Machine Learning

Marc Toussaint

July 15, 2013

This is a direct concatenation and reformatting of all lecture slides and exercises from the Machine Learning *course (summer term 2013, U Stuttgart), including a topic list to prepare for exams.*

Contents

1 Introduction

What is Machine Learning?

1) Machine Learning is an approach to understand learning by building learning systems

("synthetic" approach to a science of learning)

2) A long list of methods/algorithms for different data anlysis problems

— in sciences

— in commerce

3) A framework to develop your own learning algorithms/methods

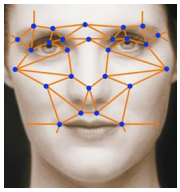
4) Machine Learning = information theory/statistics + computer science

1:1

Examples for ML applications...

1:2

Face recognition



keypoints

eigenfaces

(e.g., Viola & Jones)

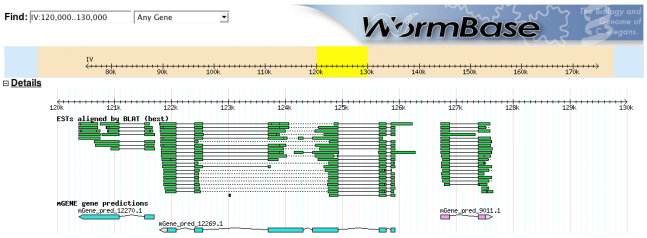
1:3

Hand-written digit recognition (US postal data)

(e.g., Yann LeCun)

1:4

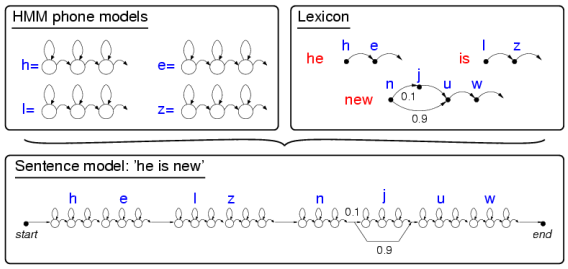
Gene annotation



(Gunnar Rätsch, Tübingen, mGene Project)

1:5

Speech recognition



(This is the basis of all commercial products)

1:6

Spam filters

	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

1:7

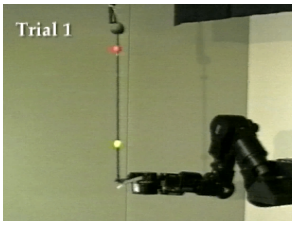
- More examples:
  - Google's (and many others') analysis of user preferences
  - Medical diagnosis
- **Machine Learning became an important technology in science as well as commerce**

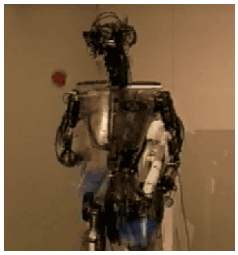
1:8

Examples of ML *for behavior*...

1:9

Learing motor skills

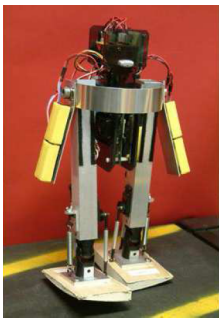




(around 2000, by Schaal, Atkeson, Vijayakumar)

1:10

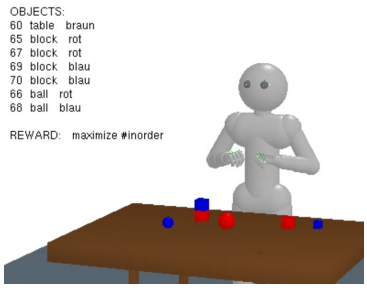
Learning to walk



(Rus Tedrake et al.)

1:11

Learning effects of actions



(Tobias Lang & M Toussaint)

1:12

Types of ML

- *Supervised* learning: learn from "labelled" data $\{(x_i, y_i)\}_{i=1}^N$

  Unsupervised learning: learn from "unlabelled" data $\{x_i\}_{i=0}^N$ only

  Semi-supervised learning: many unlabelled data, few labelled data

- *Reinforcement* learning: learn from data $\{(s_t, a_t, r_t, s_{t+1})\}$
  - learn a predictive model $(s, a) \mapsto s'$
  - learn to predict reward $(s, a) \mapsto r$
  - learn a behavior $s \mapsto a$ that maximizes reward

1:13

Organization of this lecture

- **Part 1:** *The Basis*
  - Basic regression & classification
- **Part 2:** *The Breadth of ML ideas*
  - PCA, PLS
  - Local & lazy learning
  - Combining weak learners: boosting, decision trees & stumps
  - Other loss functions & Sparse approximations: SVMs
  - Deep learning
- **Part 3:** *In Depth Topics*
  - Bayesian Learning, Bayesian Ridge/Logistic Regression
    Gaussian Processes GP classification
  - Active Learning
    Recommender Systems
- Missing:
  - Neural Networks
  - Graphical Models & structure learning
  - un-, semi-supervised learning

1:14

- Is this a theoretical or practical course?

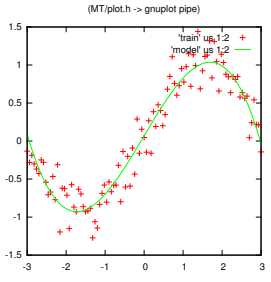
  Neither alone.

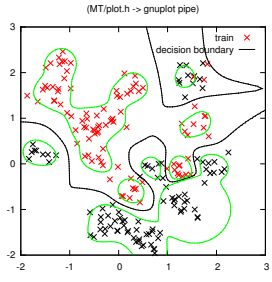
- The goal is to teach how to design good learning algorithms

  data
  ↓
  modelling [requires theory & practise]
  ↓
  algorithms [requires practise & theory]
  ↓
  testing, problem identification, restart

1:15

Basic regression & classification





- Regression: map input x to continuous value $y \in \mathbb{R}$

  Classification: map input x to one of M classes $y \in \{1, 2, .., M\}$

1:16

Basic regression & classification

- A must-know!
- High practical relevance for applications
- Focus on linear methods on non-linear features, regularization, cross-validation

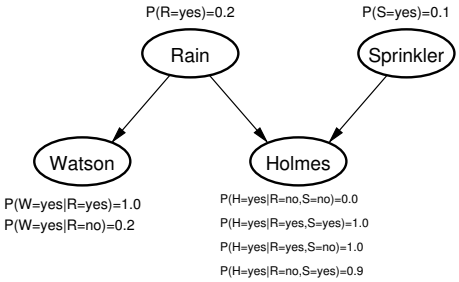
"linear|polynomial|Kernel Ridge|Lasso Regression|Classification"

- Relations to SVM, GPs, feature selection

1:17

Bayesian Modelling

- Mr. Holmes lives in Los Angeles. One morning when Holmes leaves his house, he realizes that his grass is wet. Is it due to rain, or has he forgotten to turn off his sprinkler?
- Holmes checks Watsons grass, and finds it is also wet. What does that imply on rain vs. sprinkler?



$$\iff P(H, W, S, R) = P(H|S, R)\ P(W|R)\ P(S)\ P(R)$$

1:18

Bayesian modelling

- Fundamental view on information processing and learning
- Provides general tools for formulating structured probabilistic models

  (e.g., latent variables, mixtures, hierarchical, deep models)

  → a framework for formulating novel learning algorithms
- Bayesian view on linear models + regularization

  – regularization ↔ prior, "error" ↔ likelihood

1:19

Reinforcement Learning (another course..)

[PacMan]

I Szita, A Lorincz: *Learning to Play Using Low-Complexity Rule-Based Policies: Illustrations through Ms. Pac-Man.* JAIR 2007.

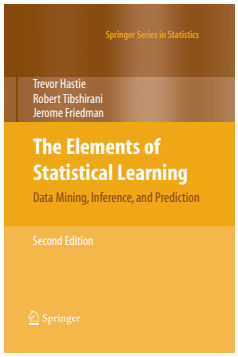
1:20

Reinforcement Learning (another course..)

- Behavior!, learning to act
- Basic RL methods (Temporal Difference, Q-learning, traces)
- Regression in RL, Bayesian methods in RL
- Applications

1:21

Books

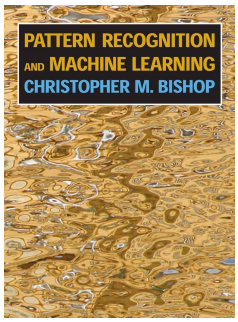


Trevor Hastie, Robert Tibshirani and Jerome Friedman: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* Springer, Second Edition, 2009.
http://www-stat.stanford.edu/~tibs/ElemStatLearn/
(recommended: read introductory chapter)

(this course will not go to the full depth in math of Hastie et al.)

1:22

Books



Bishop, C. M.: *Pattern Recognition and Machine Learning.* Springer, 2006
http://research.microsoft.com/en-us/um/people/cmbishop/prml/
(some chapters are fully online)

1:23

Organisation

- Vorlesungs-Webpage:

http://ipvs.informatik.uni-stuttgart.de/mlr/marc/teaching/13-MachineLearni

  – Slides, Übungen & Software (C++)
  – Links zu Büchern und anderen Ressourcen
- Sekretariat/Organisatorische Fragen:

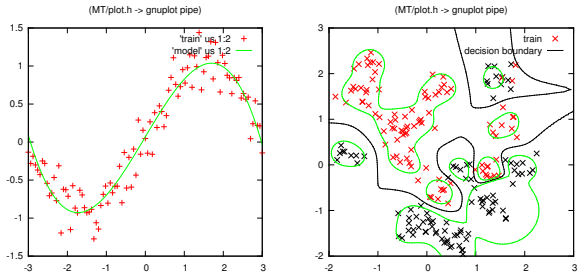
  Carola Stahl, Carola.Stahl@ipvs.uni-stuttgart.de, Raum 2.217
- 2 geplante Übungen: Dienstag 14:00-15:30 & 15:45-17:15, 0.453

- Regelung zu Übungen:
  - Bearbeitung der Übungen ist wichtig!
  - Zu Beginn jeder Übung in Liste eintragen:
    - Teilnahme
    - Welche Aufgaben wurden bearbeitet
  - Zufällige Auswahl zur Präsentation der Lösung
  - 50% bearbeitete Aufgaben notwendig für *aktive Teilnahme*

1:24

2 Regression basics

Linear regression, non-linear features (polynomial, RBFs, piece-wise), regularization, cross validation, Ridge/Lasso, kernel trick



- Are these linear models? Linear in *what*?
  - Input: No.
  - Parameters, features: Yes!

Linear Modelling is more powerful than it might seem at first!

- Linear Regression on non-linear features → very powerful (polynomials, piece-wise, spline basis, kernels)
- Regularization (Ridge, Lasso) & cross-validation for proper generalization to test data
- Gaussian Processes and SVMs are closely related (linear in kernel features, but with different optimality criteria)
- Liquid/Echo State Machines, Extreme Learning, are examples of linear modelling on many (sort of random) non-linear features
- Basic insights in model complexity (effective degrees of freedom)
- Input relevance estimation (z-score) and feature selection (Lasso)
- Linear regression → linear classification (logistic regression: outputs are likelihood ratios)

⇒ Good foundation for learning about ML

(We roughly follow Hastie, Tibshirani, Friedman: *Elements of Statistical Learning*)

Linear Regression

- Notation:
  - input vector $x \in \mathbb{R}^d$
  - output value $y \in \mathbb{R}$
  - parameters $\beta = (\beta_0, \beta_1, .., \beta_d)^\top \in \mathbb{R}^{d+1}$
  - linear model

$$f(x) = \beta_0 + \sum_{j=1}^d \beta_j x_j$$

- Given training data $D = \{(x_i, y_i)\}_{i=1}^n$ we define the *least squares* cost (or "loss")

$$L^{\text{ls}}(\beta) = \sum_{i=1}^n (y_i - f(x_i))^2$$

Optimal parameters β

- Augment input vector with a 1 in front: $\boldsymbol{x} = (1, x) = (1, x_1, .., x_d)^\top \in \mathbb{R}^{d+1}$

  $\beta = (\beta_0, \beta_1, .., \beta_d)^\top \in \mathbb{R}^{d+1}$

$$f(x) = \beta_0 + \sum_{j=1}^n \beta_j x_j = \boldsymbol{x}^\top \beta$$

- Rewrite sum of squares as:

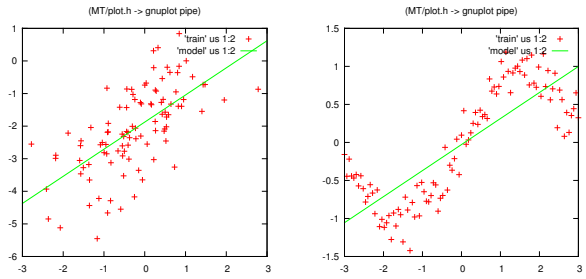
$$L^{\text{ls}}(\beta) = \sum_{i=1}^n (y_i - \boldsymbol{x}_i^\top \beta)^2 = \|y - \boldsymbol{X}\beta\|^2$$

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_n^\top \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ \vdots & & & & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

- Optimum:

$$\mathbf{0}_d^\top = \frac{\partial L^{\text{ls}}(\beta)}{\partial \beta} = -2(y - \boldsymbol{X}\beta)^\top \boldsymbol{X} \iff \mathbf{0}_d = \boldsymbol{X}^\top \boldsymbol{X}\beta - \boldsymbol{X}^\top y$$

$$\hat{\beta}^{\text{ls}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top y$$



