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

$$Q_J(T) = \sum_{\vec{x}_n \in \mathcal{R}_J} (t_n - y_J)^2 \qquad (B\ 14.30)$$

J: region index
$\mathcal{R}_J$: region/leaf w/ index J
t_n: sample label
y_J: sample avg. in $\mathcal{R}_J$

How tall shall the tree become? --> difficult to tell
Approach: grow each node until only few samples are left, then prune (i.e., merge irrelevant splits)

Pruning criterion is tradeoff between model accuracy and model complexity:

$$C(T) = \sum_{J=1}^{|T|} Q_J(T) + \lambda |T|$$

λ: regularization parameter

Classification trees are similarly grown and evaluated:
the main difference is the use of a different objective function during training.
Option 1: Negative cross-entropy:

$$Q_J(T) = \sum_{k=1}^{K} p_{Jk} \cdot \ln p_{Jk}$$

Option 2: Gini index:

$$Q_J(T) = \sum_{k=1}^{K} p_{Jk} \cdot (1 - p_{Jk})$$

k: class label
K: total number of classes
p_{Jk}: relative frequency of samples from class k in region $\mathcal{R}_J$

Both metrics are better for tree growing than the misclassification rate (cf. Bishop, and HTF Fig. 9.3), and they are differentiable.

Please also note the remarks "Other Issues" in HTF Sec. 9.2.4, particularly on instability of trees: high variance = tendency to overfit the training data

Random Forests

CART's overfitting can be reduced with an ensembling method called bagging
Bagging: Train multiple trees with different (randomly drawn) subsets of the data, average result

A Random Forest is essentially CART plus bagging plus extra randomization
Reference: HTF Sec. 15-15.3.4 and 15.4.3

Randomization reduces the variance of the classifier:
- Averaging B i.i.d. (independent and identically distributed) random variables, each with variance σ^2 leads to variance $\frac{1}{B}\sigma^2$
- Averaging B i.d. (identically distributed) random variables with correlation ρ ("rho") leads to variance

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \qquad \text{(HTF 15.1)}$$

-> with bagging, B goes to infinity, but ρ remains constant, so we are "stuck" at the variance $\rho \cdot \sigma^2$

Thus, random forests additionally randomly draw at each node a subset of m feature dimensions for finding the next split. This further decorrelates the trees. (note that $m=1$ is also possible!)

one such tree is oftentimes called a "weak learner", because the individual tree is oftentimes only slightly better than guessing. ⇒ good news: an ensemble of weak learners can be strong!

The prediction of the whole Random Forest is just the average of the individual tree predictions:

$$\hat{f}(\vec{x}) = \frac{1}{B} \cdot \sum_{b=1}^{B} T(\vec{x}; \vec{\theta}_b) \qquad \text{(HTB 15.2)}$$

($\vec{\theta}_b$: parameters of tree b)

Remarks on Random Forests:

- A further "escalation" of the weak learner idea:
  1. replace the exhaustive search for the best split by a random draw of candidate splits
  2. the candidate with the optimum value of the objective function is used.

  In python, this variant is called a "Extremely Randomized Forest"

- Out of Bag (OOB) Samples (cf. HTF Sec. 15.3.1): 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 (cf. HTF Sec. 15.3.4):
  - 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 (cf. HTF Sec. 15.4.3):
  The combination of
  a) the partitioning of the feature space into small tiles with few samples,
  b) the variability of these partitions across trees, and
  c) the averaging of the individual tree responses

  effectively creates a data-driven version of our nearest neighbors!

WOW! COOL!

Simplifications of the Feature Space: Outline & Agenda

What we have so far: different options for feature space representations
- kernel smoothing/non-parametric DE (local operators from fixed neighborhood relationships)
- trees and tree-based ensembles (local operators from learning-based feature space partitioning)

Question: can we further tailor our feature space representations towards specific goals?

Potential goal 1: clustering = grouping of data into meaningful units

Potential goal 2: manifold learning = neighborhood-preserving dimensionality reduction

Properties of clustering:

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

Considered methods and questions:
* k-means
* Gaussian Mixture Models
* Mean shift

* Model Selection Problem: how can we estimate the hyperparameters?

Properties of Manifold Learning:

Goal: represent information in a lower-dimensional space

Applications: classification, regression, visualization, data exploration

Considered methods and questions:
* (PCA) <- already known!?
* Multi-dimensional Scaling (MDS)
* A ride along notable non-linear methods to t-SNE:
  ISOMAP -> Locally linear embedding (LLE) -> Laplacian Eigenmaps (LE)
  -> t-Stochastic Neighborhood Embedding (t-SNE)

* Model Selection: what target dimension is appropriate?

This section closes with a spotlight on applications of the Graph Laplacian,
which nicely links back to our previous topics:
-> Graph Laplacian + k-means = spectral clustering
-> LE + forest-based density estimation = forest-based manifold learning

Mean Shift

Comaniciu, Meer: "Mean Shift: A Robust Approach Toward Feature Space Analysis", IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, no. 5, May 2002, pp. 603-619.

Straightforward extension of kernel smoothing: hill-climbing on the PDF to a mode (-> maximum) (this idea is also known as "bump hunting")

Let us denote the multivariate kernel density estimate as $p(\vec{x}) = \frac{1}{N}\sum_{k=1}^{N} K_\lambda(\vec{x}_i, \vec{x})$

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

-> Computation of the gradient: $\nabla p(\vec{x}) = \nabla \frac{1}{N}\sum_{k=1}^{N} K_\lambda(\vec{x}_i, \vec{x}) = \frac{1}{N}\sum_{k=1}^{N} \nabla K_\lambda(\vec{x}_i, \vec{x})$

Comaniciu & Meer now use a trick: they use a radialsymmetric kernel (-> Eqn.(4) right), i.e.

$$K_\lambda(\vec{x}_i, \vec{x}) = c_d \cdot k_\lambda(\underbrace{\|\vec{x}_i - \vec{x}\|^2}_{s})$$

First derivative: $\frac{\partial k_\lambda(s)}{\partial s} = k_\lambda'(s)$ ("$g(x)$" in the paper)

$$\frac{\partial s}{\partial \vec{x}} = \frac{\partial (\vec{x}_i - \vec{x})^T(\vec{x}_i - \vec{x})}{\partial \vec{x}} = -2(\vec{x}_i - \vec{x})$$

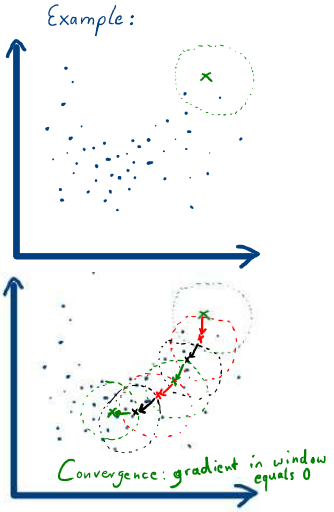
$$\Rightarrow \nabla p(\vec{x}) = \frac{1}{N}\sum_{i=1}^{N} c_d \cdot k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)(-2(\vec{x}_i - \vec{x})) \stackrel{!}{=} 0$$

$$\Rightarrow \nabla p(\vec{x}) = \frac{1}{N}\sum_{i=1}^{N} c_d \cdot k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)(-2(\vec{x}_i - \vec{x})) \stackrel{!}{=} 0$$

$$\sum_{i=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)\cdot(\vec{x}_i - \vec{x}) = 0$$

$$\sum_{i=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)\cdot\vec{x}_i - \sum_{i=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)\cdot\vec{x} = 0$$

$$\underbrace{\frac{\sum_{i=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)\cdot\vec{x}_i}{\sum_{i=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)}}_{\text{normalized gradient!}} - \vec{x} = 0$$



---> gradient ascent: compute the normalized (scaled) gradient and follow the gradient direction

Mean shift algorithm:

1.) Calculate the mean shift vector: $m^{(t)}(\vec{x}) = \frac{\sum_{k=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)\cdot\vec{x}_i}{\sum_{k=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)} - \vec{x}^{(t)}$

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

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} + m^{(t)}(\vec{x}) = \frac{\sum_{i=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)\cdot\vec{x}_i}{\sum_{i=1}^{N} k_\lambda'(\|\vec{x}_i - \vec{x}\|^2)} + \vec{x}^{(t)} - \vec{x}^{(t)}$$

3.) goto 1.) unless convergence

With Epanechnikov kernel (HTF Eqn. (6.4)): mean shift vector is mean in λ-sphere, but Gauss kernel also possible (see paper). Numerical caveat: merge multiple nearby bumps if necessary

k-Means

Reference: HTF Sec. 14.3.6 (or Bishop Sec. 9.1)

Arguably the most well-known hard clustering algorithm.
Optimizes k vectors that minimize the within-cluster distance $W(C)$ w.r.t. the squared Euclidean distance

$$W(C) = \frac{1}{2} \cdot \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} \|\vec{x}_i - \vec{x}_{i'}\|^2$$

$C(i) = k$: check whether $\vec{x}_i$ belongs to cluster k
N_k: # points in cluster k
μ_k: mean of all points in cluster k

$$= \sum_{k=1}^{K} N_k \cdot \sum_{C(i)=k} \|\vec{x}_i - \vec{\mu}_k\|^2 \quad \text{(HTF Eqn. 14.31)}$$

The optimization task is

$$\min_{C, \{m_k\}_1^K} \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|\vec{x}_i - m_k\|^2 \quad \text{(HTF Eqn. 14.33)}$$

$m_k = \mu_k$, otherwise we are not optimal

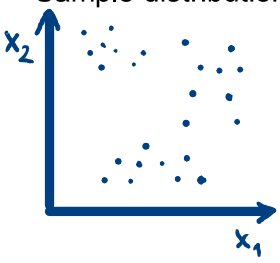
k-means algorithm:

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

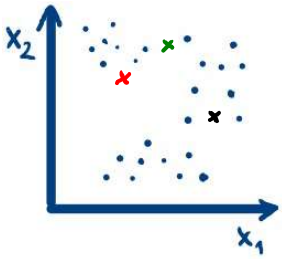
Note: the solution is only locally optimal, hence different initializations can lead to different results

The iteration partitions the space, this is called a Voronoi tesselation

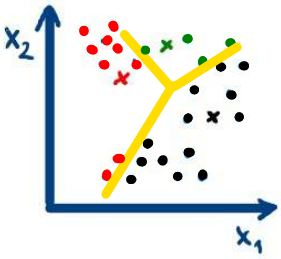
Sample distribution:



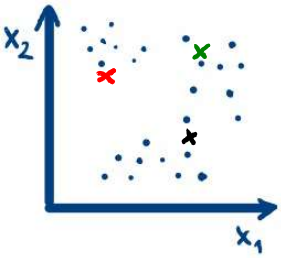
Initial cluster centers:



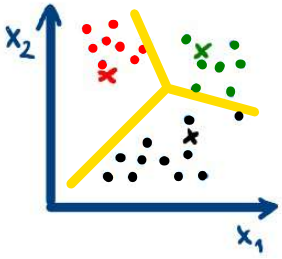
Assignments to nearest center:



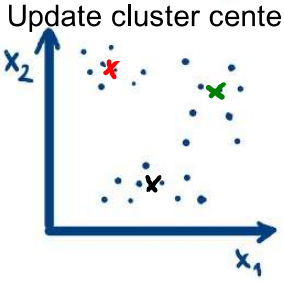
Update cluster centers:



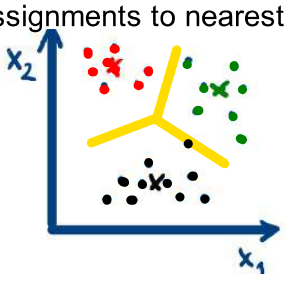
Assignments to nearest center:



Update cluster centers:



Assignments to nearest center:



Gaussian Mixture Models (GMM) Soft Clustering

References: Bishop Sec. 9.2 (including 9.2.1 and 9.2.2)
[Also possible: HTF 8.5. HTF 8.5.1 starts with an instructive 2-component mixture model, but then hastes within only two pages (HTF 8.5.2) through Bishop Sec. 9.2 and 9.3]

Gaussian Mixture Model: $p(\vec{x}) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(\vec{x} \mid \vec{\mu}_k, \Sigma_k)$ (B Eqn. 9.7)