Non-linear features

- Replace the inputs $x_i \in \mathbb{R}^d$ by some non-linear features $\phi(x_i) \in \mathbb{R}^k$

$$f(x) = \sum_{j=1}^k \phi_j(x)\, \beta_j = \phi(x)^\top \beta$$

- The optimal β is the same

$$\hat{\beta}^{\text{ls}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \quad \text{but with} \quad \boldsymbol{X} = \begin{pmatrix} \phi(x_1)^\top \\ \vdots \\ \phi(x_n)^\top \end{pmatrix} \in \mathbb{R}^{n \times k}$$

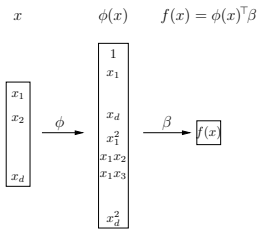
- What are "features"?

  a) Features are an arbitrary set of basis functions

  b) Any function *linear in* β can be written as $f(x) = \phi(x)^\top \beta$ for some ϕ – which we denote as "features"

Example: Polynomial features

- Linear: $\phi(x) = (1, x_1, .., x_d) \in \mathbb{R}^{1+d}$
- Quadratic: $\phi(x) = (1, x_1, .., x_d, x_1^2, x_1x_2, x_1x_3, .., x_d^2) \in \mathbb{R}^{1+d+\frac{d(d+1)}{2}}$
- Cubic: $\phi(x) = (.., x_1^3, x_1^2x_2, x_1^2x_3, .., x_d^3) \in \mathbb{R}^{1+d+\frac{d(d+1)}{2}+\frac{d(d+1)(d+2)}{6}}$

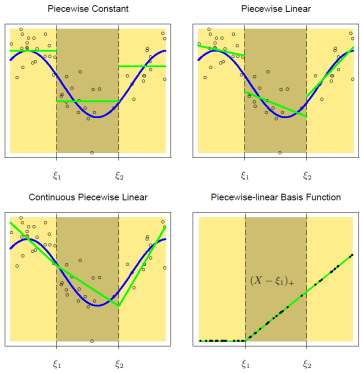



```
./x.exe -mode 1 -dataFeatureType 1 -modelFeatureType 1
```

2:7

Example: Piece-wise features

- Piece-wise constant: $\phi_j(x) = I(\xi_1 < x < \xi_2)$
- Piece-wise linear: $\phi_j(x) = xI(\xi_1 < x < \xi_2)$
- Continuous piece-wise linear: $\phi_j(x) = (x - \xi_1)_+$



2:8

Example: Radial Basis Functions (RBF)

- Given a set of centers $\{c_j\}_{j=1}^k$, define

$$\phi_j(x) = b(x, c_j) = e^{-\frac{1}{2}\|x - c_j\|^2} \in [0, 1]$$

Each $\phi_j(x)$ measures similarity with the center c_j

- Special case:

use all training inputs $\{x_i\}_{i=1}^n$ *as centers*

$$\phi(x) = \begin{pmatrix} 1 \\ b(x, x_1) \\ \vdots \\ b(x, x_n) \end{pmatrix} \quad (n + 1 \text{ dim})$$

This is related to "kernel methods" and GPs, but not quite the same – we'll discuss this later.

2:9

Features

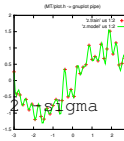
- Polynomial
- Piece-wise
- Radial basis functions (RBF)
- Splines (see Hastie Ch. 5)

- Linear regression on top of rich features is extremely powerful!

2:10

The need for regularization

Noisy $\sin$ data fitted with radial basis functions



```
./x.exe -mode 1 -n 40 -modelFeatureType 4 -dataType 2 -sigma
.3 -lambda 1e-10
```

- **Overfitting & generalization:**

The model overfits to the data – and generalizes badly

- **Estimator variance:**

When you repeat the experiment (keeping the underlying function fixed), the regression always returns a different model estimate

2:11

Estimator variance

- Assumptions:

– We computed parameters $\hat{\beta} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top y$

– The data was noisy with variance $\mathrm{Var}(y) = \sigma^2\mathbf{I}_n$

Then

$$\mathrm{Var}(\hat{\beta}) = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\sigma^2$$

– high data noise $\sigma \to$ high estimator variance

– more data $\to$ less estimator variance $\mathrm{Var}(\hat{\beta}) \propto \frac{1}{n}$

- In practise we don't know σ, but we can estimate it based on the deviation from the learnt model:

$$\hat{\sigma}^2 = \frac{1}{n - d - 1}\sum_{i=1}^n (y_i - f(x_i))^2$$

2:12

Estimator variance

- "Overfitting"

$\leftarrow$ picking one specific data set $y \sim \mathcal{N}(y_{mean}, \sigma^2\mathbf{I}_n)$

$\leftrightarrow$ picking one specific $\hat{b} \sim \mathcal{N}(\beta_{mean}, (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\sigma^2)$

- If we could reduce the variance of the estimator, we could reduce overfitting – and increase generalization.

2:13

Hastie's section on shrinkage methods is great! Describes several ideas on reducing estimator variance — by reducing model complexity. We focus on regularization.

2:14

Ridge regression: L_2-regularization

- We add a *regularization* to the cost:

$$L^{\text{ridge}}(\beta) = \sum_{i=1}^n (y_i - \phi(x_i)^\top \beta)^2 + \lambda \sum_{j=1}^k \beta_j^2$$

NOTE: β_0 is usually *not* regularized!

- Optimum:

$$\hat\beta^{\text{ridge}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda I)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

(where $I = \mathbf{I}_k$, or with $I_{1,1} = 0$ if β_0 is not regularized)

2:15

- The objective is now composed of two "potential": The loss, which depends on the data and jumps around (introduces variance), and the regularization penalty (sitting steadily at zero). Both are "pulling" at the optimal $\beta \to$ the regularization reduces variance.

- The estimator variance becomes less: $\text{Var}(\hat\beta) = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda I)^{-1} \sigma^2$

- The ridge effectively reduces the complexity of the model:

$$\text{df}(\lambda) = \sum_{j=1}^d \frac{d_j^2}{d_j^2 + \lambda}$$

where d_j^2 is the eigenvalue of $\boldsymbol{X}^\top \boldsymbol{X} = V D^2 V^\top$

(details: Hastie 3.4.1)

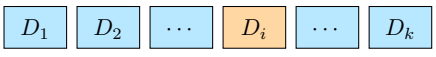
2:16

Choosing λ & cross validation

- $\lambda = 0$ will always have a lower *training* data error

  We need to estimate the *generalization* error on test data

- k-*fold cross-validation:*



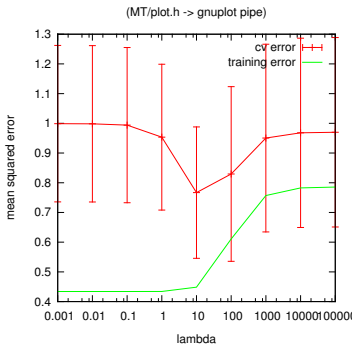
1: Partition data D in k equal sized subsets $D = \{D_1, .., D_k\}$
2: **for** $i = 1, .., k$ **do**
3: compute $\hat\beta_i$ on the training data $D \setminus D_i$ leaving out D_i
4: compute the error $\ell_i = L^{\text{ls}}(\beta_i, D_i)$ on the validation data D_i
5: **end for**
6: report mean error $\hat\ell = 1/k \sum_i l_i$ and variance $(1/k \sum_i l_i^2) - \hat\ell^2$

- Choose λ for which $\hat\ell$ is smallest

2:17

---

quadratic features on sinus data:




```
./x.exe -mode 4 -n 10 -modelFeatureType 2 -dataType 2 -sigma
.1
./x.exe -mode 1 -n 10 -modelFeatureType 2 -dataType 2 -sigma
.1
```

2:18

Lasso: L_1-regularization

- We add a L_1 regularization to the cost:

$$L^{\text{lasso}}(\beta) = \sum_{i=1}^n (y_i - \phi(x_i)^\top \beta)^2 + \lambda \sum_{j=1}^k |\beta_j|$$

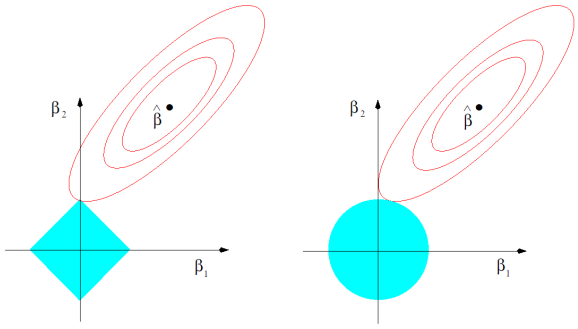
NOTE: β_0 is usually not regularized!

- Has no closed form expression for optimum

(Optimum can be found by solving a quadratic program; see appendix.)

2:19

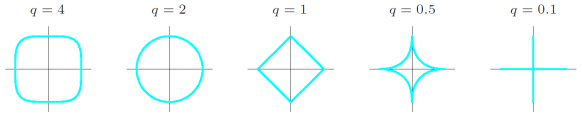
Lasso vs. Ridge:



- Lasso $\to$ sparsity! feature selection!

2:20

$$L^q(\beta) = \sum_{i=1}^n (y_i - \phi(x_i)^\top \beta)^2 + \lambda \sum_{j=1}^k |\beta_j|^q$$



- *Subset selection:* $q = 0$ (counting the number of $\beta_j \neq 0$)

2:21

Summary

- **Linear models** on non-linear features – extremely powerful

linear polynomial RBF kernel	Ridge Lasso	regression classification

- Generalization $\leftrightarrow$ **Regularization** $\leftrightarrow$ complexity/DoF penalty

- **Cross validation** to estimate generalization empirically $\rightarrow$ use to choose regularization parameters

2:22

Kernel Ridge Regression – the "Kernel Trick"

- Reconsider solution of Ridge regression:

$$\hat\beta^{\text{ridge}} = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\mathbf{I}_k)^{-1}\boldsymbol{X}^\top\boldsymbol{y} = \boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top + \lambda\mathbf{I}_n)^{-1}\boldsymbol{y}$$

- Recall $\boldsymbol{X}^\top = (\phi(x_1), .., \phi(x_n)) \in \mathbb{R}^{k\times n}$, then:

$$f^{\text{ridge}}(x) = \phi(x)^\top\beta^{\text{ridge}} = \underbrace{\phi(x)^\top\boldsymbol{X}^\top}_{\boldsymbol{\kappa}(x)}(\underbrace{\boldsymbol{X}\boldsymbol{X}^\top}_{\boldsymbol{K}}+\lambda I)^{-1}\boldsymbol{y}$$

$\boldsymbol{K}$ is called *kernel matrix* and has elements

$$\boldsymbol{K}_{ij} = k(x_i, x_j) := \phi(x_i)^\top\phi(x_j)$$

$\boldsymbol{\kappa}$ is the kernel vector: $\boldsymbol{\kappa}(x) = k(x, x_{1:n}) = \phi(x)^\top\boldsymbol{X}^\top$.

The kernel function $k(x, x')$ calculates the scalar product in feature space.

2:23

The Kernel Trick

- We can rewrite kernel ridge regression as:

$$f^{\text{rigde}}(x) = \boldsymbol{\kappa}(x)(\boldsymbol{K} + \lambda I)^{-1}\boldsymbol{y}$$
$$\text{with } \boldsymbol{K}_{ij} = k(x_i, x_j)$$
$$\boldsymbol{\kappa}_i(x) = k(x, x_i)$$

$\rightarrow$ at no place we actually need to compute the parameters $\hat\beta$

$\rightarrow$ at no place we actually need to compute the features $\phi(x_i)$

– we only need to be able to compute $k(x, x')$ for any x, x'

- This rewriting is called *kernel trick*.
- It has great implications:

– Instead of inventing funny non-linear features, we may directly invent funny kernels

– Inventing a kernel is intuitive: $k(x, x')$ expresses how correlated y and y' should be: it is a meassure of similarity, it compares x and x'. Specifying how 'comparable' x and x' are is often more intuitive than defining "features that might work".

2:24

- Every choice of features implies a kernel. But,

Does every choice of kernel correspond to specific choice of feature?

2:25

Reproducing Kernel Hilbert Space

- Let's define a vector space $\mathcal{H}_k$, spanned by infinitely many basis elements

$$\{h_x = k(\cdot, x) \;:\; x \in \mathbb{R}^d\}$$

Vectors in this space are linear combinations of such basis elements, e.g.,

$$f = \sum_i \alpha_i h_{x_i}\,, \quad f(x) = \sum_i \alpha_i k(x, x_i)$$

- Let's define a scalar product in this space, by first defining the scalar product for every basis element,

$$\langle h_x, h_y\rangle := k(x, y)$$

This is positive definite. Note, it follows

$$\langle h_x, f\rangle = \sum_i \alpha_i \langle h_x, h_{x_i}\rangle = \sum_i \alpha_i k(x, x_i) = f(x)$$

- The $\phi(x) = h_x = k(\cdot, x)$ is the 'feature' we associate with x. Note that this is a function and infinite dimensional.
Choosing $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda I)^{-1}\boldsymbol{y}$ represents $f^{\text{ridge}}(x) = \sum_{i=1}^n \alpha_i k(x, x_i)$, and shows that ridge regression has a finite-dimensional solution in the basis elements $\{h_{x_i}\}$. A much more general version of this insight is called **representer theorem**.

2:26

Example Kernels

- Kernel functions need to be positive definite: $\forall_{z:|z|>0} : k(z, z') > 0$

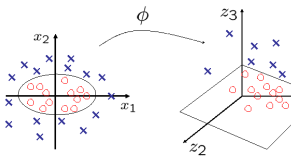
$\rightarrow \boldsymbol{K}$ is a positive definite matrix

- Examples:

– Polynomial: $k(x, x') = (x^\top x')^d$

$d = 2, x \in \mathbb{R}^2$, $\phi(x) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)^\top$

$$k(x, x') = ((x_1, x_2)\begin{pmatrix}x_1'\\x_2'\end{pmatrix})^2$$
$$= (x_1x_1' + x_2x_2')^2$$
$$= x_1^2{x_1'}^2 + 2x_1x_2x_1'x_2' + x_2^2{x_2'}^2$$
$$= (x_1^2, \sqrt{2}x_1x_2, x_2^2)({x_1'}^2, \sqrt{2}x_1'x_2', {x_2'}^2)^\top$$
$$= \phi(x)^\top\phi(x')$$



– Gaussian (radial basis function): $k(x, x') = \exp(-\gamma\,|\,x - x'\,|^2)$

2:27

Appendix: Alternative formulation of Ridge

- The standard way to write the Ridge regularization:

$$L^{\text{ridge}}(\beta) = \textstyle\sum_{i=1}^n (y_i - \phi(x_i)^\top\beta)^2 + \lambda\sum_{j=1}^k \beta_j^2$$

- Finding $\hat\beta^{\text{ridge}} = \text{argmin}_\beta\, L^{\text{ridge}}(\beta)$ is equivalent to solving

$$\hat\beta^{\text{ridge}} = \text{argmin}_\beta \sum_{i=1}^n (y_i - \phi(x_i)^\top\beta)^2$$
$$\text{subject to } \sum_{j=1}^k \beta_j^2 \le t$$

λ is the Karush-Kuhn-Tucker multiplier for the inequality constraint (generalization of Lagrange multiplier)

2:28

Appendix: Alternative formulation of Lasso

- The standard way to write the Lasso regularization:
$$L^{\text{lasso}}(\beta) = \sum_{i=1}^n (y_i - \phi(x_i)^\top\beta)^2 + \lambda \sum_{j=1}^k |\beta_j|$$

- Equivalent formulation (via KKT):

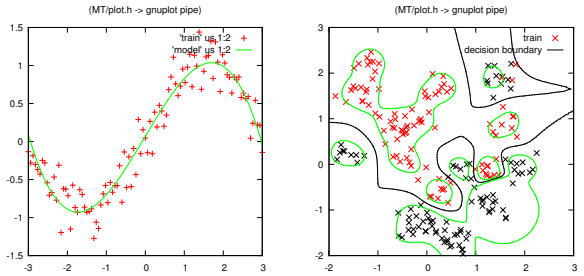
$$\hat\beta^{\text{lasso}} = \text{argmin}_\beta \sum_{i=1}^n (y_i - \phi(x_i)^\top\beta)^2$$
$$\text{subject to } \sum_{j=1}^k |\beta_j| \le t$$

2:29

3 Classification basics

Classification, discriminative function, logistic regression, binary & multi-class case, conditional random fields

Regression vs. classification



- regression: map input x to continuous value $\hat{y} \in \mathbb{R}$

  classification: map input x to one of M classes $\hat{y} \in \{1, 2, .., M\}$

3:1

Discriminative function

- Regression

  "map input x to continuous value $\hat{y} \in \mathbb{R}$"

  $\rightarrow$ we want to learn a *predictive function* $\hat{y}(x) = f(x)$

- Classification

  "map input x to one of M classes $\hat{y} \in \{1, 2, .., M\}$"

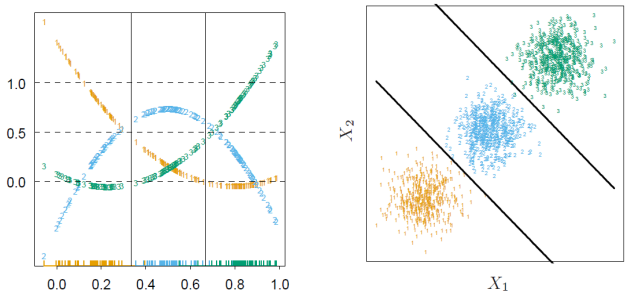
  $\rightarrow$ we want to learn a *discriminative function* $f(y, x)$,

$$\hat{y}(x) = \text{argmax}_y f(y, x)$$

$f(y, x)$ should be high if y is the correct class, and low if y is the false class

3:2

Discriminative function



Left: shows the three functions $f(1, x)$, $f(2, x)$, $f(3, x)$ that discriminate the three classes (along diagonal axis).

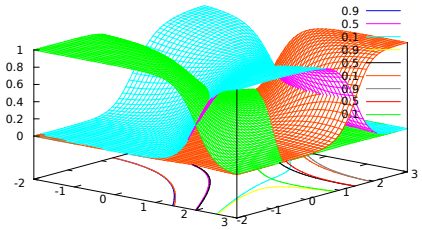
Sometimes it is helpful to think of $f(y, x)$ as M separate functions for each class $y \in \{1, .., M\}$ – the highest one determines the class prediction $\hat{y}$

```
plot[-3:3] -x-2,0,x-2    splot[-3:3][-3:3] -x-y-2,0,x+y-2
```

3:3

Discriminative function

$f(1, x)$, $f(2, x)$, $f(3, x)$



(here already "scaled" to interval [0,1]... explained later)

3:4

Regression of class indicators?

- What is a good objective to optimize $f(y, x)$?
- Let's start with a *bad* idea: *regression of class indicators*

Train $f(y, x)$ to be the indicator function for class y

that is, $\forall y$: train $f(y, x)$ on the regression data $D = \{(x_i, I(y = y_i))\}_{i=1}^n$

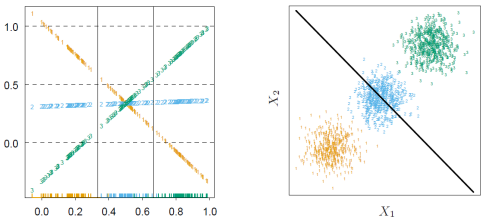
train $f(1, x)$ on value 1 for all x_i with $y_i = 1$ and on 0 otherwise
train $f(2, x)$ on value 1 for all x_i with $y_i = 2$ and on 0 otherwise
train $f(3, x)$ on value 1 for all x_i with $y_i = 3$ and on 0 otherwise

$\vdots$

3:5

Regression of class indicators?

this may fail:



Although the optimal separating boundaries are linear and linear discriminating functions could represent them, the linear functions trained on class indicators fail to discriminate.

$\rightarrow$ *regression on class indicators is the "wrong objective"*

Hastie 4.2

3:6

- What we need is a proper loss function for classification!

  – In regression, we had the *least squares* loss
  $$L^{\text{ls}}(f) = \sum_{i=1}^n (y_i - f(x_i))^2$$

- For classification:

  – We interpret the discriminative function $f(y,x)$ as class probabilities
  $$p(y\,|\,x) = \frac{e^{f(y,x)}}{\sum_{y'} e^{f(y',x)}}$$
  $\to p(y\,|\,x)$ should be high for the correct class, and low otherwise

  – For each (x_i, y_i) we want to maximize $p(y_i\,|\,x_i)$ (the *likelihood*)
  $$L^{\text{neg-log-likelihood}}(f) = -\sum_{i=1}^n \log p(y_i\,|\,x_i)$$

3:7

Regression vs. Classification

Regression	**Classification**
parameters β	parameters β
$\downarrow$	$\downarrow$
predictive function	discriminative function
$f(x) = \phi(x)^\top \beta$	$f(y,x) = \phi(x)^\top \beta_y$
$\downarrow$	$\downarrow$
least squares loss	class probabilities
$L^{\text{ls}}(\beta) = \sum_{i=1}^n (y_i - f(x_i))^2$	$p(y\,|\,x) \propto e^{f(y,x)}$
	$\downarrow$
	neg-log-likelihood
	$L^{\text{neg-log-likelihood}}(\beta) = -\sum_{i=1}^n \log p(y_i\,|\,x_i)$

- In the following: explicit equations for

  – binary case

  – multi-class case

  – Conditional Random Fields

3:8

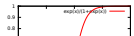
Logistic regression: binary case

- Data $D = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$

- Discriminative functions $f(0,x) = 0$ and $f(1,x) = \phi(x)^\top \beta$
  $$\Rightarrow \quad \hat{y} = \text{argmax}_y f(y,x) = \begin{cases} 0 & \text{else} \\ 1 & \text{if } \phi(x)^\top \beta > 0 \end{cases}$$

- Conditional class probabilities
  $$p(1\,|\,x) = \frac{e^{f(1,x)}}{e^{f(0,x)} + e^{f(1,x)}} = \sigma(f(1,x))$$
  with the *logistic sigmoid function* $\sigma(z) = \frac{e^z}{1+e^z} = \frac{1}{e^{-z}+1}$.

- Neg-log-Likelihood for data $D = \{(x_i, y_i)\}_{i=1}^n$



$$L^{\text{logistic}}(\beta) = -\sum_{i=1}^n \log p(y_i\,|\,x_i) + \lambda\|\beta\|^2$$
$$= -\sum_{i=1}^n \Big[y_i \log p(1\,|\,x_i) + (1-y_i) \log[1 - p(1\,|\,x_i)] \Big] + \lambda\|\beta\|^2$$

(here with Ridge regularization term $\lambda\|\beta\|^2$)

3:9

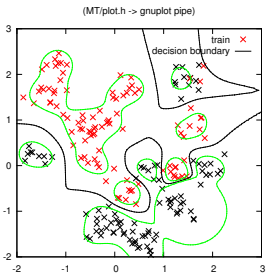
Optimal parameters β

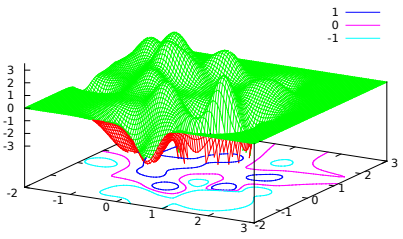
- Gradient (see exercises):
  $$\frac{\partial L^{\text{logistic}}(\beta)}{\partial \beta}^\top = \sum_{i=1}^n (p_i - y_i)\phi(x_i) + 2\lambda I\beta = \boldsymbol{X}^\top(\boldsymbol{p} - \boldsymbol{y}) + 2\lambda I\beta$$
  $$, \quad p_i := p(y{=}1\,|\,x_i) \quad , \quad \boldsymbol{X} = \begin{pmatrix} \phi(x_1)^\top \\ \vdots \\ \phi(x_n)^\top \end{pmatrix} \in \mathbb{R}^{n \times k}$$
  $\frac{\partial L^{\text{logistic}}(\beta)}{\partial \beta}$ is non-linear in β (it enters also the calculation of p_i)
  $\to$ does not have analytic solution

- Newton algorithm: iterate
  $$\beta \leftarrow \beta - H^{-1} \frac{\partial L^{\text{logistic}}(\beta)}{\partial \beta}^\top$$
  with Hessian $H = \frac{\partial^2 L^{\text{logistic}}(\beta)}{\partial \beta^2} = \boldsymbol{X}^\top W \boldsymbol{X} + 2\lambda I$
  W diagonal with $W_{ii} = p_i(1 - p_i)$

3:10

RBF ridge logistic regression:

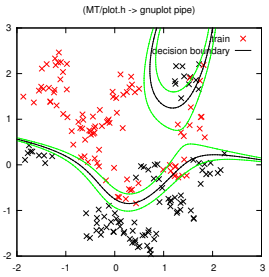


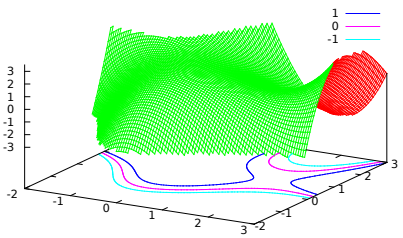


```
./x.exe -mode 2 -modelFeatureType 4 -lambda 1e+0 -rbfBias
0 -rbfWidth .2
```

3:11

polynomial (cubic) logistic regression:





```
./x.exe -mode 2 -modelFeatureType 3 -lambda 1e+0
```

3:12

Logistic regression: multi-class case

- Data $D = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \{1, .., M\}$
- Discriminative functions $f(y, x) = \phi(x)^\top \beta_y$

  (optionally we may set $f(M, x) = 0$)

  $\to$ we have M (or $M-1$) parameter vectors β_y
- Conditional class probabilties

  $p(y \mid x) = \frac{e^{f(y,x)}}{\sum_{y'} e^{f(y',x)}} \quad \leftrightarrow \quad f(y, x) = \log \frac{p(y \mid x)}{p(y{=}M \mid x)}$

  (the discriminative functions model "*log-ratios*")
- Neg-log-Likelihood for data $D = \{(x_i, y_i)\}_{i=1}^n$

  $L^{\text{logistic}}(\beta) = -\sum_{i=1}^n \log p(y{=}y_i \mid x_i) + \lambda \|\beta\|^2$