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

To fit the model, we use a K-dim. binary indicator $\vec{z}$, $z_k = \{0,1\}$, $\sum_{k=1}^{K} z_k = 1$

Note that $\vec{z}$ is a "one-hot" vector, where π_k is the marginal probability, i.e., $p(z_k = 1) = \pi_k$

This can also be written as $p(\vec{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$ (the exponent cancels all terms except the one where $z_k = 1$)

We further model a Gaussian as a distribution conditioned on $\vec{z}$,

$$p(\vec{x} \mid z_k = 1) = \mathcal{N}(\vec{x} \mid \vec{\mu}_k, \Sigma_k) \quad \text{or equivalently}$$
$$p(\vec{x} \mid \vec{z}) = \prod_{k=1}^{K} \mathcal{N}(\vec{x} \mid \mu_k, \Sigma_k)^{z_k}$$

The joint distribution $p(\vec{x}, \vec{z}) = p(\vec{x} \mid \vec{z}) \cdot p(\vec{z})$ consists of the conditional distribution $p(\vec{x} \mid \vec{z})$ and the distribution of the hidden variable $p(\vec{z})$

As a result, the GMM can also be written as a marginalization over the hidden variables:

$$p(\vec{x}) = \sum_{\vec{z}} p(z) \cdot p(x \mid z) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(\vec{x} \mid \mu_k, \Sigma_k)$$

Furthermore, we note that $p(z_k = 1 \mid \vec{x}) \overset{\text{Bayes}}{=} \frac{p(z_k = 1) \cdot p(\vec{x} \mid z_k = 1)}{p(\vec{x})} = \frac{\pi_k \cdot \mathcal{N}(\vec{x} \mid \vec{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(\vec{x} \mid \vec{\mu}_j, \Sigma_j)}$ (B Eqn. 9.13)

This conditional probability is called "responsibility" $\gamma(z_k)$ of component k for sample $\vec{x}$

The GMM is fitted via maximum likelihood using the Expectation-Maximization (EM) algorithm.

GMM fitting is like a soft clustering variant of kMeans - both are almost identical! → See HTF Sec. 14.3.7

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

$$\ln p(X \mid \vec{\pi}, \vec{\mu}, \Sigma) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(\vec{x}_n \mid \vec{\mu}_k, \Sigma_k) \right)$$ (B Eqn. 9.14)

As usual, the optimum values for $\vec{\mu}_k$, Σ_k, π_k are found by setting the derivative to 0.

The responsibilities $\gamma(z_k)$ are part of that optimum, e.g.,

$$\frac{\partial \ln p(X \mid \vec{\pi}, \vec{\mu}, \Sigma)}{\partial \vec{\mu}_k} = \sum_{n=1}^{N} \underbrace{\frac{\pi_k \cdot \mathcal{N}(\vec{x}_n \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(\vec{x}_n \mid \vec{\mu}_j, \Sigma_j)}}_{\gamma(z_{nk})} \cdot \Sigma_k^{-1} (\vec{x}_n - \vec{\mu}_k)$$ (B Eqn. 9.16)

← excessive use of the chain rule!

and analogously for Σ_k.

The final update equations are listed at the end of Bishop Sec. 9.2.2:

- "Expectation" updates the responsibilities $\gamma(z_k)$ (see Eqn. above)
- "Maximization" updates the GMM params:

$$\vec{\mu}_k^{new} = \frac{1}{N_k} \cdot \sum_{n=1}^{N} \gamma(z_{nk}) \cdot \vec{x}_n$$
$$\Sigma_k^{new} = \frac{1}{N_k} \cdot \sum_{n=1}^{N} \gamma(z_{nk}) \cdot (\vec{x}_n - \vec{\mu}_k^{new}) \cdot (x_n - \vec{\mu}_k^{new})^T$$
$$\pi_k^{new} = \frac{N_k}{N}$$
$$N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

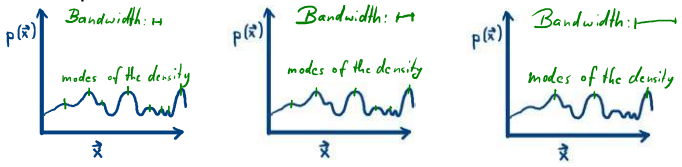
Note that this is a weighted fit, similar to Ex.3

Model Selection for Mean Shift

References: Paper Comaniciu, Meer: "Mean Shift: (..)" PAMI 2002 (-> studOn), Sec. 3.1

Model selection problem: kernel size ("bandwidth") impacts the result
(remark: also the choice of the kernel may influence the result, but arguably to a lesser extent)

Example in 1-D:



Bandwidth impacts the number of modes, and maybe their location (-> why the location?)

How can this model selection task be solved?

For supervised tasks, cross-validation is a universal model selector.

However, using cross-validation (CV) for clustering is complicated by two issues:
a) clustering is unsupervised, i.e., there are no "ground truth" labels as CV targets
b) 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

Comaniciu and Meer briefly discuss the bandwidth selection problem in Sec. 3.1 of their paper

That discussion does not give a clear answer (which is not surprising, given that the task of clustering is not sharply defined).

The discussion is nevertheless instructive, because it describes the space of possibilities to tackle the bandwidth selection problem:

1.) define a statistical measure that, under some further constraints, selects the bandwidth that minimizes the asymptotic mean integrated square error (AMISE)
2.) cluster stability as a (heuristic) indicator of structure: select a bandwidth from within the largest interval where the number of estimated clusters does not change
3.) define and optimize a metric on the cluster quality, e.g. inter- versus intra-cluster distances or isolation and connectivity
(-> why are inter- versus intra-cluster distances not necessarily a good choice here?)
4.) an oracle decides the cluster quality (i.e., using some outside information, e.g., a user choice)
5.) do not decide for a global bandwidth, but instead use a locally-defined, adaptive bandwidth
(-> we know this idea already from our discussion of kernel smoothing versus random forests!).
This local bandwidth can again be set by any scheme among a number of choices.

Model Selection for kMeans

Reference: HTF Sec. 14.3.11 (pls. backtrack at will into the short, easy-to-follow earlier subsections)

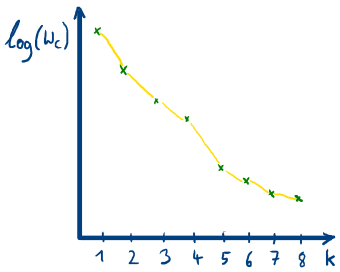
The model selection problem for clustering requires some form of hack.
The previous video/slides span the space of possibilities.

How can we tackle model selection for kMeans?

kMeans optimizes the within-cluster distance. An optimality criterion on the within-cluster distance would fall under category 3) of the options in the previous video.

A naive approach might attempt to calculate the within-cluster distance for different values of k

Note, however, that this will result in a continuously decreasing curve:

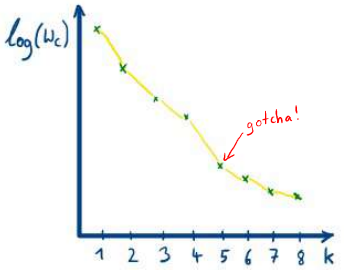


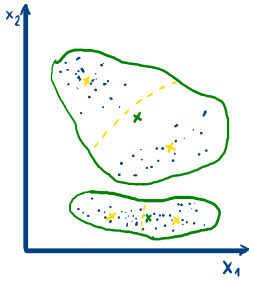
This makes sense: average distances to cluster centers decrease with increasing number of centers
(also reasonable in a broader context: model fitting error decreases with increasing model complexity)

One alternative is to track the decrease of the within-cluster distance (a.k.a. "ellbow method")

Splitting a natural cluster:
small change

Splitting disparate points:
bigger change





However, choosing a concrete threshold on the decrease is difficult:
curve segments for smaller k have a "natural" larger decrease than curve segments for larger k

Tibshirani's trick: the "gap statistic"
normalize the curve of the within-cluster distances 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

$$K^* = \arg\min_K \{ K \mid G(K) \geq G(K+1) - s'_{K+1} \}$$ (HTF Eqn. 14.39)

where $G(K) = \log(W_K^{unif.}) - \log(W_K^{data})$

$s'_{K+1} = s_K \cdot \sqrt{1 + 1/20}$

s_K = std. dev. of $\log(W_K^{unif.})$ across 20 runs

Curse of Dimensionality

References: Bishop Sec. 1.4

So far, the data dimensionality was only 1-D or 2-D.
Real data oftentimes consists of 100s of dimensions.

It is important to note that the difficulty of all pattern recognition / pattern analysis tasks increases with increasing data dimensionality.
This increase in difficulty is sometimes referred to as "Curse of Dimensionality" (Bellman, 1961)

As a take-home message, it is always preferable to work with data of lower dimensionality.
This may be possible: the actual information can often be represented in a lower dimension.

We illustrate here three difficulties associated with high-dimensional data, and use this as a motivation to look into dimensionality reduction/manifold learning techniques in the next videos.

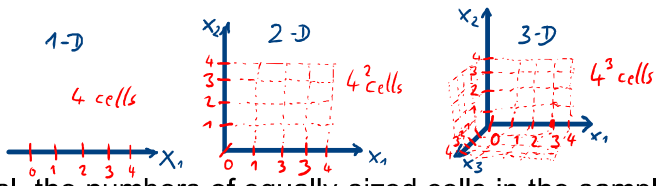
Issue 1:
How to plot or browse data that exceeds 3 or 4 dimensions?

For example, input data can be high-dimensional, as in the case of hyperspectral satellite images: these images oftentimes have hundreds of color channels.

As another example, feature spaces can be very high-dimensional, which can make it quite difficult to find issues in classification tasks.

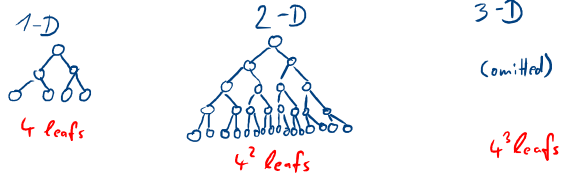
Issue 2:
The subdivision of the sample space becomes increasingly more complex with increasing dimensionality.

We already spent some thoughts on partitioning the feature space into small, local components (-> see Sec. 1 on kernel density estimation and random forests)



In general, the numbers of equally-sized cells in the sample space grows exponentially in the dimensions.
Memory methods (i.e., our kernel density estimators) are not affected by this, but any other, smarter classifier/regressor needs much more model parameters to distinguish this exponentially growing number of cells.

Bishop illustrates this on fitting a polynomial (B 1.74), but we can transfer this argument to CART: here, the worst case growth of the number of tree nodes is also exponential.
(--> note, however, that the expected increase in the number of parameters is almost certainly lower)



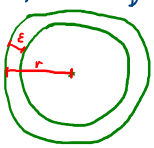
Issue 3:
In a high-dimensional space, almost all samples lie at the boundary.

This is probably the most surprising property of high-dimensional spaces.

Consider a D-dimensional sphere with radius 1, and samples are uniformly distributed in that sphere.

The volume of the sphere is $V_D(r) = K_D \cdot r^D$ K_D: constant volume factor (not important here), r: radius of the sphere

Let $f(\varepsilon)$ denote the fraction of data at the boundary of the sphere, i.e., between $V_D(1)$ and $V_D(1-\varepsilon)$



This fraction is $f(\varepsilon) = \frac{V_D(1) - V_D(1-\varepsilon)}{V_D(1)} = \frac{1-(1-\varepsilon)^D}{1} = 1-(1-\varepsilon)^D$

Thus, $f(\varepsilon)$ rapidly approaches 1!

For example: $D = 10$, $\varepsilon = 0.1$ (10% of the radius!): $f(0.1) = 65\%$

$D = 100$, $\varepsilon = 0.01$: $f(0.01) = 63\%$ ← in a 100-dim. space, almost 3/3 of the uniformly distributed data is located at the outer shell within 1% of the radius

Thus, if we draw a high-dimensional sample from a uniform distribution, it is with very high probability at the boundary of the space.

If we draw two such samples, their distance will with increasing probability be close to some (high) average distance value.

$x_1 = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix}$ $x_2 = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix}$ $d_{Eucl.}(x_1, x_2) = \sum_{d=1}^{D} \|x_{1,d} - x_{2,d}\|_2^2 \approx 0.5 \cdot D \cdot (\text{small dist}) + 0.5 \cdot D \cdot (\text{large dist})$

As a consequence, distance values are less discriminative in high-dimensional spaces

Two more thoughts on the issue that most samples are located at the boundary, taken from the book HTF Sec. 2.5 (<- not a mandatory read):

- Many methods have difficulties at the boundary of the sample space.
For example, our kernel estimator became inacurrate at the boundary, which is why we calculated the local linear kernel regressor in the exercise.
In increasing dimensions of the sample space, more and more samples are located at the boundary.

- With increasing dimensionality, the sample density becomes increasingly sparse
(we would need to exponentially increase the number of samples to maintain constant density, but where should that exponential number of samples come from?)

HTF illustrate this with a nice calculation on the sphere from the previous slide:
the median distance between the center of the sphere and its closest(!) point is

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

D: dimensionality of the sample space
N: total number of samples

which leads to similarly surprising numbers as on the previous slide.

For example, for $N = 500$, $D = 10$: $d(10, 500) \sim 0.52$

Thus, sample sparsity becomes an issue in high-dimensional spaces.

Principal Component Analysis (PCA)

Reference: Bishop Sec. 12.1.1

PCA is also known as "Karhunen-Loeve Transform" or "KL Transform". PCA is the work horse of many fields in engineering and science. It is also a central tool for data analysts.

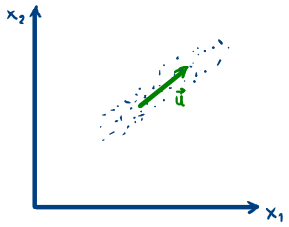
PCA is the first go-to method if you aim to reduce the dimensionality of your data.

Quick synopsis:
- PCA is a linear projection onto an orthogonal basis. $\vec{u}_i^T \cdot \vec{u}_j = \delta_{ij} = \begin{cases} 1 \text{ if } i=j \\ 0 \text{ otherwise} \end{cases}$
- This basis coincides with the eigenvectors of the mean-free covariance matrix of the data.
- Thus, PCA consists mostly of normalizing the data and performing an eigenvector decomposition
- The magnitude of the eigenvalues indicates the importance of a dimension w.r.t. the covariance

Core idea: Find a linear mapping $\Phi: \mathbb{R}^d \to \mathbb{R}^{d'}, \quad d' \ll d,$ that maximizes the variance (spread) of the data along each dimension.

Objective function: $J = \sum_{i,j=1}^{N} (\Phi\vec{x}_i - \Phi\vec{x}_j)^T (\Phi\vec{x}_i - \Phi\vec{x}_j) + \lambda(\Phi^T\Phi - 1)$ where $\vec{x}_i, \vec{x}_j \in \mathbb{R}^d$

Now assume zero mean samples, i.e., $\sum_{i=1}^{N} \vec{x}_i = 0$.



Goal of the following calculation:
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.

Let $\vec{u} \in \mathbb{R}^d$ denote an arbitrary direction vector, normalized to unit length, i.e., $\vec{u}^T\vec{u} = 1$

Note that the inner product $\vec{u}^T\vec{x}$ projects $\vec{x}$ onto a 1-D space,

and the variance of the projected data is $\frac{1}{N}\sum_{i=1}^{N} (\vec{u}^T\vec{x}_i - \vec{u}^T\bar{\vec{x}})^2 = \vec{u}^T S \vec{u}$ where $\bar{\vec{x}}$ is the component-wise mean vector over all $\vec{x}_i$

with the covariance matrix $S = \frac{1}{N}\sum_{i=1}^{N} (\vec{x}_i - \bar{\vec{x}})(\vec{x}_i - \bar{\vec{x}})^T$

(Note that expanding S in $\vec{u}^T S \vec{u}$ is identical to $\frac{1}{N}\sum_{i=1}^{N} (\vec{u}^T(\vec{x}_i - \bar{\vec{x}})(\vec{x}_i - \bar{\vec{x}})^T\vec{u})$)

Thus, let us find that $\vec{u}$ that maximizes the variance:

$$\vec{u}^T S \vec{u} + \lambda(1 - \vec{u}^T\vec{u}) \to \max.$$

λ: Lagrange multiplier that includes the constraint $\vec{u}^T\vec{u} = 1$

Maximization is done by calculating the derivative w.r.t. $\vec{u}$ and to set the equation equal to 0:

$$\frac{\partial}{\partial \vec{u}} \vec{u}^T S \vec{u} + \lambda(1 - \vec{u}^T\vec{u}) \stackrel{!}{=} 0$$
$$\Leftrightarrow \quad 2 \cdot S\vec{u} = 2 \cdot \lambda \vec{u}$$
$$\Leftrightarrow \quad S\vec{u} = \lambda\vec{u}$$

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".

Not shown here: This 1-D subspace can be increased to arbitrary more dimensions by induction: just repeat this procedure after projecting the data to the space orthogonal to our 1-D space.

Multidimensional Scaling (MDS)

MDS is equivalent to PCA, but it takes distances between points as inputs.

Task: Given a distance matrix $\mathcal{D}^2 = [d_{ij}^2]$, $1 \le i,j \le N$, $d_{ij} = (\vec{x}_i - \vec{x}_j)^T(\vec{x}_i - \vec{x}_j)$, calculate $X = (\vec{x}_1, \dots, \vec{x}_N) \in \mathbb{R}^{d' \times N}$

$\vec{x}_1, \dots, \vec{x}_N$ have zero mean.

The components of the distance matrix can be written in matrix notation:

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

$$\Rightarrow \mathcal{D}^2 = \operatorname{diag}(X^TX)\cdot\mathbb{1}^T + \mathbb{1}\cdot\operatorname{diag}(X^TX)^T - 2X^TX$$

where $\mathbb{1} = (1,\dots,1)^T \in \mathbb{R}^N$ and $\operatorname{diag}(X)$ constructs a matrix where only the diagonal entries of X are set (and the rest is 0)

This equation links distances to actual coordinates.
We can get direct access to the coordinates after multiplying $\mathcal{D}^2$ by a centering matrix $C = (I - \frac{1}{N}\mathbb{1}\mathbb{1}^T)$:

$$\frac{1}{2}C\mathcal{D}^2C = -\frac{1}{2}(I - \frac{1}{N}\mathbb{1}\mathbb{1}^T)\big(\underbrace{\operatorname{diag}(X^TX)\mathbb{1}^T}_{(1)} + \underbrace{\mathbb{1}\cdot\operatorname{diag}(X^TX)^T}_{(2)} - \underbrace{2\cdot X^TX}_{(3)}\big)(I - \frac{1}{N}\mathbb{1}\mathbb{1}^T)$$

(1): $(I - \frac{1}{N}\mathbb{1}\mathbb{1}^T)\operatorname{diag}(X^TX)\cdot\mathbb{1}^T(I - \frac{1}{N}\cdot\mathbb{1}\cdot\mathbb{1}^T) =$

$(I - \frac{1}{N}\cdot\mathbb{1}\mathbb{1}^T)\cdot\operatorname{diag}(X^TX)\cdot\underbrace{\big(\mathbb{1}^T\cdot I - \frac{1}{N}\cdot\underbrace{\mathbb{1}^T\cdot\mathbb{1}}_{=N}\cdot\mathbb{1}^T\big)}_{=0} = 0$

(2): Analogously to (1)

(3): $-\frac{1}{2}(I - \frac{1}{N}\mathbb{1}\cdot\mathbb{1}^T)\cdot(-2X^TX)\cdot(I - \frac{1}{N}\mathbb{1}\mathbb{1}^T) =$

$= \big(I\cdot X^T - \underbrace{\frac{1}{N}\cdot(\mathbb{1}\cdot\mathbb{1}^T)X^T}_{=0\ (\text{zero mean!})}\big)\cdot\big(X\cdot I - \underbrace{\frac{1}{N}\cdot X\cdot\mathbb{1}\cdot\mathbb{1}^T}_{=0\ (\text{zero mean!})}\big) = X^TX$

Access X via singular value decomposition (SVD):

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

$$\Rightarrow X = \Sigma^{\frac{1}{2}}U \qquad \left(\text{where } \Sigma^{\frac{1}{2}} = \begin{pmatrix}\sqrt{\sigma_1} & & \\ & \sqrt{\sigma_2} & \\ & & \ddots \\ & & & \sqrt{\sigma_N}\end{pmatrix},\right.$$

since the square root of a diagonal matrix is just the square root of each entry.)

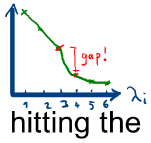
One notorious issue for any such dimensionality reduction method is to determine the target dimensionality of the projection:

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 eigenvalues, such that

$$d'^* = \underset{d'}{\operatorname{argmin}}\ x\% \le \frac{\sum_{i=1}^{d'}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$$

Independent of that, there is for all methods that rely on an eigendecomposition or a singular value decomposition the concept of the "eigenvalue gap".

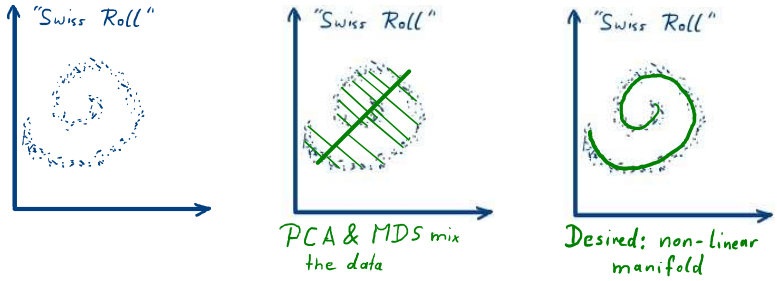


In theory, the eigenvalue distribution exhibits a gap when hitting the data's intrinsic dimensionality. We do not go much deeper into detail here, but how surprisingly similar is this to our elbow method and gap statistics in clustering!

PCA and MDS are linear methods, in the sense that they use one global linear projection mapping to transform the data.

This is fine in many cases - in fact, this is why PCA is so popular!

However, non-linear manifolds require non-linear manifold learning methods:



We will look at two concrete examples: ISOMAP and Laplacian Eigenmaps (LE)

ISOMAP, because it directly links to MDS, and LE as a representative of the theoretically quite rich family of methods that use the graph Laplacian.

There exist numerous other methods. To name only three of them:
- The classical "Sammon Mapping" and
- "Locally Linear Embedding" are commonly known in the community, and
- the currently very popular data visualization tool "t-SNE", which is a stochastic variant of the ideas discussed in this video.

Isometric Feature Mapping (ISOMAP)

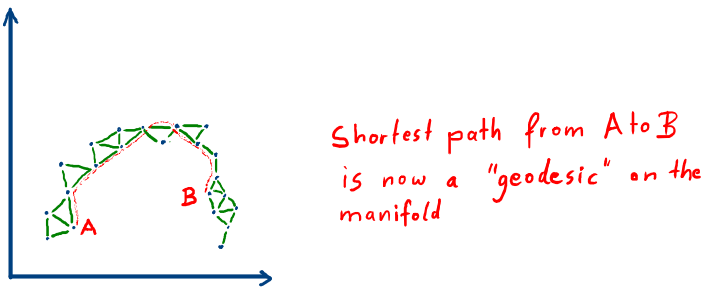
Straightforward non-linearity hack for MDS: find a mapping that preserves the global non-linear geometry of data by preserving the "geodesic" distances along the manifold

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

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 distance matrix (e.g. via the Floyd-Warshall algorithm, or repeated application of Dijkstra's algorithm)

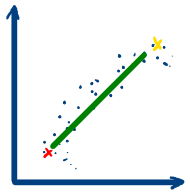
3. Perform MDS on that distance matrix



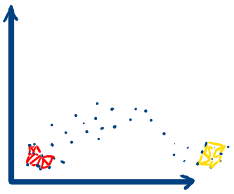
Laplacian Eigenmaps (LE)

PCA and MDS map data to a lower-dimensional space by preserving as much variance as possible.

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



PCA & MDS: "Large distances remain large"
keeps the red and yellow points apart.
However, this criterion automatically uses global information



LE: "Local neighborhoods remain together"
kee

Algorithm:
1. Build adjacency graph from the samples (e.g., using k-NN or fixed distance threshold)
2. Weight the edges in the graph
3. Calculate an eigendecomposition of the graph Laplacian
4. Perform low-dimensional embedding

Consider the task of mapping a weighted graph G onto a line (d' = 1), so that the connected vertices in the 1-D space are as close as possible.