3:13

Optimal parameters β

- Gradient:

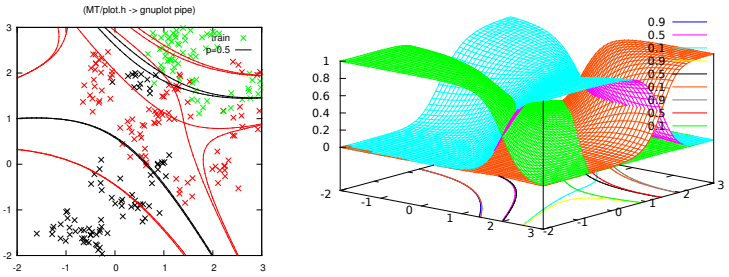
  $\frac{\partial L^{\text{logistic}}(\beta)}{\partial \beta_c} = \sum_{i=1}^n (p_{ic} - y_{ic}) \phi(x_i) + 2\lambda I \beta_c = \boldsymbol{X}^\top (\boldsymbol{p}_c - \boldsymbol{y}_c) + 2\lambda I \beta_c$

  , $p_{ic} = p(y{=}c \mid x_i)$
- Hessian: $H = \frac{\partial^2 L^{\text{logistic}}(\beta)}{\partial \beta_c \partial \beta_d} = \boldsymbol{X}^\top W_{cd} \boldsymbol{X} + 2[c = d]\, \lambda I$

  W_{cd} diagonal with $W_{cd,ii} = p_{ic}([c = d] - p_{id})$

3:14

polynomial (quadratic) ridge 3-class logistic regression:




```
./x.exe -mode 3 -modelFeatureType 3 -lambda 1e+1
```

3:15

Conditional Random Fields

- The **discriminative function** $f(y, x)$ is central for classification

  For any x, the function discriminates between labels $y = 1, .., M$
- Can we do the same when y is a much more complex "label"

  not just a $0/1$ or class integer, but a complex large *labelling structure*
- Words for this

  – **Structured output learning**

  – **Conditional Random Field (CRF)**

  The name CRF originates from a probabilistic formulation, which we don't need for now.

3:16

Examples for CRFs

- Text tagging

  X = sentence

  Y = tagging of each word

  http://sourceforge.net/projects/crftagger
- Image segmentation

  X = image

  Y = labelling of each pixel

  http://scholar.google.com/scholar?cluster=13447702299042713582
- Depth estimation

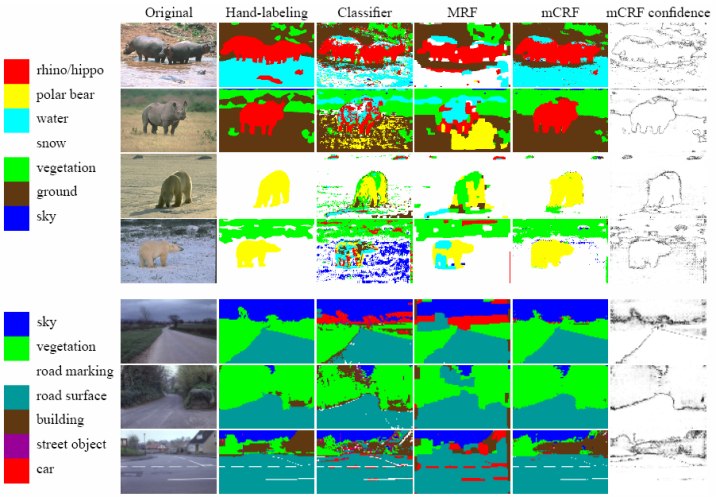
  X = single image

  Y = depth map

  http://make3d.cs.cornell.edu/

3:17

CRFs in image processing



3:18

CRFs in image processing

- Google "conditional random field image"

  – Multiscale Conditional Random Fields for Image Labeling (CVPR 2004)

  – Scale-Invariant Contour Completion Using Conditional Random Fields (ICCV 2005)

  – Conditional Random Fields for Object Recognition (NIPS 2004)

  – Image Modeling using Tree Structured Conditional Random Fields (IJCAI 2007)

  – A Conditional Random Field Model for Video Super-resolution (ICPR 2006)

3:19

Conditional Random Fields

- Actually, we can directly apply (multi-class) logistic regression on these problems! No need to change anything "in principle". Same model for $f(y,x)$, same cost function.

- However, for this to be practical we need to learn how to elegantly define/parameterize $f(y,x)$ when y is a complex thing.

3:20

Conditional Random Fields

- Assume $y = (y_1, .., y_l)$ is a tuple of many individual (local) discrete labels
- As always, we assume that $f(y,x)$ is **linear in features:**

$$f(y,x) = \sum_{j=1}^k \phi_j(y_{\partial j}, x)\beta_j = \phi(y,x)^\top\beta$$

Note: Each feature $\phi_j(y_{\partial j}, x)$ only depends on a subset $y_{\partial j}$ of labels. It **couples** these labels.

- Recall, in the multi-class case we thought of $f(y,x)$ as M different functions, each one having different parameters β_y, but same features $\phi(x)$. In the above convention this would be encoded

$$\phi(y,x) = \begin{pmatrix} [y=1]\ \phi(x) \\ [y=2]\ \phi(x) \\ [y=3]\ \phi(x) \end{pmatrix}$$

such that $f(y,x) = \phi(y,x)^\top\beta = \phi(x)^\top\beta_y$, as before.

3:21

CRF: Basic equations

$$\begin{aligned}
f(y,x) &= \phi(y,x)^\top\beta \\
p(y|x) &= \frac{e^{f(y,x)}}{\sum_{y'} e^{f(y',x)}} = e^{f(y,x) - Z(x,\beta)} \\
Z(x,\beta) &= \log \sum_{y'} e^{f(y',x)} \quad \text{(log partition function)} \\
L(\beta) &= -\sum_i \log p(y_i|x_i) = -\sum_i [\phi(y,x)^\top\beta - Z(x_i,\beta)] \\
\frac{\partial}{\partial\beta} Z(x,\beta) &= \sum_y p(y|x)\phi(y,x)^\top \\
\frac{\partial^2}{\partial\beta^2} Z(x,\beta) &= \sum_y p(y|x)\phi(y,x)\phi(y,x)^\top
\end{aligned}$$

- This gives the neg-log-likelihood $L(\beta)$, its gradient and Hessian

3:22

Training CRFs

- Maximize conditional likelihood

But Hessian is typically too large (Images: $\sim$10 000 pixels, $\sim$50 000 features)

Alternative: Efficient gradient method, e.g.:

Vishwanathan et al.: Accelerated Training of Conditional Random Fields with Stochastic Gradient Methods

- Other loss variants, e.g., hinge loss as with Support Vector Machines

("Structured output SVMs")

- Perceptron algorithm: Minimizes hinge loss using a gradient method

3:23

End of Part I: The Basis

features/kernel	+	**regularization**	+	**loss**
polynomial RBF sigmoidal kernel		Ridge Lasso **cross validation**		least squares regression likelihood maximization (logistic regression) (cond. random fields)

- I consider this to be a core basis for understanding ML methods
  - In terms of regression/classification performance, they should be state-of-the-art (for good choice of features/kernel)
  - But in direct form these are not always computationally most efficient

- **Part 2** will consider a multitude of ideas to extend this

- **Part 3** will introduce to the Bayesian formalism, which allows to derive the same core methods from another prespective—and extend them

3:24

4 Part 2: The Breadth of ML ideas

Features & data preprocessing: centering & whitening, PCA, PLS; Local & lazy learning, kNN, kd-trees; Combining weak or randomized learners: Bootstrap, bagging, model averaging, boosting; Other loss functions: hinge loss, linear programming SVMs; Deep learning & representations

The Breadth of ML ideas

- A. Ideas about features & data preprocessing
  – centering & whitening
  – PCA
  – PLS (for classification?)
- B. Ideas about local learners
  – local & lazy learning
  – kNN
  – kd-trees
- C. Ideas about combining weak or randomized learners
  – Bootstrap, bagging, and model averaging
  – Boosting
  – (Boosted) decision trees & stumps
  – Random forests
- D. Ideas about other loss functions
  – hinge loss, linear programming, SVMs
- E. Ideas about deep learners and representations

4:1

Centering & Whitening

- Some researchers prefer to *center* (shift to zero mean) the data before applying methods:

$$x \leftarrow x - \langle x \rangle \ , \quad y \leftarrow y - \langle y \rangle$$

this spares augmenting the bias feature 1 to the data.

- More interesting: The loss and the best choice of λ depends on the *scaling* of the data. If we always scale the data in the same range, we may have better priors about choice of λ and interpretation of the loss

$$x \leftarrow \frac{1}{\sqrt{\text{Var}(x)}}\, x \ , \quad y \leftarrow \frac{1}{\sqrt{\text{Var}(y)}}\, y$$

- **Whitening:** Transform the data to remove all correlations and varances.

Let $A = \text{Var}(x) = \frac{1}{n}\boldsymbol{X}^\top\boldsymbol{X} - \mu\mu^\top$ with Cholesy decomposition $A = MM^\top$.

$$x \leftarrow M^{-1}x \ , \quad \text{with } \text{Var}(M^{-1}x) = \mathbf{I}_d$$

4:2

Principle Component Analysis (PCA)

4:3

Principle Component Analysis (PCA)

- Imagine $x \in \mathbb{R}^d$ with really large d

  $\rightarrow$ high computational costs

  $\rightarrow$ many parameters, overfitting

  Are really all input dimensions necessary?

- Idea: We only select a subset of input dimensions

4:4

Principle Component Analysis (PCA)

- Heuristic: Pick the directions of **largest variance**

- Given data in $\mathbb{R}^d$ we can compute
  – the mean $\mu = \langle x \rangle = \frac{1}{n}\sum_i x_i$
  – the variance $A = \text{Var}(x) = \langle xx^\top \rangle - \mu\mu^\top = \frac{1}{n}\boldsymbol{X}^\top\boldsymbol{X} - \mu\mu^\top$
- The *principle components* are the those directions that have largest variance
  The **singular value decomposition** of a symmetric matrix is

$$A = VDV^\top$$

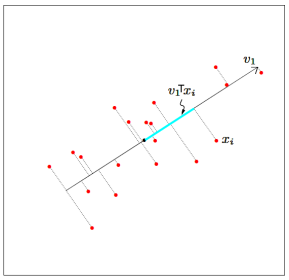
with $D = \text{diag}(\sigma_1^2, .., \sigma_d^2)$, $\sigma_1 \geq ... \geq \sigma_d$
and $V = (v_1, .., v_d)$ with *orthonormal* row vectors v_i

The sub-matrix $V_p = (v_1, .., v_p) \in \mathbb{R}^{d\times p}$ defines an orthogonal "p-dimensional coordinate system"

- $z = V_p^\top x$ is the projection of an input into these p dimensions

4:5

Principle Component Analysis (PCA)



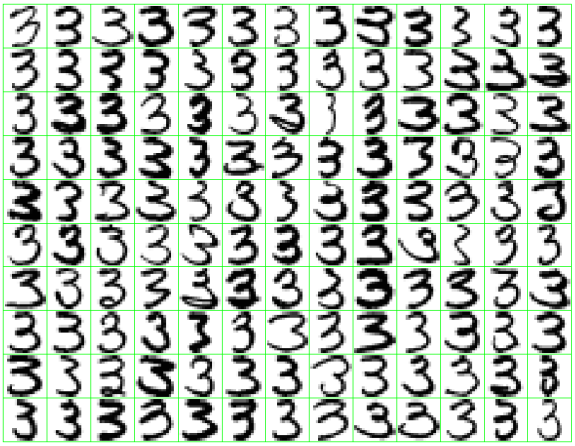
$V_p^\top$ is the matrix that projects to the largest variance directions of $\boldsymbol{X}^\top\boldsymbol{X}$

$$x \mapsto z = V_p^\top x$$
$$\boldsymbol{X} \mapsto \boldsymbol{Z} = \boldsymbol{X}V_p$$

- Heuristic: Apply ML method on top of $\boldsymbol{Z}$ instead of $\boldsymbol{X}$

4:6

Example: Digits



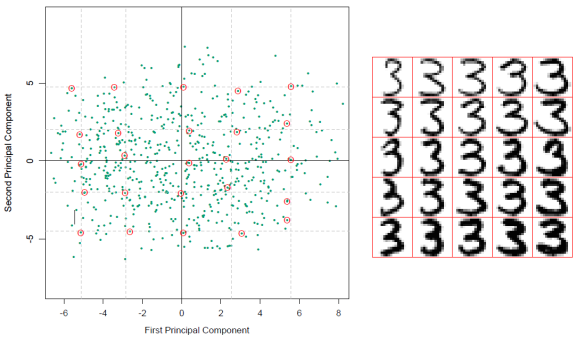
4:7

Example: Digits

- The "basis vectors" in V_p are also **eigenvectors**

  Every data point can be expressed in these eigenvectors

$$\begin{aligned} x &\approx \mu + V_p z \\ &= \mu + z_1 v_1 + z_2 v_2 + \dots \\ &= \boxed{3} + z_1 \cdot \boxed{3} + z_2 \cdot \boxed{3} + \cdots \end{aligned}$$



4:8

Example: Eigenfaces

(Viola & Jones)

4:9

"Feature PCA" & Kernel PCA (for reference only)