(Generalization to arbitrary dimensions is not too difficult, but omitted in this derivation)

Objective function: $\sum_{i=1}^{N}\sum_{j=1}^{N}(x_i'-x_j')^2 w_{ij} \to \min.$

Note that x_i', x_j' are in the lower-dimensional space, but w_{ij} is calculated in the higher-dimensional space

where w_{ij} is a suitably chosen edge weighting, for example the heat kernel $w_{ij} = e^{-\frac{\|x_i - x_j\|_2^2}{t}}$ or a binary kernel $w_{ij} = \begin{cases} 1 & \text{if } \|x_i - x_j\|_2^2 < t \\ 0 & \text{otherwise} \end{cases}$

We also need an optimization constraint to prevent the trivial solution:

subject to $\vec{x}'D\vec{x}' = 1$, where $\vec{x}' = \begin{pmatrix} x_1' \\ \vdots \\ x_N' \end{pmatrix}$ and $D = \operatorname{diag}\left(\sum_{i=1}^{N} w_{1i}, \dots, \sum_{i=1}^{N} w_{Ni}\right)$

Let us rewrite the objective function:

$$\sum_{i,j}(x_i'-x_j')^2 \cdot w_{ij} = \sum_{i,j}(x_i'^2 + x_j'^2 - 2x_i'x_j') \cdot w_{ij}$$
$$= 2 \cdot \sum_i x_i'^2 \left(\sum_j w_{ij}\right) - 2 \cdot \sum_{i,j} x_i'x_j' \cdot w_{ij}$$
$$= 2 \cdot (\vec{x}'^T D \vec{x}' - \vec{x}'^T W \vec{x}')$$
$$= 2 \cdot (\vec{x}'^T (D-W) \cdot \vec{x}')$$

The matrix $L = D - W$ is also known as the graph Laplacian.

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

$$\Rightarrow \frac{\partial}{\partial x'}\left(\vec{x}'^T L \vec{x}' + \lambda(1 - \vec{x}'D\vec{x}')\right) = 2L\vec{x}' - \lambda D\vec{x}' \stackrel{!}{=} 0$$

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

Embedding for sample $\vec{x}_i$: select eigenvector e with smallest non-zero eigenvalue, $x_i' = e_i$ (see next video)

Laplacian Eigenmaps and Random Forests - Part 1: A Deeper Look into the Optimization

Reference: Shotton, Criminisi, Konukoglu: "Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Supervised Learning" (ONLY as backup!).

Recap: weighting function of Laplacian Eigenmaps is an suitably chosen kernel, e.g.

- the heat kernel (a.k.a. Gaussian kernel / RBF kernel): $w_{ij} = e^{-\frac{\|x_i - x_j\|_2^2}{t}}$
- the disc kernel $w_{ij} = \begin{cases} 1 & \text{if } \|x_i - x_j\|_2^2 < t \\ 0 & \text{otherwise} \end{cases}$

These kernels are calculated over a suitably chosen neighborhood (e.g., k-NN or a distance threshold)

Recall our discussion from Sec. 1 of the lecture:
- what if a globally fixed kernel function is too inflexible to represent our data?
- can we replace the kernel by a learning-based neighborhood measure?

-> Yes, we can!

"affinity" is any measure of similarity: it grows when the distance shrinks

We will see in this lecture how to replace our kernel-based affinities by affinities from Random Forests

in your professional life, you can also use any other learned embedding here, e.g., using a deep neural net.

However, to do this, we will have to overcome two technical hurdles:
1. How can train a Random Forest without labeled data?
2. How can we obtain a neighborhood operator from that Random Forest?

A Closer Look at the Random Forest Splitting Criterion (for Classification)

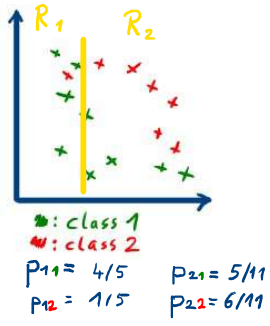
The objective function is the gini index or the cross-entropy.

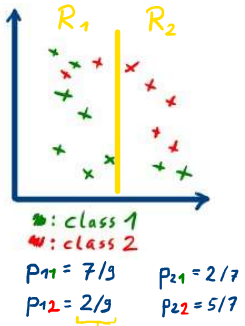
Without loss of generality, let us work with the cross-entropy: $Q_J(T) = \sum_{k=1}^{K} p_{Jk} \cdot \log p_{Jk}$

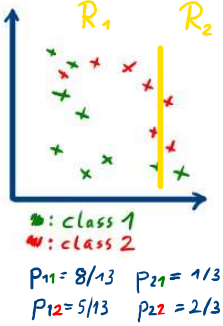
This equation is from Lecture 11 on CART

K: # classes
J: region index
p_{Jk}: relative frequency of samples of class k in region R_J

Let us check how the cross-entropy evaluates on different split candidates:

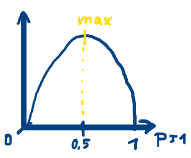






Shape of the entropy function:

$-\sum_{k=1}^{2} p_{Jk} \cdot \log p_{Jk}$ for 2 classes: $p_{J2} = 1 - p_{J1}$



Example: $-(p_{11} \cdot \log p_{11} + p_{12} \cdot \log p_{12}) = -(p_{11} \log p_{11} + (1-p_{11}) \cdot \log(1-p_{11}))$
$= -(7/9 \cdot \log(7/9) + 2/9 \cdot \log(2/9))$

A classifier needs regions with small entropy: these regions have a majority for one class.
We return that class when classifying a test sample for that region.

Criminisi, Shotton, Konokoglu (CSK) summarize the entropies of the left and right region in the information gain:

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

Mapping of symbols:

$H(S_j)$: entropy function, same as $Q_T(T)$, on tree node S_j ($\hat{=}$ region $R_1 \cup R_2$)

$S_j^L(\vartheta)$ and $S_j^R(\vartheta)$: regions R_1 and R_2 after split at boundary ϑ

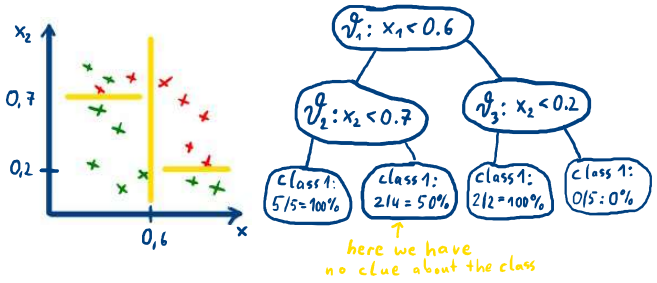
The Random Forests seeks splitting parameters ϑ such that the information gain is maximized. This minimizes the entropies of the left and right split regions $S_j^i(\vartheta)$.

(note that the entropy of the whole area $H(S_j) = R_1 \cup R_2$ is a constant, and hence irrelevant for maximization)

Reminder from the CART and Random Forest sections:

This maximization is performed greedily from the tree root to the leaves.

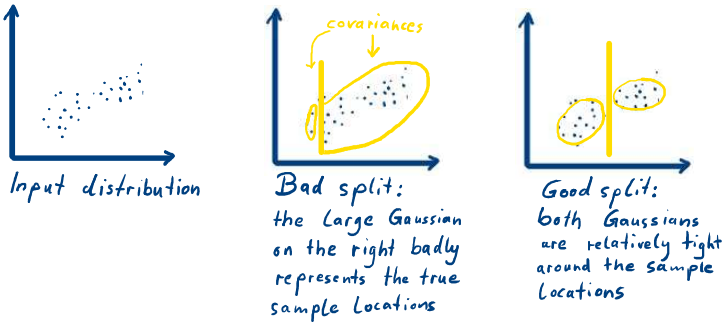
Eventually, the space is partitioned into many small, "informative" regions, where (ideally) one class has a clear majority in each region.



Laplacian Eigenmaps and Random Forests - Part 2: Density Forests and Manifold Forests

Question: How can we modify the Random Forest Classifier to work in an unsupervised scenario, i.e., with unlabeled data?

Idea in "Density Forests": a split is good if the samples in the regions are compactly clustered.
We can use, e.g., the covariance of a Gaussian to measure compactness:



To accomplish this, we can re-use the information gain, but we need to redefine the entropy as the volume of a Gaussian fitted to the samples in the left or right regions:

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

where $\Lambda(S_j^i(\vartheta))$: covariance matrix of the samples in region $S_j^i(\vartheta)$

$|\Lambda(..)|$: determinant of Λ gives the volume of a box around Λ

Plugged into the information gain, and constant factors removed, we obtain the objective function:

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

this factor will again also be constant for varying ϑ

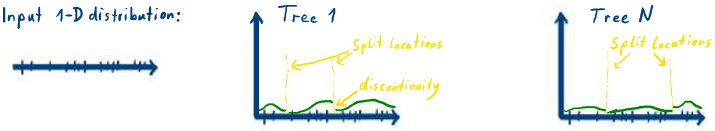
It is difficult, but rewarding to compare the objective functions for labeled and unlabeled data:

Why is it conceptually the same to seek a minimum covariance on unlabeled data to seeking a majority of labels in the labeled data case?

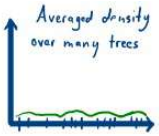
-> It helps to consider entropy as a measure for "surprise" or "the amount of guessing we need" - and we certainly do not like to do much guessing in our predictions.

The output of a Density Forest can be used as a probability density estimate:

Each individual tree provides a mixture of truncated Gaussians. In 1-D:



Averaging the output of multiple trees of the forest smoothes the boundaries:



To sample from that density:
(a) randomly select a tree,
(b) select a leaf w/ probability acc. to the # of leaf samples
(c) sample from that truncated Gaussian

Manifold Forests

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

-> Define affinities on the leaves of the Density Forest.
For each tree t, define an affinity matrix W^t: $W^t_{ij} = e^{-d^t(\vec{x}_i, \vec{x}_j)}$

where $d^t(\vec{x}_i, \vec{x}_j)$ is a distance function, e.g.,

Gaussian affinity: $d^T(\vec{x}_i, \vec{x}_j) = \begin{cases} \frac{(\vec{x}_i - \vec{x}_j)^T(\vec{x}_i - \vec{x}_j)}{t} & \text{if } leaf(\vec{x}_i) = leaf(\vec{x}_j) \\ \infty & \text{otherwise} \end{cases}$

Binary affinity: $d^T(\vec{x}_i, \vec{x}_j) = \begin{cases} 0 & \text{if } leaf(\vec{x}_i) = leaf(\vec{x}_j) \\ \infty & \text{otherwise} \end{cases}$

binary affinities in one leaf

$W^t =$ 

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.

Averaging removes the block structure.

However,
- the affinity between sample pairs that are in many trees in one leaf will be much higher than
- the affinity between sample pairs that are only occasionally in the same leaf.

This creates a strong sense of locality. This locality is directly learned from the data.

The affinity matrix W can be directly used as weight matrix for the Laplacian eigenmaps.

CSK use a normalized version of the graph Laplacian - don't worry, this is essentially equivalent to the Laplacian from our previous lecture.

Specifically, CSK calculate the graph Laplacian as

$$L = I - \Upsilon^{-\frac{1}{2}} W \Upsilon^{\frac{1}{2}}$$

where I: identity matrix
Υ: diagonal matrix $diag\left(\sum_{i=1}^{N} w_{1i}, \ldots, \sum_{i=1}^{N} w_{N,i}\right)$

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 eigenvalues $\lambda_i = 0$
(their number corresponds to the number of connected components in W)

2.) Arrange the eigenvectors associated with the next d' smallest eigenvalues in a matrix:

$$E = \left(\vec{e}_1 \ldots \vec{e}_{d'}\right)$$

3.) The lower-dimensional embedding of point $\vec{x}_i$ corresponds to the i-th row of matrix E.

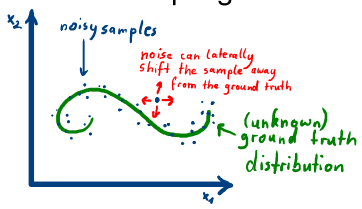
Special Topic: Noise in ISOMAP and Laplacian Eigenmaps

This is a recap of our recent Raccoon session on that topic

The question is: why is ISOMAP more sensitive to noise than Laplacian Eigenmaps?

Initial clarification: "noise" refers to some random re-location of a sample point.

For example, consider sampling from a 2-D function: "noise" subsumes deviations between observations and ground truths, e.g., a shift



To understand why ISOMAP is by tendency more sensitive to noise, let us start with a side-by-side comparison of ISOMAP and Laplacian Eigenmaps:

ISOMAP:
1) Define local neighborhood
2) Calculate distance matrix as shortest paths between all pairs of observations
3) Calculate MDS, i.e., a PCA-like projection that preserves longest distances

Laplacian Eigenmaps:
1) Define local neighborhood
2) Setup Graph Laplacian, i.e., essentially an affinity matrix from each local neighborhood
3) Perform a projection with minimum distortion, i.e., that preserves local neighborhoods

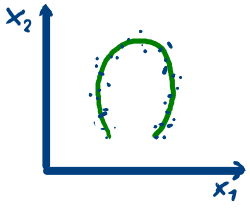
The goal is to understand that step 2) of ISOMAP can collapse the manifold if we are unlucky

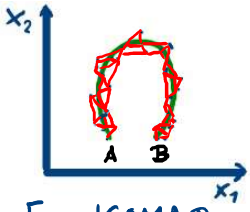
More specifically, the issue with ISOMAP is that

calculating shortest paths in conjunction with a projection that preserves long distances

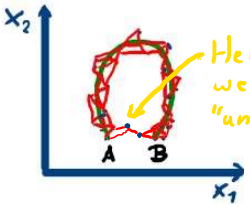
is a potentially unstable combination of tools.

Consider this example manifold:





For ISOMAP with low noise, the distance between A and B is long
⇒ Manifold is correctly projected onto a line



However, here we have to noisy samples that create a "bridge" between A and B. The distance between A and B becomes short, hence the projection will place these points closely together

Very few noisy samples can create a shortcut in the shortest paths. In our case, only 2 samples shortcut the distance from A to B.

Since ISOMAP subsequently projects the points along their longest distances, the projection for A and B (and their surroundings) will be close together - a failure case of the method

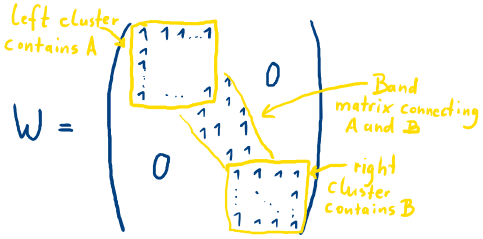
Generally speaking for robust methods: small changes in the input (e.g., due to noise) should not lead to large changes in the output

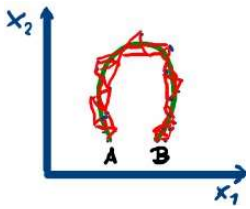
Why are Laplacian Eigenmaps more Robust in this Case?

Answer: the two noise samples only change the Graph Laplacian locally, and the optimization also aims to only keep local neighborhoods together

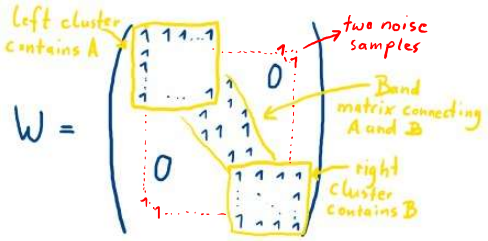
To see this, let us have a look at the affinity matrix.

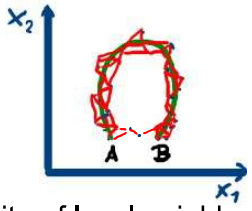
Without loss of generality, we assume that
a) the samples are numbered according to their ordering on the ground truth manifold
b) affinities are binary, i.e., either 0 (no neighbors) or 1 (neighbors)





With noise:





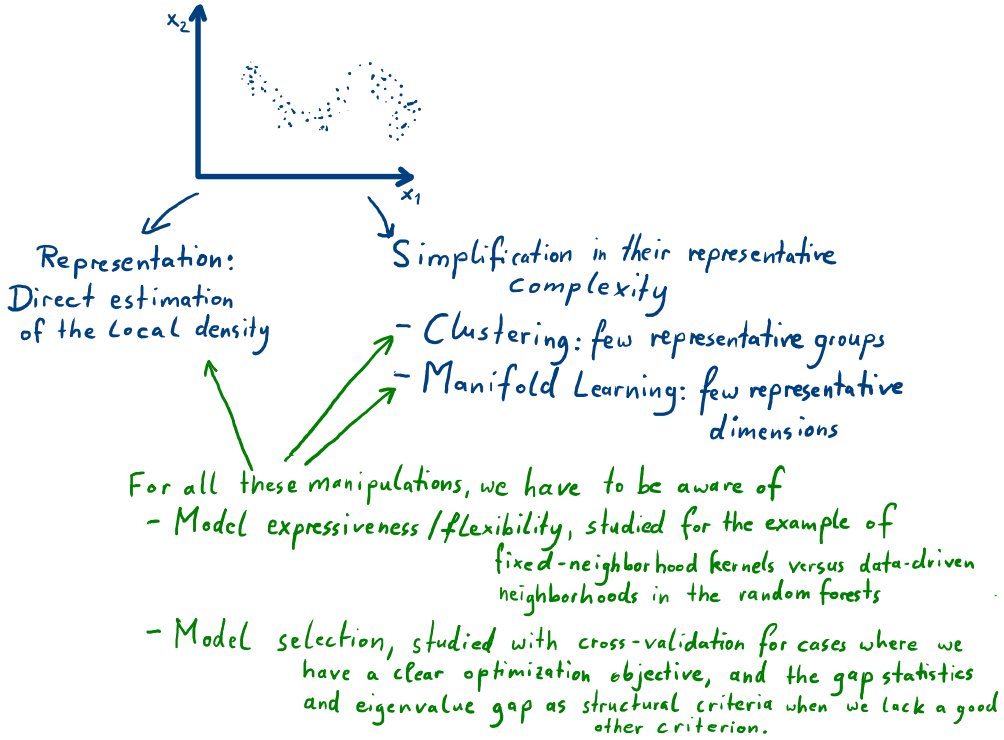
The noise samples connect A and B, but the big majority of local neighborhoods is still formed by the much thicker band matrix in the center => the overall projection is not really changed.

Recap on Sections 1 & 2, and Remarks on the Exam

Overarching topic of Sections 1 and 2:

Representations and Simplification of Feature Spaces

Starting point is always a distribution of samples in a d-dimensional space:



Several cross-links are important to internalize when studying the material - this is what we practice in the raccoon sessions:

1) We can oftentimes "float" between applications. For example:
   a) if you have one density estimator per class of labeled training data, you can directly use these represenations for classification.
   The opposite direction might also work to some extend, but certainly not as good.
   b) you can do regression on kernels, but also conversely define an "equivalent kernel" for a locally operating regressor.
   c) the Random Forest-based space partitioning can be used for several local operations in feature space: classification, regression, density estimation, manifold learning

2) Different optimization criteria or algorithm variants can lead to different results. For example:
   a) Mean shift clusters and k-means clusters can have a fundamentally different shape
   b) as a consequence, the gap statistics is a better match for k-means clusters than for mean shift clusters
   c) PCA/MDS/ISOMAP projections preserve long distances, while LE preserves neighborhoods. The latter makes non-linear projections more robust

3) Working in an applied field, we have to keep an eye on the computational complexity / feasibility of our algorithms. For example:
   a) memory-based methods such as the Parzen window estimator scale poorly to higher-dimensional spaces
   b) forest pruning is does not only help with bias/variance, but also to reduce memory requirements
   c) sampling from a Density Forest can be done in different ways. However, the Monte-Carlo approach from the lecture slides is considerably more efficient in high-dimensional spaces than interpolation of the density and estimation via the CDF

What Type of Exam Questions Can You Expect?

The exam will contain questions at three levels of cognition:

1) Reproduction Questions

Examples:
a) Please write down the objective function for mean shift
b) State the processing steps of the ISOMAP method
c) Name 3 possibilities for randomizing the result of a Random Forest

2) Explanation Questions:

Examples:
a) How can a Random Forest achieve a smooth classification boundary?
b) What complicates working with high-dimensional feature spaces?

3) Comparison / Analysis Questions:

Examples:
a) Can you tell whether the kernel density estimate in Fig. [X] originates from a box kernel or a Gaussian kernel? Explain your answer!
b) Sketch a density where the Laplacian Eigenmaps with kernel-based affinities might experience difficulties, but the Manifold Forest could still work

As always in exam situations, it pays to think about short/concise replies already during the preparation

Hints on the Study Material

As indicated in the first lecture, there are two sources for the learning materials online:

a) PA 2020 class material
This is the reference for the exam, consisting of relatively compact videos/slides in Sec. A, programming exercises in Sec. B (-> no coding in the exam, though!), and in-depth discussions in Sec. C.

b) PA 2018 class material
This is a recording of a classical blackboard lecture. Content overlaps over large portions, but there are also slight differences, hence please check for differences.
In studOn Sec. A is a PDF "00 Mapping of PA 2018 videos to PA 2020 contents" to assist with the mapping between both lectures.

The two major differences between PA 2018 and PA 2020 are:
a) PA 2020 more often references the book by Bishop than HTF
b) Connections between topics are more clearly discussed in PA 2020

As a general remark for the exam preparation:

For many learner types, it is helpful to join a learning group. However, in this year, distancing rules make this more complicated if you do not already have a group.

Feel free to use the forum to search for companions. If you have another idea with which infrastructure we can support the search for learning partners, please let us know.

Teaser to Probabilistic Graphical Models

Literature reference: Introductory pages of Sec. 8 and 8.1 in Bishop (just the first 4 pages, i.e., without Subsec. 8.1.1 until 8.1.4)

This third part of Pattern Analysis focuses on a important family of data modelling approaches, namely probabilistic graphical models.

Let us decompose this term in two parts:
1) A probabilistic model aims to express the properties of our data in a probabilistic representation

We usually start with a large joint PDF that captures all unknowns in random variables

$$p(\vec{x}_1, \vec{x}_2, \dots, \vec{x}_K)$$

This PDF is a bit "unwieldy", so let us factorize it via the product rule of probabilities to better understand the dependencies:

$$p(\vec{x}_1, \dots, \vec{x}_K) = p(\vec{x}_K | \vec{x}_1, \dots, \vec{x}_{K-1}) \cdot \dots \cdot p(\vec{x}_2 | \vec{x}_1) \cdot p(\vec{x}_1)$$

Note that without any additional knowledge or assumptions, the order of factorization is arbitrary, so any other order of variables is also a valid factorization, for example

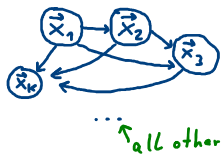
$$p(\vec{x}_1, \dots, \vec{x}_K) = p(\vec{x}_3 | \vec{x}_1, \vec{x}_2, \vec{x}_4, \dots, x_{K-1}) \cdot \dots \cdot p(\vec{x}_1 | \vec{x}_2) \cdot p(\vec{x}_2)$$

2) The "graphical" part aims to make these properties better understandable through visualization in a graph of probabilistic relationships. The graph can be directed or indirected.

- Directed edges (the general class is called Bayesian Networks) express conditional probabilities.
  We will study Hidden Markov Models (HMMs) as a practically relevant special case
- Indirected edges express joint probabilities.
  We will study Markov Random Fields (MRFs) as a practically relevant special case

Thus, a first graphical representation of our large joint PDF looks like this:

$$p(x_1, \dots, x_K) = p(\vec{x}_K | \vec{x}_1, \dots, \vec{x}_{K-1}) \cdot \dots \cdot p(\vec{x}_2 | \vec{x}_1) \cdot p(\vec{x}_1)$$



Let us highlight a few properties of this graph:
- a directed edge expresses a conditional dependency
- $\vec{x}_1$ does not depend on anything in our factorization, so there are no incoming edges
- $\vec{x}_2$ depends on $\vec{x}_1$, and it conditions all other variables, so there are one incoming and $K-2$ outgoing edges
- $\vec{x}_K$ depends on all other variables, hence it has $K-1$ incoming edges

Drawing conclusions from the relation of these variables is called inference.
Inference on a fully connected graph is infeasible for relevant problem sizes (exponential complexity).
If some variables are independent, the graph becomes sparser, and the problem becomes tractable.