- The *feature* trick: $\boldsymbol{X} = \begin{pmatrix} \phi(x_1)^\top \\ \vdots \\ \phi(x_n)^\top \end{pmatrix} \in \mathbb{R}^{n \times k}$

- The *kernel* trick: rewrite all necessary equations such that they only involve scalar products $\phi(x)^\top \phi(x') = k(x, x')$:

We want to compute eigenvectors of $\boldsymbol{X}^\top \boldsymbol{X} = \sum_i \tilde{\phi}(x_i)\tilde{\phi}(x_i)^\top$. We can rewrite this as

$$\begin{aligned} \boldsymbol{X}^\top \boldsymbol{X} v_j &= \lambda v_j \\ \underbrace{\boldsymbol{X}\boldsymbol{X}^\top}_{\boldsymbol{K}} \underbrace{\boldsymbol{X} v_j}_{\boldsymbol{K}\alpha_j} &= \lambda \underbrace{\boldsymbol{X} v_j}_{\boldsymbol{K}\alpha_j} , \quad v_j = \sum_i \alpha_{ji} \tilde{\phi}(x_i) \\ \boldsymbol{K}\alpha_j &= \lambda \alpha_j \end{aligned}$$

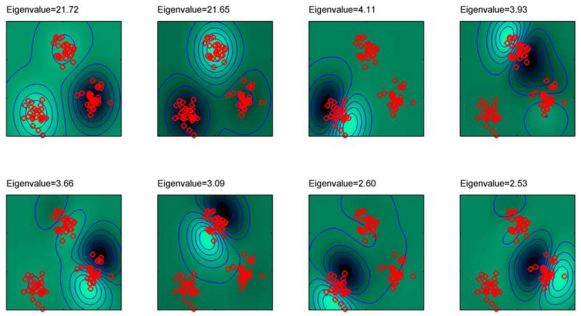
Where $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^\top$ with entries $\boldsymbol{K}_{ij} = \tilde{\phi}(x_i)^\top \tilde{\phi}(x_j)$.
$\rightarrow$ We compute SVD of the Gram matrix $\boldsymbol{K} \rightarrow$ gives eigenvectors $\boldsymbol{\alpha}_j \in \mathbb{R}^n$.
Projection: $x \mapsto z = V_p^\top \tilde{\phi}(x) = \sum_i \alpha_{1:p,i} \tilde{\phi}(x_i)^\top \tilde{\phi}(x) = A\boldsymbol{\kappa}(x)$
(with matrix $A \in \mathbb{R}^{p \times n}$, $A_{ji} = \alpha_{ji}$ and vector $\boldsymbol{\kappa}(x) \in \mathbb{R}^n$, $\boldsymbol{\kappa}_i(x) = k(x_i, x)$)
Since we cannot *center the features* $\phi(x)$ we actually need "the double centered Gram matrix" $\widetilde{\boldsymbol{K}} = (\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top)\boldsymbol{K}(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top)$, where $\boldsymbol{K}_{ij} = \phi(x_i)^\top \phi(x_j)$ is uncentered.

4:10

Kernel PCA

red points: data

green shading: eigenvector $\boldsymbol{\alpha}_j$ represented as $z_j(x) = \sum_i \alpha_{ji} k(x_j,$



Kernel PCA "coordinates" allow us to discriminate clusters!

4:11

Kernel PCA (for reference only)

- Kernel PCA uncovers quite surprising structure:

  While PCA is "merely" picks high-variance dimensions

  Kernel PCA picks high variance *features*—where features correspond to basis functions (RKHS elements) over x

- Kernel PCA may map data x_i to latent coordinates z_i where *clustering* is much easier

- All of the following can be represented as kernel PCA:
  – Local Linear Embedding
  – Metric Multidimensional Scaling
  – Laplacian Eigenmaps (Spectral Clustering)

see "Dimensionality Reduction: A Short Tutorial" by Ali Ghodsi

4:12

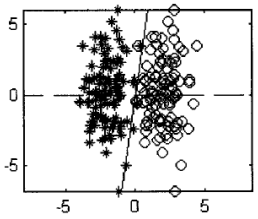
Partial Least Squares (PLS)

4:13

PLS

- Is it really a good idea to just pick the p-higest variance components??

Why should that be a good idea?



4:14

PLS

- Idea: The first dimension to pick should be the one **most correlated with the OUTPUT**, not with itself!

Input: data $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, $\boldsymbol{y} \in \mathbb{R}^n$
Output: predictions $\hat{\boldsymbol{y}} \in \mathbb{R}^n$

1: initialize the *predicted output*: $\hat{\boldsymbol{y}} = \langle y \rangle \, \mathbf{1}_n$
2: initialize the *remaining input dimensions*: $\hat{\boldsymbol{X}} = \boldsymbol{X}$
3: **for** $i = 1,..,p$ **do**
4: $\quad$ i-coordinate for all data points: $\boldsymbol{z}_i = \hat{\boldsymbol{X}}\hat{\boldsymbol{X}}^\top \boldsymbol{y}$
5: $\quad$ update prediction: $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} + \frac{\boldsymbol{z}_i^\top \boldsymbol{y}}{\boldsymbol{z}_i^\top \boldsymbol{z}_i}\, \boldsymbol{z}_i$
6: $\quad$ remove "used" input dimensions: $\hat{\boldsymbol{x}}_j \leftarrow \hat{\boldsymbol{x}}_j - \frac{\boldsymbol{z}_i^\top \hat{\boldsymbol{x}}_j}{\boldsymbol{z}_i^\top \boldsymbol{z}_i}\, \boldsymbol{z}_i$
$\quad$ where $\hat{\boldsymbol{x}}_j$ is the jth column of $\hat{\boldsymbol{X}}$
7: **end for**

(Hastie, page 81)

Line 4 identifies a new "coordinate" as the maximal correlation between the remaning input dimensions and $\boldsymbol{y}$. All z_i are orthogonal.
Line 5 updates the prediction that is possible using z_i
Line 6 removes the "used" dimension from $\hat{\boldsymbol{X}}$ to ensure orthgonality in line 4

4:15

PLS for classification

- Not obvious.
- We'll try to invent one in the exercises :-)

4:16

B. Ideas about local learners

– local & lazy learning
– kNN
– kd-trees

4:17

Local & lazy learning

- Idea of local (or "lazy") learning:

Do not try to build one global model $f(x)$ from the data. Instead, whenever we have a query points x^*, we build a specific local model in the neighborhood of x^*.

- Typical approach:
  – Given a query point x^*, find all kNN in the data $D = \{(x_i, y_i)\}_{i=1}^N$
  – Fit a local model f_{x^*} only to these kNNs, perhaps weighted
  – Use the local model f_{x^*} to predict $x^* \mapsto \hat{y}_0$

- Weighted local least squares:

$$L^{\text{local}}(\beta, x^*) = \textstyle\sum_{i=1}^n K(x^*, x_i)(y_i - f(x_i))^2 + \lambda\|\beta\|^2$$

where $K(x^*, x)$ is called *smoothing kernel*. The optimum is:

$$\hat{\beta} = (\boldsymbol{X}^\top W \boldsymbol{X} + \lambda I)^{-1} \boldsymbol{X}^\top W \boldsymbol{y}\,, \quad W = \text{diag}(K(x^*, x_{1:n}))$$

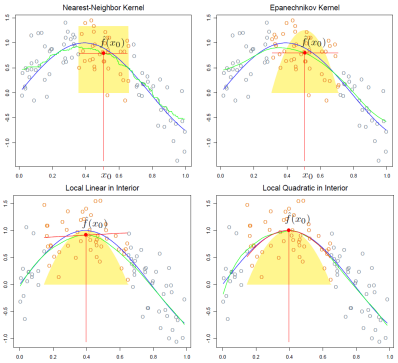
4:18

Local & lazy learning

kNN smoothing kernel: $K(x^*, x_i) = \begin{cases} 1 & \text{if } x_i \in \text{kNN}(x^*) \\ 0 & \text{otherwise} \end{cases}$

Epanechnikov quadratic smoothing kernel: $K_\lambda(x^*, x) = D(|x^* - x|/\lambda)\,, \quad D$
$\begin{cases} \frac{3}{4}(1-s^2) & \text{if } s \le 1 \\ 0 & \text{otherwise} \end{cases}$

(Hastie, Sec 6.3)



4:19

kd-trees

4:20

kd-trees

- For local & lazy learning it is essential to efficiently retrieve the kNN

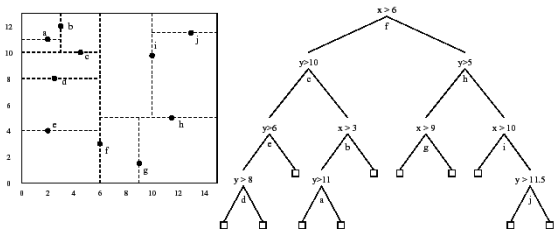
Problem: Given a data set X, a query x^*, identify the kNNs of x^* in X.

- Linear time (stepping through all of X) is far too slow.

A kd-tree pre-structures the data into a binary tree, allowing $O(\log n)$ retrieval of kNNs.

4:21

kd-trees



(There are "typos" in this figure... Exercise to find them.)

- Every node plays two roles:
  – it defines a hyperplane that separates the data along *one* coordinate
  – it hosts a data point, which lives exactly on the hyperplane (defines the division)

4:22

kd-trees

- Simplest (non-efficient) way to construct a kd-tree:
  – hyperplanes divide alternatingly along 1st, 2nd, ... coordinate
  – choose random point, use it to define hyperplane, divide data, iterate

- Nearest neighbor search:
  – descent to a leave node and take this as initial nearest point
  – ascent and check at each branching the possibility that a nearer point exists on the other side of the hyperplane

- Approximate Nearest Neighbor (libann on Debian..)

4:23

C. Ideas about combining weak or randomized learners

– Bootstrap, bagging, and model averaging

– Boosting

– (Boosted) decision trees & stumps

– Random forests

4:24

Combining learners

- The general idea is:
  – Given a data D, let us learn *various models* $f_1, .., f_M$
  – Our prediction is then some combination of these, e.g.

$$f(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$

- *Various models* could be:

Model averaging: Fully different types of models (using different (e.g. limited) feature sets; neural net; decision trees; whatever)

Bootstrap: Models of same type, trained on randomized versions of D

Boosting: Models of same type, trained on cleverly designed modifications/reweightings of D

- Concerning their combination, the interesting question is

How to choose the α_m?

4:25

Bootstrap & Bagging

- **Bootstrap:**
  – Data set D of size n
  – Generate M data sets D_m by resampling D *with replacement*
  – Each D_m is also of size n (some samples doubled or missing)

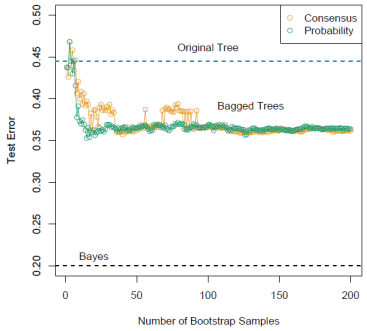
  – Distribution over data sets $\leftrightarrow$ distribution over β (compare slide 02-14)
  – The ensemble $\{f_1, .., f_M\}$ is similar to cross-validation
  – Mean and variance of $\{f_1, .., f_M\}$ can be used for model assessment

- **Bagging:** ("bootstrap aggregation")

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x)$$

4:26

- Bagging has similar effect to regularization:



(Hastie, Sec 8.7)

4:27

Bayesian Model Averaging

- If $f_1, .., f_M$ are very different model
  – Equal weighting would not be clever
  – More confident models (less variance, less parameters, high likelihood) $\rightarrow$ higher weight

- Bayesian Averaging

$$P(y|x) = \sum_{m=1}^{M} P(y|x, f_m, D)\ P(f_m|D)$$

The term $P(f_m|D)$ is the weighting α_m: it is high, when the model is likely under the data ($\leftrightarrow$ the data is likely under the model & the model has "fewer parameters").

4:28

The basis function view: Models are features!

- Compare model averaging $f(x) = \sum_{m=1}^M \alpha_m f_m(x)$ with regression:

$$f(x) = \sum_{j=1}^k \phi_j(x)\ \beta_j = \phi(x)^\top \beta$$

- We can think of the M models f_m as **features** ϕ_j for linear regression!
  - We know how to find optimal parameters α
  - But beware overfitting!

4:29

Boosting

4:30

Boosting

- In Bagging and Model Averaging, the models are trained on the same data, or unbiased randomized versions
- Boosting tries to be cleverer:
  - It adapts the data for each learner
  - It assigns each learner a differently *weighted* version of the data
- With this, boosing can
  - *Combine many "weak" classifiers to produce a powerful "committee"*
  - A weak learner only needs to be somewhat better than random

4:31

AdaBoost

(Freund & Schapire, 1997)

- Binary classification problem with data $D = \{(x_i, y_i)\}_{i=1}^n$, $y_i \in \{-1, +1\}$
- We know how to train discriminative functions $f(x)$; let

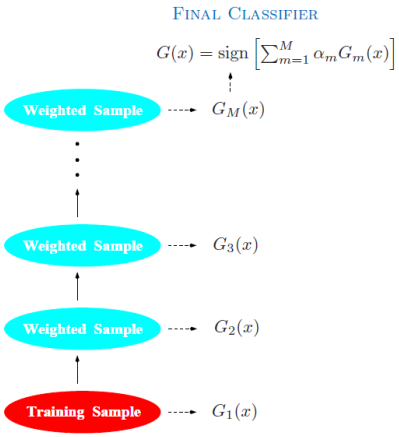
$$G(x) = \text{sign}\, f(x) \quad \in \{-1, +1\}$$

- We will train a sequence of classificers $G_1, .., G_M$, each on differently weighted data, to yield a classifier

$$G(x) = \text{sign} \sum_{m=1}^M \alpha_m G_m(x)$$

4:32

AdaBoost



(Hastie, Sec 10.1)

4:33

AdaBoost

Input: data $D_m = \{(x_i, y_i)\}_{i=1}^n$
Output: family of classifiers G_m and weights α_m

1: initialize $\forall_i:\ w_i = 1/m$
2: **for** $m = 1, .., M$ **do**
3: Fit classifier G_m to the training data weighted by w_i
4: $\text{err}_m = \frac{\sum_{i=1}^n w_i\ I(y_i \neq G_m(x_i))}{\sum_{i=1}^n w_i}$
5: $\alpha_m = \log[\frac{1-\text{err}_m}{\text{err}_m}]$
6: $\forall_i:\ w_i \leftarrow w_i\ \exp[\alpha_m\ I(y_i \neq G_m(x_i))]$
7: **end for**

(Hastie, sec 10.1)

Weights unchanged for correctly classified points

Multiply weights with $\frac{1-\text{err}_m}{\text{err}_m} > 1$ for mis-classified data points

- *Real AdaBoost:* A variant exists that combines probabilistic classifiers $\sigma(f(x)) \in [0, 1]$ instead of discrete $G(x) \in \{-1, +1\}$

4:34

The basis function view

- In AdaBoost, each model G_m depends on the data weights w_m

  We could write this as

$$f(x) = \sum_{m=1}^M \alpha_m f_m(x, w_m)$$

  The "features" $f_m(x, w_m)$ now have additional parameters w_m

  We'd like to optimize

$$\min_{\alpha, w_1, .., w_M} L(f)$$

  w.r.t. α and all the feature parameters w_m.

- In general this is hard.

  But assuming $\alpha_{\hat{m}}$ and $w_{\hat{m}}$ fixed, optimizing for α_m and w_m is efficient.

- AdaBoost does exactly this, choosing w_m so that the "feature" f_m will best reduce the loss (cf. PLS)

  (Literally, AdaBoost uses exponential loss or neg-log-likelihood; Hastie sec 10.4 & 10.5)

4:35

Gradient Boosting

- AdaBoost generates a series of basis functions by using different data weightings w_m depending on so-far classification errors
- We can also generate a series of basis functions f_m by fitting them to the gradient of the so-far loss
- Assume we want to miminize some loss function

$$\min_f L(f) = \sum_{i=1}^n L(y_i, f(x_i))$$

We can solve this using gradient descent

$$f^* = f_0 + \alpha_1 \underbrace{\frac{\partial L(f_0)}{\partial f}}_{\approx f_1} + \alpha_2 \underbrace{\frac{\partial L(f_0 + \alpha_1 f_1)}{\partial f}}_{\approx f_2} + \alpha_3 \underbrace{\frac{\partial L(f_0 + \alpha_1 f_1 + \alpha_2 f_2)}{\partial f}}_{\approx f_3} + \dots$$

– Each f_m aproximates the so-far loss gradient

– Can use linear regression to choose α_m (instead of line search)

4:36

Gradient Boosting

- Hastie's book quite "likes" gradient boosting
  – Can be applied to any loss function
  – No matter if regression or classification (or CRFs)
  – Very good performance

  – Simpler, more general, better than AdaBoost

4:37

Decision Trees

4:38

Decision Trees

- So far we always referred to our "core learners" (Part I of this lecture)
- Decision trees are particularly used in Bagging and Boosting contexts
- We'll learn about
  – Boosted decision trees & stumps
  – Random Forests

4:39

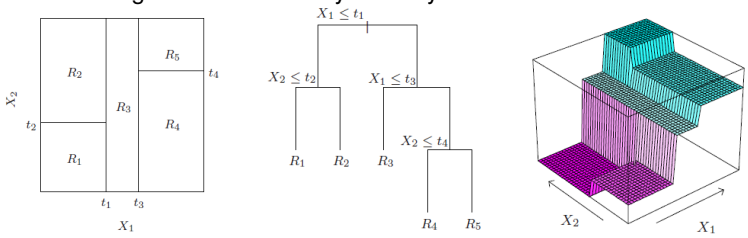
Decision Trees

- We describe CART (classification and regression tree)
- Decision trees are linear in features:

$$f(x) = \sum_{j=1}^k c_j \, I(x \in R_j)$$

where R_j are disjoint rectangular regions and c_j the constant prediction in a region

- The regions are defined by a binary decision tree



4:40

Growing the decision tree

- The constants are the region averages $c_j = \frac{\sum_i y_i \, I(x_i \in R_j)}{\sum_i I(x_i \in R_j)}$
- Each split $x_a > t$ is defined by a choice of feature (input dimension) $a \in \{1, .., d\}$ and a threshold t
- Given a yet unsplit region R_j, we split is by choosing

$$\min_{a,t} \Big[\min_{c_1} \sum_{i: x_i \in R_j \wedge x_a \le t} (y_i - c_1)^2 + \min_{c_2} \sum_{i: x_i \in R_j \wedge x_a > t} (y_i - c_2)^2 \Big]$$

– Finding the threshold t is really quick (slide along)

– We do this for every feature (input dimension) a

4:41

Growing the decision tree

- We first grow a very large tree (e.g. until a minimum region size of 5)
- Then we rank all nodes using "weakest link pruning":

  Iteratively remove the node that least increases

$$\sum_{i=1}^n (y_i - f(x_i))^2$$

- Use cross-validation to choose the eventual level of pruning

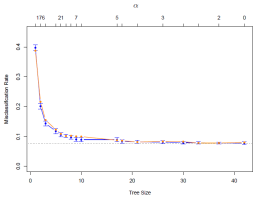
  This is equivalent to choosing a regularization parameter λ for

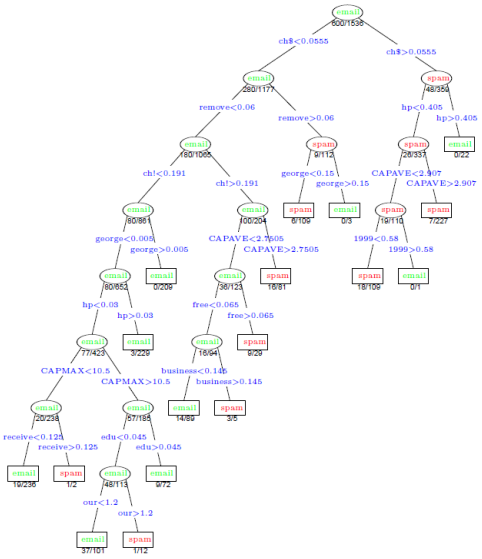
$$L(T) = \sum_{i=1}^n (y_i - f(x_i))^2 \; + \; \lambda |T|$$

where the regularization $|T|$ is the tree size

4:42

Example:
CART on the Spam data set
(details: Hastie, p 320)





	Predicted	
True	email	spam
email	57.3%	4.0%
spam	5.3%	33.4%

Test error rate: 8.7%

4:43

Boosting trees & stumps

- A **decision stump** is a decision tree with just one split
- Gradient boosting of decision trees and stumps is very effective

Test error rates on Spam data set:

full decision tree	8.7%
boosted decision stumps	4.7%
boosted decision trees with $J = 5$	4.5%

4:44

Random Forests: Bagging & randomized splits

- Recall that Bagging averages models $f_1, .., f_M$ where each f_m was trained on a bootstrap resample D_m of the data D

  This randomizes the models and avoids overgeneralization
- Random Forests do Bagging, but additionally randomize the trees:

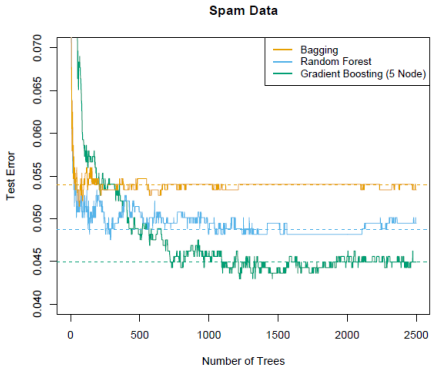
  – When growing a new split, choose the feature (input dimension) a

  only from a *random subset* m features

  – m is often very small; even $m = 1$ or $m = 3$

4:45

Random Forests vs. gradient boosted trees



(Hastie, Fig 15.1)

4:46

D. Ideas about other loss functions

– hinge loss, linear programming, SVMs

[The following slides of section D. are by Vien Ngo.]

4:47

Machine Learning
Part 2: The Breadth of ML ideas

Marc Toussaint

Machine Learning & Robotics Lab, U Stuttgart

03/06/2013

4:48

Table of contents

4:49

The Breadth of ML ideas

- A. Ideas about features & data preprocessing
  – centering & whitening
  – PCA
  – PLS (for classification?)
- B. Ideas about local learners
  – local & lazy learning
  – kNN
  – kd-trees
- C. Ideas about combining weak or randomized learners
  – Bootstrap, bagging, and model averaging
  – Boosting
  – (Boosted) decision trees & stumps
  – Random forests
- D. Ideas about other loss functions
  – hinge loss, linear programming, SVMs
- E. Ideas about deep learners

D. Ideas about other loss functions

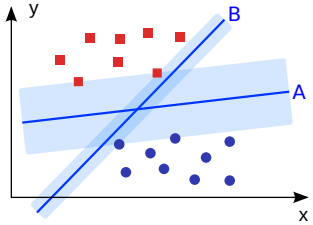
– hinge loss, linear programming, SVMs

Support Vector Machine

(see Hastie 12.3.2)

- binary linear classifier: $y \in \{-1, +1\}$
- SVM builds a hyperplane to separate two classes.
  A hyperplane is defined by means of β as $\{x \| f(x) = \phi(x)^\top \beta + \beta_0 = 0\}$. The separating hyperplane is linear in the feature space $\phi(x)$, but non-linear in the input space x.
- classification: $x \mapsto \text{sign}(\phi(x)^\top \beta + \beta_0)$
  (linear discriminative function like ridge regression)
- (Warning: offset β_0 requires special attention in kernel methods, but we ignore this issue in the following.)

Support Vector Machine



- why maximize the margin? (fat margin vs. thin margin)
- compute the margin? (x_k is the nearest point to the plane)

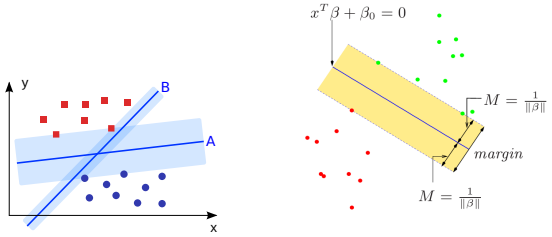
$$M = \frac{y_k(\phi(x_k)^\top \beta + \beta_0)}{\|\beta\|}$$

- maximize the margin?

$$\max_{\beta,\beta_0} \min_{x_k} \frac{y_k(\phi(x_k)^\top \beta + \beta_0)}{\|\beta\|}$$

Support Vector Machine (Case 1: linearly separable)

- normalize β: $|\phi(x_k)^\top \beta + \beta_0| = 1$



- can be rephrased as

$$\min_\beta \|\beta\| \quad \text{subject to } y_i(\phi(x_i)^\top \beta + \beta_0) \geq 1, \quad i = 1, \dots, n$$

Ridge regularization like ridge regression, but different loss

Support Vector Machine (Case 1: linearly separable)

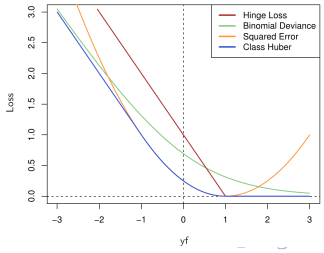
SVM as a Penalization Method

- **Difference** to ridge regression: Hinge loss
- SVM uses the *hinge loss* instead of neg-log-likelihood:

$$L^{\text{hinge}(\beta)} = \sum_{i=1}^n [1 - y_i \phi(x_i)^\top \beta - \beta_0]_+ + \lambda \|\beta\|^2$$

subscript $+$ indicates the positive part

- Hinge loss is zero if $f(x_i) = \phi(x_i)^\top \beta + \beta_0 \geq 1$ if $y_i = 1$ and $f(x_i) = \phi(x_i)^\top \beta + \beta_0 \leq -1$ if $y_i = -1$.