In this Chapter 3 of the lecture, we will use a theoretically founded way to
1. use variable independence to obtain tractable probabilistic models,
2. but still retain a sufficient amount of dependencies to express relationships between samples.
   We will see that these models are better suited for some tasks than our tools from Chapters 1 and 2

Outline of Chapter 3 of the lecture:
1. Hidden Markov Models for inference on sequences (speech, weather forecasts, stock market, ...)
2. A closer look at independence assumptions
3. Markov Random Fields for inference on joint distributions (denoising, segmentation, depth est., ...)

Hidden Markov Models (HMMs)

Literature references:

this paper is in studOn. This Section also uses Rabiner's notation

1. Recommended: Sec. II and III of Rabiner: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, Proceedings of the IEEE, vol. 77, no. 2, Feb. 1989, pp. 257-286.
2. Not recommended (better integration, but difficult, with many backward references): Bishop Sec. 13.2

HMMs are
- generative probabilistic models, i.e., we can also sample from these models.
- best known for use on sequential data, most notably temporal data (e.g., speech, weather changes)

The model can be thought of as a probabilistic state machine. It maps observation sequences to states (called "hidden states", because a user typically only looks at input and output, but not at the internals)

Outline of this lecture:
1. Link the notations and introduce our assumptions
2. Introduction of a model sketch and the associated quantities according to Rabiner
3. Brief sketch of a speech recognizer

The algorithms that are required to work with a HMM are subject of the next lecture

HMMs as Graphical Models, and Model Assumptions

The joint PDF in the sense of the previous lecture is $p(\underbrace{\vec{x}_1,\dots,\vec{x}_N}_{\text{observations}},\underbrace{\vec{y}_1,\dots,\vec{y}_N}_{\text{hidden states}}) = p(\vec{x}_1,\dots,\vec{x}_N \mid \vec{y}_1,\dots,\vec{y}_N)\cdot p(y_1,\dots,y_N)$

We assume that both $\vec{x}_1,\dots,\vec{x}_N$ and $\vec{y}_1,\dots,\vec{y}_N$ are sequences

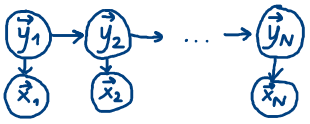
We require assumptions to further factorize the equation.

We use these assumptions:
1. Observations only depend on a single hidden state, i.e., $p(\vec{x}_1,\dots,\vec{x}_N \mid y_1,\dots,y_N) = \prod_{i=1}^{N} p(\vec{x}_i \mid \vec{y}_i)$
2. A hidden state depends only on its predecessor ("Markov assumption"), i.e., $p(y_1,\dots,y_N) = p(y_1)\cdot\prod_{i=2}^{N} p(y_i \mid y_{i-1})$

This leads to the factorization $p(x_1,\dots,x_N,y_1,\dots,y_N) = p(y_1)\cdot\prod_{i=1}^{N} p(x_i \mid y_i)\cdot\prod_{i=2}^{N} p(y_i \mid y_{i-1})$

This factorization consists of many small factors, which makes the problem tractable (details later)

The corresponding sketch as a graphical model is:



(Note that the HMM notation on this slide is somewhat simplified - the "full truth" is in Bishop, just FYI)

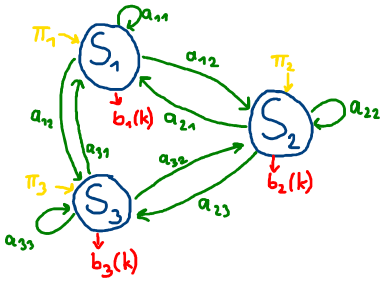
This sketch emphasizes the sequence-modelling nature of HMMs.

Rabiner uses a different approach, which emphasizes that a HMM can also be seen as a state machine, and he uses a slightly different notation

Both are introduced on the next slide

Considering a HMM as a State Machine, and Rabiner's Notation

This visualization resembles a state machine, as seen in automata theory (if you are a computer scientist):



Notation on previous slide

S_i: states

$\pi_i = p(q_1 = S_i)$: starting probability ($\Rightarrow p(\vec{y}_1)$)

$a_{ij} = p(q_{t+1} = S_j \mid q_t = S_i)$: transition probability ($\Rightarrow p(\vec{y}_j \mid \vec{y}_i)$)

$b_i(k) = p(v_k \text{ at } t \mid q_t = S_i)$: output probability ($\Rightarrow p(\vec{x}_i \mid \vec{y}_i)$)

where

q_i: state where we are at time t / observation t

v_k: k-th symbol in our alphabet (assuming that the alphabet is discrete)

We have a set of states (blue) with associated outputs (red), and transitions between states (green). Additionally, each state has a probability to be the starting state (yellow)

Analogously to automata theory, you can match an input word to this state machine:
1. start in one of the states,
2. parse one symbol of the word per time step in that state,
3. move to the next state (note also the self-loops!) and repeat step 2.

All model parameters are PDFs, and can be represented as follows:
- the starting probabilities π_i can be represented as a vector $\vec{\pi}$, where $\sum_{i=1}^{\#states} \pi_i = 1$
- the transition probabilities a_{ij} can be collected in a state transition matrix $A = [a_{ij}]$, where $\sum_{j=1}^{\#states} a_{ij} = 1$
- the output probabilities $b_i(k)$ can also be collected in a matrix $B = [b_i(k)]$, where $\sum_{k=1}^{|alphabet|} b_i(k) = 1$

Example: Outline of a Speech Recognition System

The task of speech recognition can be decomposed into a set of hierarchical subtasks:
1. Recognize individual phonemes (short sounds)
2. Recognize individual words consisting of phonemes
3. Recognize individual sentences consisting of words

This can be realized by stacking a phoneme classifier and two levels of HMMs:

Sentences: $HMM_S(1)$ "directions to (...)", $HMM_S(2)$ "buy ticket to(..)", ...

Words: $HMM_W(1)$ "directions", $HMM_W(2)$ "ticket", $HMM_W(3)$ "buy", ...

Phonemes: Phoneme(1): /a/, Phoneme(2): /o/, Phoneme(3): /t/, ...

1. A "symbol" is a short part of the recording (~25ms), which is mapped to a phoneme "alphabet" by calculating its distance to each phoneme and normalizing the distances to form a PDF.

2. Create one HMM per word that we would like to recognize (e.g., "directions", "ticket", "buy"). Return for each HMM the probability that the sequence of phonemes describes the HMM's word. Renormalize these probabilities to a PDF.

3. Create one HMM per sentence that shall be recognized (e.g., "give me directions to Frankfurt", "buy a ticket to Frankfurt"). Feed to each HMM the sequence of word PDFs, select the sentence with the highest probability.

Since the individual HMMs are fully probabilistic models, their composition is also fully probabilistic.

Three Algorithms for HMMs

Literature references:

this paper is in studOn. This Section also uses Rabiner's notation

1. Recommended: Sec. II and III of Rabiner: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, Proceedings of the IEEE, vol. 77, no. 2, Feb. 1989, pp. 257-286.
2. Not recommended (better integration, but difficult, with many backward references): Bishop Sec. 13.2

Three Algorithms are required for working with a HMM with parameters $\lambda = (A, B, \vec{\pi})$

1. Matching algorithm: calculates $p(\langle o_1, \dots, o_N \rangle | \lambda)$, i.e., the probability that an input sequence matches the HMM via forward or backward algorithm

2. Most likely state sequence: calculates $\underset{\langle q_1, \dots, q_N \rangle}{\operatorname{argmax}} \; p(\langle o_1, \dots, o_N \rangle, \langle q_1, \dots, q_N \rangle | \lambda)$, i.e., the most likely state sequence for an input sequence via Viterbi algorithm

3. HMM training: calculates $\lambda = (A, B, \vec{\pi})$ from example data via Baum-Welch (Expectation-Maximization)

Note that we explicitly include the model parameters λ in the expressions in 1) and 2) to emphasize that these tasks are computed on a trained model

Task 1: Matching a Word to a HMM

$p(\langle o_1, \dots, o_N \rangle)$ can be calculated by marginalization over all state sequences from the full model, i.e.,

$$p(\langle o_1, \dots, o_N \rangle) = \sum_{\langle q_1, \dots, q_N \rangle} p(\langle o_1, \dots, o_N \rangle, \langle q_1, \dots, q_N \rangle) = \sum_{\langle q_1, \dots, q_N \rangle} p(\langle o_1, \dots, o_N \rangle | \langle q_1, \dots, q_N \rangle) \cdot p(\langle q_1, \dots, q_N \rangle)$$

This marginalization can be directly calculated via summation over all state sequences

$$p(\langle o_1, \dots, o_N \rangle) = \sum_{q_1=1}^{M} \sum_{q_2=1}^{M} \dots \sum_{q_N=1}^{M} \pi_{q_1} \cdot b_{q_1}(o_1) \cdot a_{q_1 q_2} \cdot b_{q_2}(o_2) \cdot a_{q_2 q_3} \cdot \dots \cdot a_{q_{N-1} q_N} \cdot b_{q_N}(o_N)$$

N sums over M = #states many states

start here, parse the first symbol, go to the next state, parse the next symbol, ...

Unfortunately, this expression is intractable for realistically sized inputs, due to $O(N^M)$ evaluations

However, a dynamic programming solution admits a more efficient approach.
compute for each time steps all possible states, then proceed to the next step:

$$p(\langle o_1, \dots, o_N \rangle) = \sum_{q_1=1}^{M} \pi_{q_1} \cdot b_{q_1}(o_1) \cdot \sum_{q_2=1}^{M} a_{q_1 q_2} \cdot b_{q_2}(o_2) \cdot \sum_{q_3=1}^{M} a_{q_2 q_3} \cdots \cdot \sum_{q_N=1}^{M} a_{q_{N-1} q_N} \cdot b_{q_N}(o_N)$$

This approach requires $O(M^2)$ evaluations from one state to the next, for N states, totalling $O(N \cdot M^2)$
(note that this efficient transformation is enabled from our independence assumptions on the HMM!)

The forward algorithm applies this trick from start to end, the backward algorithm from end to start:

Forward alg.: 1) $t = 1$: $\alpha_1(q_1) = \pi_{q_1} \cdot b_{q_1}(o_1) \quad \forall 1 \le q_1 \le M$

2) $1 < t \le N$: $\alpha_t(q_t) = \left(\sum_{q_{t-1}=1}^{M} \alpha_{t-1}(q_{t-1}) \cdot a_{q_{t-1} q_t} \right) \cdot b_{q_t}(o_t) \quad \forall 1 \le q_t \le M$

3) $t = N$: $p(\langle o_1, \dots, o_N \rangle) = \sum_{q_N=1}^{M} \alpha_N(q_N)$

backward alg.: 1) $t = N$: $\beta_N(q_N) = 1 \quad \forall 1 \le q_N \le M$

2) $1 \le t < N$: $\beta_t(q_t) = \sum_{q_{t+1}=1}^{M} a_{q_t q_{t+1}} \cdot b_{q_{t+1}}(o_{t+1}) \cdot \beta_{t+1}(q_{t+1}) \quad \forall 1 \le q_t \le M$

Task 2: Finding the Most Likely State Sequence

Goal: find the most likely state sequence $\langle q_1,\dots,q_N\rangle^* = \underset{\langle q_1,\dots,q_N\rangle}{\arg\max}\, p(\langle o_1,\dots,o_N\rangle, \langle q_1,\dots,q_N\rangle|\lambda)$ for a fixed input

It is important that the best sequence does not necessarily include the single most likely state at time t. Instead, the sequence is optimized as a whole, selecting the most likely path through the HMM.

The algorithm is almost identical to the forward algorithm, with the following induction step at its core:

$$\delta_t(i) = \underset{\langle q_1,\dots,q_t\rangle}{\arg\max}\; p(\langle o_1,\dots,o_t\rangle, \langle q_1,\dots,q_t=i\rangle|\lambda) \quad \forall t \le N$$

expressing that $\delta_t(i)$ is the most likely subpath for times $1 \le t \le N$ that ends in state q_i.

Again, we consider all possible end states simultaneously, and progress step by step in time.