Support Vector Machine (Case 1: linearly separable)

The Lagrange (primal) function, to be minimized w.r.t. β and β_0 is

$$L(\beta, \beta_0) = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{N} \alpha_i [y_i(\phi(x_i)^\top \beta + \beta_0) - 1]$$

where $\alpha_i \geq 0$
Setting the derivatives to zero, we obtain:

$$\beta = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i)$$
$$0 = \sum_{i=1}^{N} \alpha_i y_i$$

4:56

Support Vector Machine (Case 1: linearly separable)

Substituing β and β_0, obtain a simpler convex optimization problem (dual)

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N}\sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k \phi(x_i)^\top \phi(x_k)$$
$$\text{such that } \alpha_i \geq 0, \text{ and } 0 = \sum_{i=1}^{N} \alpha_i y_i$$

4:57

E. Ideas about deep learners and representations

– First some comments on history of ML

– Very briefly: idea of deep NNs and literature

4:58

Free Access: http://link.springer.com/journal/10994/82/3/page/1

- Machine Learning Journal 25th Anniversary Issue (Vol. 82:3, 2011)

4:59

- Editorial by Pat Langley: *The changing science of machine learning*
- Discussion papers:

  Thomas G. Dietterich et al.: *Structured machine learning: the next ten years*

  Bengio, Yoshua and LeCun, Yann: *Scaling learning algorithms towards AI*

  Rodney Douglas et al.: *Future Challenges for the Science and Engineering of Learning*

  Pat Langley: *The changing science of machine learning*

4:60

Mach Learn (2008) 73: 3–23
DOI 10.1007/s10994-008-5079-1

Structured machine learning: the next ten years

Thomas G. Dietterich · Pedro Domingos · Lise Getoor · Stephen Muggleton · Prasad Tadepalli

Received: 25 October 2007 / Accepted: 2 February 2008 / Published online: 20 August 2008
Springer Science+Business Media, LLC 2008

Abstract The field of inductive logic programming (ILP) has made steady progress, since the first ILP workshop in 1991, based on a balance of developments in theory, implementations and applications. More recently there has been an increased emphasis on Probabilistic ILP and the related fields of Statistical Relational Learning (SRL) and Structured Prediction. The goal of the current paper is to consider these emerging trends and chart out the strategic directions and open problems for the broader area of structured machine learning for the next 10 years.

4:61

- My thoughts:

  1) ML is heathy, very successful, productive, exactly the right *foundation* to understand learning

  2) There are open issues w.r.t. ML for *intelligent (autonomous) behavior* → Autonomous Learning, Reinforcement Learning, Robotics, etc

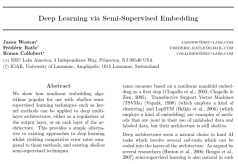
  3) There are open issues w.r.t. **representations** (e.g. discovering/developing abstractions, hierarchies, etc).

  Deep Learning and the revival of Neural Networks was driven by the latter.

4:62

Deep Learning

- Must read:


Deep Learning via Semi-Supervised Embedding

Jason Weston et al.: *Deep Learning via Semi-Supervised Embedding* (ICML 2008)

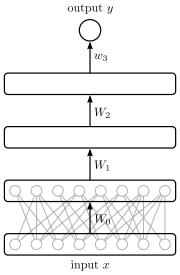
www.thespermwhale.com/jaseweston/papers/deep_embed.pdf

- Rough ideas:
  - Multi-layer NNs are a very "deep" function model
  - Only training on least squares gives too little "pressure" on inner representations
  - Mixing least squares cost with representational costs (cf. regularization) leads to better representations & generalization

4:63

Neural Networks

- Consider a regression problem with input $x \in \mathbb{R}^d$ and output $y \in \mathbb{R}$

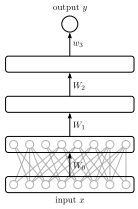


  - Linear function: $(w \in \mathbb{R}^d)$
    $$f(x) = w^\top x$$
  - Sigmoidal function:
    $$f(x) = \sigma(w^\top x)$$
  - 1-layer Neural Network function: $(W_0 \in \mathbb{R}^{h_1 \times d})$
    $$f(x) = w_1^\top \sigma(W_0 x)$$
  - 2-layer Neural Network function:
    $$f(x) = w_2^\top \sigma(W_1 \sigma(W_0 x))$$

- Neural Networks are a special function model $y = f(x, w)$, i.e. a special way to parameterize non-linear functions

4:64

Neural Networks



  - Usually trained based on the gradient $\frac{\partial}{\partial W_l} f(x)$
    (The output weights can be optimized analytically as for linear regression).
  - NNs are a very powerful function class
    By tweaking/training the weights one can approximate any non-linear function

- **BUT,** are there any guarantees on generalization?
  Is there any insight on what the neurons will actually represent? Why/How this should generalize to new data?
  The gradient resolves the "credit assignment" for errors, but
  - This is in practise not efficient for deep networks (the gradient "dissolves")
  - Is this the right thing? (Natural gradient? Other metric?)

  ⇒ *Make an explicit statement on what you want from internal representations! — How to regularize!*

4:65

Deep Learning

- In this perspective,

  1) Deep Learning considers deep function models (NNs usually)

  2) Deep Learning makes explicit statements about what internal representations should capture

- In Weston et al.'s case, for instance

$$L(\beta) = \underbrace{\sum_{i=1}^{n} (y_i - f(x_i))^2}_{\text{squared error}} + \lambda \underbrace{\sum_i \sum_{jk} (\|z_{ij} - z_{ik}\|^2 - d_{jk})^2}_{\text{Multidimensional Scaling regularization}}$$

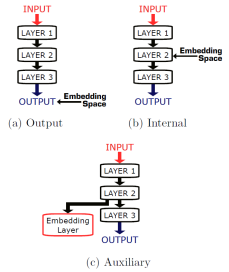
  where $z_i = \sigma(W_i z_{i\text{-}1})$ are the activations in the ith layer

  This mixes the squared error with a representational cost for each layer.

4:66

Deep Learning

- Results from Weston et al. (ICML, 2008)



Mnist1h dataset, deep NNs of 2, 6, 8, 10 and 15 layers; each hidden layer 50 hidden units

	2	4	6	8	10	15
NN	26.0	26.1	27.2	28.3	34.2	47.7
$Embed^O$NN	19.7	15.1	15.1	15.0	13.7	11.8
$Embed^{ALL}$NN	18.2	12.6	7.9	8.5	6.3	9.3

4:67

Deep Learning – further reading

- Weston, Ratle & Collobert: *Deep Learning via Semi-Supervised Embedding*, ICML 2008.
- Hinton & Salakhutdinov: *Reducing the Dimensionality of Data with Neural Networks*, Science 313, pp. 504-507, 2006.
- Bengio & LeCun: *Scaling Learning Algorithms Towards AI*. In Bottou et al. (Eds) “Large-Scale Kernel Machines”, MIT Press 2007.
- Hadsell, Chopra & LeCun: *Dimensionality Reduction by Learning an Invariant Mapping*, CVPR 2006.

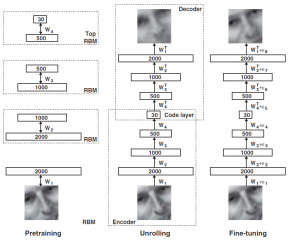
4:68

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such “autoencoder” networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer “encoder” network



4:69

The Breadth of ML ideas

- A. Ideas about features & data preprocessing
  – centering & whitening
  – PCA
  – PLS (for classification?)
- B. Ideas about local learners
  – local & lazy learning
  – kNN
  – kd-trees
- C. Ideas about combining weak or randomized learners
  – Bootstrap, bagging, and model averaging
  – Boosting
  – (Boosted) decision trees & stumps
  – Random forests
- D. Ideas about other loss functions
  – hinge loss, linear programming, SVMs
- E. Ideas about deep learners and representations

4:70

5 Bayes Basics

Bayes, probabilities, Bayes' theorem & examples

- So far:
  - Basic regression & classification methods:
    Features + Loss + Regularization & CV
  - All kinds of extensions & ideas to improve upon this
- Today: **Bayes**
  - A fully alternative framework to think about learning
  - A framework for modelling learning and inference problems in general
  - A framework to derive learning algorithms for specific models

5:1

The need for modelling

- Given a real world problem, translating it to a well-defined learning problem is non-trivial.
- The "framework" of plain regression/classification is rather restricted: input x, output y.
- Graphical models (probabilstic models with multiple random variables and dependencies) are a more general framework for modelling "problems"; regression & classification become a special case; Reinforcement Learning, decision making, unsupervised learning, but also language processing, image segmentation, are special cases.

5:2

Thomas Bayes (1702-1761)

Rev. T. Bayes

"Essay Towards Solving a Problem in the Doctrine of Chances"

- Addresses problem of *inverse probabilities*:
  Knowing the conditional probability of B given A, what is the conditional probability of A given B?
- Example:
  40% Bavarians speak dialect, only 1% of non-Bavarians speak (Bav.) dialect
  Given a random German that speaks non-dialect, is he Bavarian?
  (15% of Germans are Bavarian)

5:3

- "Inference" = Given some pieces of information (prior, observed variabes) what is the implication (the implied information, the posterior) on a non-observed variable

Learning as Inference

  - given pieces of information: data, assumed model, *prior* over β
  - non-observed variable: β

5:4

Probability Theory

- Why do we need probabilities?
  - Obvious: to express inherent stochasticity of the world (data)
- But beyond this: (also in a "deterministic world"):
  - lack of knowledge!
  - hidden (latent) variables
  - expressing *uncertainty*
  - expressing *information* (and lack of information)
- Probability Theory: an information calculus

5:5

Probability: Frequentist and Bayesian

- Frequentist probabilities are defined in the limit of an infinite number of trials
  Example: "The probability of a particular coin landing heads up is 0.43"
- Bayesian (subjective) probabilities quantify degrees of belief
  Example: "The probability of it raining tomorrow is 0.3"
  – Not possible to repeat "tomorrow"

5:6

Probabilities & Random Variables

- For a random variable X with discrete domain $\text{dom}(X) = \Omega$ we write:

  $\forall_{x\in\Omega}:\ 0 \le P(X=x) \le 1$

  $\sum_{x\in\Omega} P(X=x) = 1$

  Example: A dice can take values $\Omega = \{1, .., 6\}$.
  X is the random variable of a dice throw.
  $P(X=1) \in [0, 1]$ is the probability that X takes value 1.
- A bit more formally: a random variable relates a measureable space with a domain (sample space) and thereby introduces a probability measure on the domain ("assigns a probability to each possible value")

5:7

Probabilty Distributions

- $P(X=1) \in \mathbb{R}$ denotes a specific probability
  $P(X)$ denotes the probability distribution (function over Ω)

  Example: A dice can take values $\Omega = \{1,2,3,4,5,6\}$.
  By $P(X)$ we discribe the full distribution over possible values $\{1,..,6\}$. These are 6 numbers that sum to one, usually stored in a *table*, e.g.: $[\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}]$

- In implementations we typically represent distributions over discrete random variables as tables (arrays) of numbers

- Notation for summing over a RV:
  In equation we often need to sum over RVs. We then write
  $$\sum_X P(X) \ \cdots$$
  as shorthand for the explicit notation $\sum_{x \in \text{dom}(X)} P(X=x) \ \cdots$

5:8

Joint distributions

Assume we have *two* random variables X and Y

- Definitions:

  $P(X=x, Y=y)$

x				P_{xy}	

  y

  Joint: $P(X,Y)$

  Marginal: $P(X) = \sum_Y P(X,Y)$

  Conditional: $P(X|Y) = \frac{P(X,Y)}{P(Y)}$

  The conditional is normalized: $\forall_Y : \ \sum_X P(X|Y) = 1$

- X is *independent* of Y iff: $P(X|Y) = P(X)$
  (table thinking: all columns of $P(X|Y)$ are equal)

5:9

Joint distributions

joint: $P(X,Y)$
marginal: $P(X) = \sum_Y P(X,Y)$
conditional: $P(X|Y) = \frac{P(X,Y)}{P(Y)}$

- Implications of these definitions:

  product rule: $P(X,Y) = P(X|Y)\ P(Y) = P(Y|X)\ P(X)$

  Bayes' Theorem $P(X|Y) = \frac{P(Y|X)\ P(X)}{P(Y)}$

5:10

Bayes' Theorem

$$P(X|Y) = \frac{P(Y|X)\ P(X)}{P(Y)}$$

$$\text{posterior} \ = \ \frac{\text{likelihood} \cdot \text{prior}}{\text{normalization}}$$

5:11

Example 1: Bavarian dialect

- 40% Bavarians speak dialect, only 1% of non-Bavarians speak (Bav.) dialect

  Given a random German that speaks non-dialect, is he Bavarian?

  (15% of Germans are Bavarian)

  $P(D=1 \,|\, B=1) = 0.4$
  $P(D=1 \,|\, B=0) = 0.01$
  $P(B=1) = 0.15$

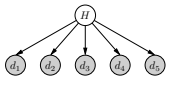
  If follows
  $P(B=1 \,|\, D=0) = \frac{P(D=0 \,|\, B=1)\ P(B=1)}{P(D=0)} = \frac{.6 \cdot .15}{.6 \cdot .15 + 0.99 \cdot .85} \approx 0.097$

5:12

Example 2: Coin flipping

HHTHT

HHHHH



- What process produces these sequences?