Compared to the forward algorithm, we have one additional variable $\psi_t(i)$ that tracks the predecessor state, in order to reconstruct the path in the last algorithm step.

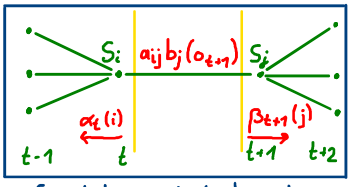
Viterbi algorithm:

1) $t=1$: $\delta_1(i) = \pi_i\, b_i(o_1) \quad \forall 1 \le q_1 \le M$
   $\psi_1(i) = 0$
2) $2 \le t \le N$: $\delta_t(i) = \max_{1\le i\le M} (\delta_{t-1}(i)\cdot a_{ij})\cdot b_j(o_t) \quad \forall 1 \le j \le M$
   $\psi_t(i) = \underset{1\le i\le M}{\arg\max}\, \delta_{t-1}(i)\cdot a_{ij}$
3) $p^* = \max_{1\le i\le M} \delta_N(i)$, $q_N^* = \underset{1\le i\le M}{\arg\max}\, \delta_N(i)$
4) Path reconstruction: $q_t^* = \psi_{t+1}^*(q_{t+1}^*) \quad \forall 1 \le t \le M$

Task 3: HMM Training via the Baum-Welch Formulae

Expectation-Maximization algorithm that (analogously to GMM fitting) iteratively assigns "responsibilities" and maximizes the model parameters according to their respective responsibility.

Responsibilities are calculated in auxiliary variables:



ξ isolates a single transition at time t, since $\alpha_t(i)$ sums over all paths leading to S_i earlier than t, and $\beta_{t+1}(j)$ sums over all paths that lead away from S_j later than $t+1$.

Probability for transition $S_i \to S_j$ at time t:

$$\xi_t(i,j) = p(q_t = S_i, q_{t+1} = S_j) = \frac{\alpha_t(i)\cdot a_{ij}\cdot b_{t+1}(o_{t+1})\cdot \beta_{t+1}(j)}{\sum_{i=1}^{M}\sum_{j=1}^{M} \alpha_t(i)\cdot a_{ij}\cdot b_j(o_{t+1})\cdot \beta_{t+1}(j)}$$

Probability of being in state S_i at time t: $\gamma_t(i) = \sum_{j=1}^{M} \xi_t(i,j)$

Furthermore, we can derive
- the expected # of transitions $S_i \to S_j$ at all times: $\sum_{t=1}^{N} \xi_t(i,j)$
- the expected # of times S_i is visited / expected # of transitions from state S_i: $\sum_{t=1}^{N} \gamma_t(i)$

Maximization:
- $\bar{\pi}_i = \gamma_1(i)$ Expected # of times in S_i at time $t=1$
- $\bar{a}_{ij} = \frac{\sum_{t=1}^{N-1} \xi_t(i,j)}{\sum_{t=1}^{N-1} \gamma_t(i)}$ Expected # of transitions $S_i \to S_j$ / Expected # of times in state S_i
- $\bar{o}_j(k) = \frac{\sum_{t=1}^{N} \gamma_t(j)\cdot \mathbb{I}(o_t = v_k)}{\sum_{t=1}^{N} \gamma_t(j)}$ Expected # of times in S_j and observing v_k / Expected # of times in S_j

From Directed to Undirected Models

Literature references:
- Bishop Sec. 8.3.1 "MRFs" -> "Conditional independence properties"
- Bishop Sec. 8.2 "Conditional Independence": strictly optional (many more details on independence)
- Bishop Sec. 8.3.4: also strictly optional: relations of directed and undirected models

Recall that we aim to express all unknowns in a large joint distribution $p(\vec{x}_1, \dots, \vec{x}_N, \vec{z}_1, \dots \vec{z}_N)$

Two properties are required for working with such a joint distribution:
- Factorization via the rules of probability ← always possible
- Sufficient independence between the variables to reduce the number and size of the factors, which makes operations on the model tractable ← needs to be appropriately designed

For a HMM, we have a sequence of latent (hidden) variables with associated observations

We can read from this model that
1) Each hidden variable y_i only depends on its immediate predecessor, each observation x_i on its state y_i
2) If we do not know the state of any hidden variable, the chain of conditional dependencies makes all hidden variables dependent of each other
3) However, if we know the state of y_i (i.e., if we know at time i in which state we are), we effectively break the chain into two parts: then, states y_{i-1} and y_{i+1} are conditionally independent given y_i

Recall that these properties allow for the factorization

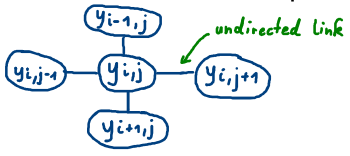
$$p(\vec{x}_1, \dots, \vec{x}_N, \vec{y}_1, \dots, \vec{y}_N) = p(\vec{y}_1) \cdot \prod_{i=2}^{N} p(y_i | y_{i-1}) \cdot \prod_{i=1}^{N} p(\vec{x}_i | \vec{y}_i)$$ (Eqn. Bishop (13.6), p. 610)

Models of directed graphs are great for sequences, HMMs in particular.

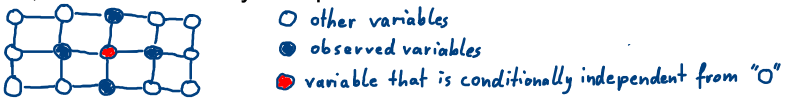
Question: isn't it possible to serialize any task, and to use a HMM on that serialization?
Answer: NO. ← doomed to fail ☹
- From a theoretical perspective, HMMs model only a subset of all distributions
- From a practical perspective, even if we are able to "cram" our data into a sequence, maybe by extending the model to a slightly higher order (allowing 2 or 3 predecessor states), such a representation is likely not the most effective choice. ← expensive, hard to train, difficult to maintain

Example: Image pixels reside on a 2-D grid, and common image processing operations such as segmentation or denoising do not impose a direction on the grid

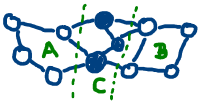
Such relationships are oftentimes better captured by an undirected model:



The equivalent independence assumption on undirected models is the "Markov Blanket":
- if all neighbors of a variable are observed, it is conditionally independent from all other variables

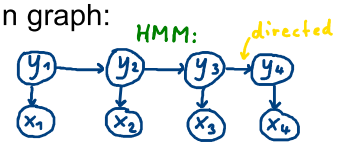


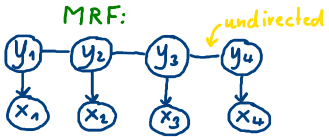
- this also implies (confer observation 3 on the previous slide) that two sets A and B of variables can be separated by a set of observations C



Probabilistic Interpretation of Undirected Edges

Let us directly compare directed and undirected edges on the same underlying graph structure, namely a chain graph:





y_i: hidden variables
x_i: observations

The difference shows in the dependencies of the hidden variables:

HMM continued:

Dependencies for y_2:

$p(y_2 | y_1, \dots y_4) = p(y_2 | y_1)$

$p(y_3 | y_1, \dots, y_4) = p(y_3 | y_2)$

(the third term is not important for the directed/undirected discussion, but for the sake of completeness:
$p(x_2 | y_1, \dots, y_4) = p(x_2 | y_2)$)

MRF continued:

Dependencies for y_2:

$p(y_2 | y_1, \dots, y_4) = p(y_2 | y_1, y_3) \overset{!}{=} p(y_2 | y_1) \cdot p(y_2 | y_3)$

$p(y_1 | y_2, \dots, y_4) = p(y_1 | y_2)$

note the symmetry!

$p(y_3 | y_1, \dots, y_4) = p(y_3 | y_2, y_4) \overset{!}{=} p(y_3 | y_2) \cdot p(y_3 | y_4)$

(and as for the HMM case also
$p(x_2 | y_1, \dots, y_4) = p(x_2 | y_2)$)

Thus, undirected edges introduce two conditional dependencies between neighboring variables.

The chain graph indicates that it is possible to also use undirected graphs on sequential data.

yay! MRFs for speech!

On more complicated graph structures, these additional dependencies make the training intractable. However, we will see later that inference, i.e., finding specific label assignments, is still well feasible.

On data without a clear sequential structure (e.g., images), it may make sense to choose symmetric distributions $p(y_i | y_j)$ and $p(y_j | y_i)$, i.e., $p(y_i = \ell_k | y_j = \ell_l) = p(y_j = \ell_k | y_i = \ell_l)$ for discrete assignments ("labels") ℓ_k, ℓ_l.

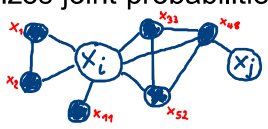
Markov Random Fields

Literature Reference: Bishop Sec. 8.3.2

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

$$p(\vec{x}_i, \vec{x}_j \mid \vec{x}_{k\neq i,j}) = p(\vec{x}_i \mid \vec{x}_{k\neq i,j}) \cdot p(\vec{x}_j \mid \vec{x}_{k\neq i,j})$$

↑ Equality implies conditional independence



x_i, x_j: random variables

: "observed" variables form Markov Blanket

The expression for $\vec{x}_i$ can be further simplified:
it only depends on its neighbors $\vec{x}_1, \vec{x}_2, \vec{x}_{44}, \vec{x}_{33}, \vec{x}_{48}, \vec{x}_{52}$: $p(\vec{x}_i \mid \vec{x}_{k\neq i,j}) = p(\vec{x}_i \mid \vec{x}_1, \vec{x}_2, \vec{x}_{44}, \vec{x}_{33}, \vec{x}_{48}, \vec{x}_{52})$

The dependencies can be further decomposed by considering the maximal cliques to which $\vec{x}_i$ belongs.
Maximal cliques are the largest fully connected subgraphs that contain $\vec{x}_i$:

$$\{x_i, \vec{x}_1, \vec{x}_2\}, \{x_i, x_{44}\}, \{x_i, \vec{x}_{33}, \vec{x}_{48}, \vec{x}_{52}\}$$

$$\Rightarrow p(\vec{x}_i \mid x_{k\neq i,j}) = p(\vec{x}_i \mid \vec{x}_1, \vec{x}_2) \cdot p(\vec{x}_i \mid \vec{x}_{44}) \cdot p(\vec{x}_i \mid \vec{x}_{33}, \vec{x}_{48}, \vec{x}_{52})$$

(intuitively, a maximal clique is something like the smallest irreducible component on the graph)

In most applications, it is easier to find suitable "potential functions" instead of conditional distributions.
Hence, the Markov Random Field is typically written in terms of such potential functions ("factors") $\varphi_c(\vec{x}_c)$:

$$p(\vec{x}_1, \dots, \vec{x}_N) = \frac{1}{Z} \prod_c \varphi_c(\vec{x}_c) \quad \text{(Bishop Eqn. (8.39))}$$

where $Z = \sum_{\substack{\text{all value} \\ \text{combinations} \\ \text{for } \vec{x}_1, \dots, \vec{x}_N}} \prod_c \varphi_c(\vec{x}_c)$ denotes the "partition function" for normalization to a PDF (intractable!).

and
- $\vec{x}_1, \dots, \vec{x}$: random variables with discrete states (e.g. $\{0,1\}$ for binary vars)
- C: set of all maximal cliques in the graphical model
- $\varphi_C(\vec{x}_C)$: potential function over a clique with variables $\vec{x}_C$. In our example, we would have three functions $\varphi_1(\vec{x}_1, \vec{x}_2)$, $\varphi_2(\vec{x}_{44})$, and $\varphi_3(\vec{x}_{33}, \vec{x}_{48}, \vec{x}_{52})$

(Note: replace the sum by an integral to work on continuous variables)

Potential functions / factors φ_c are more flexible than conditional distributions, since they can encode an arbitrary coupling between the clique variables. ← note from previous lecture: they are often chosen to be symmetric w.r.t. the variable assignments.

Our formulation of a MRF as $p(\vec{x}_1, \dots, \vec{x}_N) = \frac{1}{Z} \prod_c \varphi_c(\vec{x}_c)$ is identical to a so-called Gibbs Random Field if and only if the potential functions are strictly positive (the Hammersley-Clifford Theorem proofs this).

To achieve this, the potential functions are commonly chosen as Boltzmann distributions, i.e., affinities with exponentials of energy functions $\varphi_C = e^{-E(x_C)}$ (Bishop Eqn. (8.41))

The partition function Z normalizes $p(\vec{x}_1, \dots, \vec{x}_N)$ to a probability distribution.

However, Z sums over all combinations of values, which causes exponential computational cost.

As a consequence, learning the MRF potential functions and/or the MRF graph structure from training data is prohibitively expensive on general graphs. ← i.e., no equivalent to the "Baum-Welch"

However, inference is tractable (Z is just a constant), i.e., we can find marginal distributions of variables.
To this end, we will
- introduce a simple, locally optimal Gibbs sampling method in the next lecture
- learn about a globally optimal inference algorithm that uses graph cuts in two lectures

Remarks:
MRFs can model distributions that are impossible to express in a HMM and vice versa
(curious about this? Bishop shows to minimal examples on page 393, Fig. 8.35 and Fig. 8.36).

There exists a huge set of applications of MRFs, particularly for labelling and segmentation tasks, i.e., to assign to each variable a discrete label.

Concrete MRF Example and Inference via Gibbs Sampling

Literature reference: Bishop Sec. 8.3.3
(w/ flipped variables: we use y_i for hidden variables, x_i for observations)

Given is an binary black-and-white image with noise:

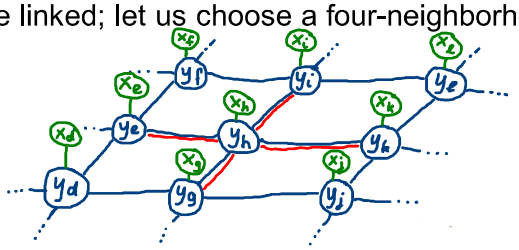
Without loss of generality, we represent yellow as -1 and blue as 1.

The input pixels are indexed as observations $x_1, \dots, x_N$.
The denoised image shall be the label assignments to the hidden variables $y_1, \dots, y_N$.

The MRF graph hence consists of one node per input pixel, and one node per output pixel.

We use a similar connection structure as for the HMMs, by assuming that
- neighboring observations are independent
- neighboring hidden variables are linked; let us choose a four-neighborhood of nearby pixels

This leads to the graphical model



Note that the maximal cliques in this connection structure have size 2, i.e., there are only pairs of nodes

We need to define the relationship between
a) hidden variables and observations
b) hidden variables to neighboring hidden variables

From an application perspective, these relationships shall serve two goals:

a) Hidden variables should be close to observations:
The resulting image should not be arbitrary. It should resemble in most parts the input image.

b) Hidden variables should be similar to their neighboring hidden variables:
The resulting denoised image should be smoother. We encourage to have only few changes in the intensities.

Hence, we can define the clique potentials as
a) $e^{-E(x_i, y_i)}$ with the energy term $E(x_i, y_i) = -\eta x_i y_i$.
Here, η is a positive, constant weighting factor. Note that the energy is lower for equal values $x_i = y_i = \pm 1$ than for different values. Hence, the potential $e^{-E(x_i, y_i)}$ is larger for equal values.

b) Analogously, we can set $e^{-E(y_i, y_j)}$ with energy term $E(y_i, y_j) = -\beta y_i y_j$, which also leads to a higher potential if $y_i = y_j = \pm 1$.

So, we are closer to the preferred solution if the potentials are larger.

The mathematical background for this is on the next slide.

Objective Function for MRF Inference

Consider a joint distribution of observations and hidden variables:

$$p(x_1, \dots, x_N, y_1, \dots, y_N) = p(x_1, \dots, x_N \mid y_1, \dots, y_N) \cdot p(y_1, \dots, y_N)$$

We seek to find the assignment to variables that maximizes the joint probability, i.e.,

$$\underset{y_1, \dots, y_N}{\operatorname{argmax}} \; p(x_1, \dots, x_N \mid y_1, \dots, y_N) \cdot p(y_1, \dots, y_N)$$

We make the following 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

$$\underset{y_1, \dots, y_N}{\operatorname{argmax}} \prod_{i=1}^{N} p(x_i \mid y_i) \cdot \prod_{i=1}^{N} \prod_{y_j \in \mathcal{N}(y_i)} p(y_i \mid y_j)$$

$\mathcal{N}(y_i)$: set of neighboring hidden variables of y_i

The Hammersley-Clifford Theorem states that the solution to this expression is equivalent to the solution of a Gibbs Random Field with strictly positive potentials:

$$\underset{y_1, \dots, y_N}{\operatorname{argmax}} \prod_{i=1}^{N} p(x_i \mid y_i) \cdot \prod_{i=1}^{N} \prod_{y_j \in \mathcal{N}(y_i)} p(y_i \mid y_j) = \underset{y_1, \dots, y_N}{\operatorname{argmax}} \frac{1}{Z} e^{-\sum E(x_i, y_i) - \sum E(y_i, y_j) \left(- \sum E(y_i)\right)}$$

$\frac{1}{Z}$: constant, can be ignored for max. $\sum E(y_i)$: unused in our task

Since this expression is maximized: larger potentials are closer to the solution

Remarks:
- Potentials with energy terms $E(x_i, y_i)$ are also called "unary potentials" (w/ only 1 hidden variable)
- Potentials with energy terms $E(y_i, y_j)$ are also called "pairwise potentials" (w/ 2 hidden variables)
- $E(y_i)$ could be used to bias individual hidden variables to one value, we do not use it in our example

Locally Optimal MRF Inference via Gibbs Sampling

Gibbs sampling is a straightforward way of finding a labeling $y_1, \dots, y_N$:

While (infinity):
- Select a node y_i (randomly chosen, or following a pre-defined path)
- Fix the assignments to all its neighbors
- Assign a new label to y_i if this yields a lower energy

In practice, the Gibbs sampler is either run for a pre-defined number of steps or until the assignments do not change for a preset number of rounds.

Note that the Gibbs sampler only finds a locally optimal solution.

An example result is shown in Bishop page 388, Fig. 8.30 (bottom left).

MRF Inference via Graph Cuts

Literature reference: Kolmogorov, Zabih: "What Energy Functions Can Be Minimized via Graph Cuts?" IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 2, Feb. 2004.

Inference task: find labeling $y_1^*, \dots, y_N^* = \underset{y_1,\dots,y_N}{\operatorname{argmax}} \frac{1}{Z} e^{-\sum E(x_i,y_i) - \sum E(y_i,y_j)}$

This is equivalent to $y_1^*, \dots, y_N^* = \underset{y_1,\dots,y_N}{\operatorname{argmin}} \sum_{i=1}^{N} E(x_i,y_i) + \sum_{i,j=1}^{N} E(y_i,y_j)$

Thus, it suffices to find a labeling that minimizes the sums of energy functions.

Idea of this algorithm: construct a specialized graph from the energy functions, such that the minimum cut on the graph also minimizes the energy functions.

Constraints:
- requires binary labels
- maximal clique size is 2
- pairwise energies must satisfy "submodularity condition": $E(0,0) + E(1,1) \leq E(0,1) + E(1,0)$

↑ for pairwise energy terms, assuming w.l.o.g. the label set $\{0,1\}$

Benefits:
- globally optimal labeling
- polynomial-time algorithm

Extension to multiple labels: "α expansion"
- works on more than 2 labels
- not globally optimal, but within a fixed margin
- polynomial-time algorithm

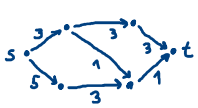
Maximum Flow in a Nutshell

Max Flow: Edge weight = network capacity per time.
Q: What is the maximum throughput from source to sink per time?

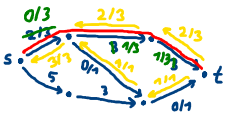
-> Standard problem in combinatorial optimization, solved by polynomial-time algorithms like Ford-Fulkerson or Preflow-Push.

Ford-Fulkerson: Iteratively search shortest path through graph (greedy!)
- max out the capacity along that path
- introduce backward edges to undo dead ends in greedy search

Example:



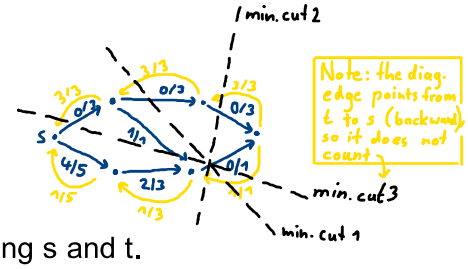
shortest path s→t: cost is 3+1+1=5
flow capacity: 1 (smallest throughput)
*: remaining capacities forward edges
*: new capacities backward edges



shortest path s→t: cost is 2+3+3=8
flow capacity: 2 (smallest throughput)

shortest path s→t: cost is 5+3+1+1+1
flow capacity: 1 (smallest throughput)

⇒ finished (there exists no other path s→t)



Min Cut problem: minimal sum of edge weights to remove for separating s and t.
The Max-Flow Min-Cut Theorem: identical solution as max flow computation.
In our example, max flow = min cut = 4. The involved edges are a subset of the maxed out edges.

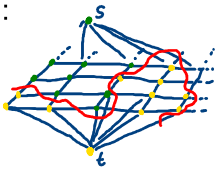
The idea of the Graph-Cut Solver is to encode the energy functions $E(x_i, y_i)$ and $E(y_i, y_j)$ in a special s-t graph, such that the minimum cut is also a minimal solution to our objective function

$$y_1^*, \dots, y_N^* = \arg\min_{y_1,\dots,y_N} \sum_{i=1}^{N} E(x_i, y_i) + \sum_{i,j=1}^{N} E(y_i, y_j)$$

The full graph is constructed as follows:

The connections among the hidden variables are in the middle. Additionally, hidden variables are connected to source s and sink t:

Identify s with label 1 (e.g., "0") and t with label 2 (e.g., "1").
A min cut through the graph separates s and t:



●: nodes with label "0"
●: nodes with label "1"
∿: Cut

After the cut, nodes on the side of s obtain label 1 ("0"), nodes on the side of t obtain label 2 ("1").

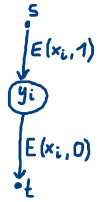
The "magic" happens in the edge weights, such that this is indeed a globally minimal solution.

We use the additivity property:
if G_1 encodes $\min(E_1)$, G_2 encodes $\min(E_2)$, then $(G_1 \cup G_2)$ encodes $\min(E_1 + E_2)$.

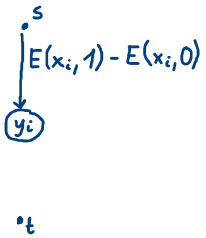
Thus: we can
- encode each potential separately in a small graph, and then
- perform the graph cut through the union of these small graphs

Encoding of Unary Energy Terms, e.g. with labels $\{0,1\}$, identifying $s=0, t=1$:

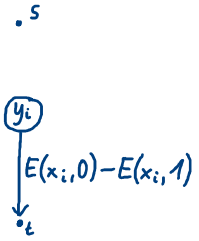
$E(x_i, y_i)$:



Shorter, equivalent graph: let w.l.o.g. $E(x_i,1) > E(x_i,0)$: omit edge to reduce computational complexity:

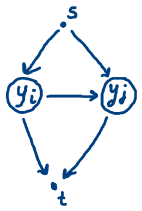


alternatively, if $E(x_i,0) > E(x_i,1)$: omit the other edge:

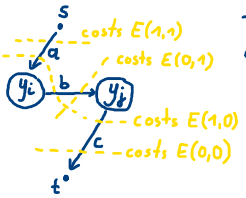


Encoding of Pairwise Energy Terms

$E(y_i, y_j)$:



Pairwise terms are more complicated than unary terms. If $E(1,0) > E(0,0)$ and $E(1,0) > E(1,1)$, the reduced graph is



The weights a, b, c are the solution to a linear system of equations:

constant offset

$$a + k = E(1,1)$$
$$c + k = E(0,0)$$
$$b + k = E(0,1)$$
$$a + c + k = E(1,0)$$

← w/o b, because b is oriented against the cut direction

These encodings are done for each energy term. The union of these small graphs sums up the weights for edges between identical nodes with identical orientations.

Generalization to non-binary labels: "Alpha expansion" algorithm (unfortunately not globally optimal)

While (solution changed)
- Select one label, denote it as alpha
- Solve binary task {alpha, all labels without alpha}
- Set this as new labeling if at least one more node has label alpha, and the total energy is lower.