- We compare two hypothesis:
  $H=1$: fair coin $P(d_i = \text{H} \,|\, H=1) = \frac{1}{2}$
  $H=2$: always heads coin $P(d_i = \text{H} \,|\, H=2) = 1$

- Bayes' theorem:
  $$P(H \,|\, D) = \frac{P(D \,|\, H) P(H)}{P(D)}$$

5:13

Coin flipping

$D = \text{HHTHT}$

$P(D \,|\, H=1) = 1/2^5$ $\quad$ $P(H=1) = \frac{999}{1000}$
$P(D \,|\, H=2) = 0$ $\quad$ $P(H=2) = \frac{1}{1000}$

$$\frac{P(H=1\,|\,D)}{P(H=2\,|\,D)} = \frac{P(D\,|\,H=1)}{P(D\,|\,H=2)}\,\frac{P(H=1)}{P(H=2)} = \frac{1/32}{0}\,\frac{999}{1} = \infty$$

5:14

Coin flipping

$D = \texttt{HHHHH}$

$$P(D\,|\,H=1) = 1/2^5 \qquad P(H=1) = \tfrac{999}{1000}$$
$$P(D\,|\,H=2) = 1 \qquad P(H=2) = \tfrac{1}{1000}$$

$$\frac{P(H=1\,|\,D)}{P(H=2\,|\,D)} = \frac{P(D\,|\,H=1)}{P(D\,|\,H=2)}\,\frac{P(H=1)}{P(H=2)} = \frac{1/32}{1}\,\frac{999}{1} \approx 30$$

5:15

Coin flipping

$D = \texttt{HHHHHHHHHH}$

$$P(D\,|\,H=1) = 1/2^{10} \qquad P(H=1) = \tfrac{999}{1000}$$
$$P(D\,|\,H=2) = 1 \qquad P(H=2) = \tfrac{1}{1000}$$

$$\frac{P(H=1\,|\,D)}{P(H=2\,|\,D)} = \frac{P(D\,|\,H=1)}{P(D\,|\,H=2)}\,\frac{P(H=1)}{P(H=2)} = \frac{1/1024}{1}\,\frac{999}{1} \approx 1$$

5:16

Learning as Bayesian inference

- *Think big:*

$$P(\text{World}|\text{Data}) = \frac{P(\text{Data}|\text{World})\;P(\text{World})}{P(\text{Data})}$$

$P(\text{World})$ describes our prior over all possible worlds. Learning means to infer about the world we live in based on the data we have!

- In the context of regression, the "world" is the function $f(x)$

$$P(f|\text{Data}) = \frac{P(\text{Data}|f)\;P(f)}{P(\text{Data})}$$

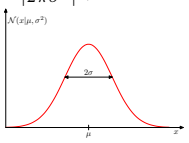
$P(f)$ describes our prior over possible functions

Regression means to infer the function based on the data we have

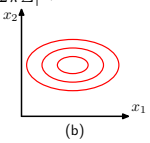
5:17

Gaussian distribution

- 1-dim: $\mathcal{N}(x\,|\,\mu,\sigma^2) = \frac{1}{|2\pi\sigma^2|^{1/2}}\; e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}$



- n-dim: $\mathcal{N}(x\,|\,\mu,\Sigma) = \frac{1}{|2\pi\Sigma|^{1/2}}\; e^{-\frac{1}{2}(x-\mu)^\top\,\Sigma^{-1}\,(x-\mu)} \propto e^{-\frac{1}{2}[x^\top\Sigma^{-1}x - 2x^\top\Sigma$



(b)

- continuous domain: *probability distribution* $F(x) = \int_{-\infty}^{x} dx\; p(x) \in [0,1]$ is the integral of a *probability density function* $p(x) \in [0,\infty)$
  discrete domain: *probability distribution* and *probability mass function* $P(X) \in [0,1]$ are used synonymously

5:18

6 Bayesian Regression & Classification

learning as inference, Bayesian Kernel Ridge regression = Gaussian Processes, Bayesian Kernel Logistic Regression = GP classification, Bayesian Neural Networks

Learning as Inference

- The parameteric view

$$P(\beta|\text{Data}) = \frac{P(\text{Data}|\beta)\ P(\beta)}{P(\text{Data})}$$

- The function space view

$$P(f|\text{Data}) = \frac{P(\text{Data}|f)\ P(f)}{P(\text{Data})}$$

- Today:

  – Bayesian (Kernel) Ridge Regression $\leftrightarrow$ Gaussian Process (GP)

  – Bayesian (Kernel) Logistic Regression $\leftrightarrow$ GP classification

  – Bayesian Neural Networks (briefly)

6:1

- Beyond learning about specific Bayesian learning methods:

  Understand relations between

  loss/error $\leftrightarrow$ neg-log likelihood

  regularization $\leftrightarrow$ neg-log prior

  cost (reg.+loss) $\leftrightarrow$ neg-log posterior

6:2

Ridge regression as Bayesian inference

- We have random variables $X_{1:n}, Y_{1:n}, \beta$
- We observe data $D = \{(x_i, y_i)\}_{i=1}^n$ and want to compute $P(\beta \mid D)$
- Let's assume:

  $P(X)$ is arbitrary

  $P(\beta)$ is Gaussian: $\beta \sim \mathcal{N}(0, \frac{\sigma^2}{\lambda}) \propto e^{-\frac{\lambda}{2\sigma^2}\|\beta\|^2}$

  $P(Y \mid X, \beta)$ is Gaussian: $y = x^\top\beta + \epsilon \ , \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$

6:3

Ridge regression as Bayesian inference

- Bayes' Theorem:

$$P(\beta \mid D) = \frac{P(D \mid \beta)\ P(\beta)}{P(D)}$$

$$P(\beta \mid x_{1:n}, y_{1:n}) = \frac{\prod_{i=1}^n P(y_i \mid \beta, x_i)\ P(\beta)}{Z}$$

$P(D \mid \beta)$ is a *product* of independent likelihoods for each observation (x_i, y_i)

Using the Gaussian expressions:

$$P(\beta \mid D) = \frac{1}{Z'} \prod_{i=1}^n e^{-\frac{1}{2\sigma^2}(y_i - x_i^\top\beta)^2}\ e^{-\frac{\lambda}{2\sigma^2}\|\beta\|^2}$$

$$-\log P(\beta \mid D) = \frac{1}{2\sigma^2}\Big[\sum_{i=1}^n (y_i - x_i^\top\beta)^2 + \lambda\|\beta\|^2\Big] - \log Z'$$

$$-\log P(\beta \mid D) \propto L^{\text{ridge}}(\beta)$$

1st insight: The *neg-log posterior* $P(\beta \mid D)$ is equal to the cost function $L^{\text{ridge}}(\beta)$!

6:4

Ridge regression as Bayesian inference

- Let us compute $P(\beta \mid D)$ explicitly:

$$\begin{aligned}
P(\beta \mid D) &= \frac{1}{Z'} \prod_{i=1}^n e^{-\frac{1}{2\sigma^2}\ (y_i - x_i^\top\beta)^2}\ e^{-\frac{\lambda}{2\sigma^2}\|\beta\|^2} \\
&= \frac{1}{Z'}\ e^{-\frac{1}{2\sigma^2}\ \sum_i (y_i - x_i^\top\beta)^2}\ e^{-\frac{\lambda}{2\sigma^2}\|\beta\|^2} \\
&= \frac{1}{Z'}\ e^{-\frac{1}{2\sigma^2}[(\boldsymbol{y}-\boldsymbol{X}\beta)^\top(\boldsymbol{y}-\boldsymbol{X}\beta) + \lambda\beta^\top\beta]} \\
&= \frac{1}{Z'}\ e^{-\frac{1}{2}[\frac{1}{\sigma^2}\boldsymbol{y}^\top\boldsymbol{y} + \frac{1}{\sigma^2}\beta^\top(\boldsymbol{X}^\top\boldsymbol{X} + \lambda\mathbf{I})\beta - \frac{2}{\sigma^2}\beta^\top\boldsymbol{X}^\top\boldsymbol{y}]} \\
&= \mathcal{N}(\beta \mid \hat\beta, \Sigma)
\end{aligned}$$

This is a Gaussian with covariance and mean

$$\Sigma = \sigma^2\ (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\mathbf{I})^{-1} \ , \quad \hat\beta = \tfrac{1}{\sigma^2}\ \Sigma\boldsymbol{X}^\top\boldsymbol{y} = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\mathbf{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$$

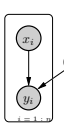
- **2nd insight:** The mean $\hat\beta$ is exactly the classical $\text{argmin}_\beta\ L^{\text{ridge}}(\beta)$.
- **3rd insight:** The Bayesian inference approach not only gives a mean/optimal $\hat\beta$, but also a variance Σ of that estimate!

6:5

Predicting with an uncertain β

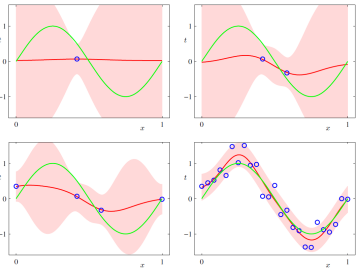
- Suppose we want to make a prediction at x. We can compute the **predictive distribution** over a new observation y^* at x^*:

$$\begin{aligned}
P(y^* \mid x^*, D) &= \textstyle\int_\beta P(y^* \mid x^*, \beta)\ P(\beta \mid D)\ d\beta \\
&= \textstyle\int_\beta \mathcal{N}(y^* \mid \phi(x^*)^\top\beta, \sigma^2)\ \mathcal{N}(\beta \mid \hat\beta, \Sigma)\ d\beta \\
&= \mathcal{N}(y^* \mid \phi(x^*)^\top\hat\beta,\ \sigma^2 + \phi(x^*)^\top\Sigma\phi(x^*))
\end{aligned}$$



Note $P(f(x) \mid D) = \mathcal{N}(f(x) \mid \phi(x)^\top\hat\beta,\ \phi(x)^\top\Sigma\phi(x))$ without the σ^2

- So, y^* is Gaussian distributed around the mean prediction $\phi(x^*)^\top \hat\beta$:



(from Bishop, p176)

6:6

Wrapup of Bayesian Ridge regression

- **1st insight:** The *neg-log posterior* $P(\beta \mid D)$ is equal to the cost function $L^{\text{ridge}}(\beta)$!

  This is a very very common relation: optimization costs correspond to neg-log probabilities; probabilities correspond to exp-neg costs.

- **2nd insight:** The mean $\hat\beta$ is exactly the classical $\text{argmin}_\beta\, L^{\text{ridge}}(\beta)$.

  More generally, the most likely parameter $\text{argmax}_\beta\, P(\beta|D)$ is also the least-cost parameter $\text{argmin}_\beta\, L(\beta)$. In the Gaussian case, mean and most-likely coincide.

- **3rd insight:** The Bayesian inference approach not only gives a mean/optimal $\hat\beta$, but also a variance Σ of that estimate!

  This is a core benefit of the Bayesian view: It naturally provides a probability distribution over predictions (*"error bars"*), not only a single prediction.

6:7

Kernelized Bayesian Ridge Regression

- As in the classical case, we can consider arbitrary features $\phi(x)$
- .. or directly use a kernel $k(x, x')$:

$$
\begin{aligned}
P(f(x) \mid D) &= \mathcal{N}(f(x) \mid \phi(x)^\top \hat\beta,\ \phi(x)^\top \Sigma \phi(x)) \\
\phi(x)^\top \hat\beta &= \phi(x)^\top \boldsymbol{X}^\top (\boldsymbol{X}\boldsymbol{X}^\top + \lambda \mathbf{I})^{-1} \boldsymbol{y} \\
&= \boldsymbol{\kappa}(x)(\boldsymbol{K} + \lambda \mathbf{I})^{-1} \boldsymbol{y} \\
\phi(x)^\top \Sigma \phi(x) &= \phi(x)^\top \sigma^2 \, (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \mathbf{I})^{-1} \phi(x) \\
&= \frac{\sigma^2}{\lambda} \phi(x)^\top \phi(x) - \frac{\sigma^2}{\lambda} \phi(x)^\top \boldsymbol{X} (\boldsymbol{X}\boldsymbol{X}^\top + \lambda \mathbf{I}_k)^{-1} \boldsymbol{X}^\top \phi(x) \\
&= \frac{\sigma^2}{\lambda} k(x, x) - \frac{\sigma^2}{\lambda} \boldsymbol{\kappa}(x)(\boldsymbol{K} + \lambda \mathbf{I}_n)^{-1} \boldsymbol{\kappa}(x)
\end{aligned}
$$

3rd line: As on slide 02:24
last lines: Woodbury identity $(A+UBV)^{-1} = A^{-1} - A^{-1}U(B^{-1}+VA^{-1}U)^{-1}VA^{-1}$ with $A = \lambda \mathbf{I}$

- In standard conventions $\lambda = \sigma^2$, $P(\beta) = \mathcal{N}(\beta|0, 1)$

  – Regularization: scale the covariance function (or features)

6:8

Kernelized Bayesian Ridge Regression

is equivalent to Gaussian Processes

(see also Welling: "Kernel Ridge Regression" Lecture Notes; Rasmussen & Williams sections 2.1 & 6.2; Bishop sections 3.3.3 & 6)

- As we have the equations alreay, I skip further math details. (See Rasmussen & Williams)

6:9

Gaussian Processes

- The function space view

$$P(f|\text{Data}) = \frac{P(\text{Data}|f)\ P(f)}{P(\text{Data})}$$

- Gaussian Processes define a probability distribution over functions:
  - A function is an infinite dimensional thing – how could we define a Gaussian distribution over functions?
  - For every finite set $\{x_1, .., x_M\}$, the function values $f(x_1), .., f(x_M)$ are Gaussian distributed with mean and cov.

$$
\begin{aligned}
\langle f(x_i) \rangle &= \mu(x_i) \qquad \text{(often zero)} \\
\langle [f(x_i) - \mu(x_i)][f(x_j) - \mu(x_j)] \rangle &= k(x_i, x_j)
\end{aligned}
$$

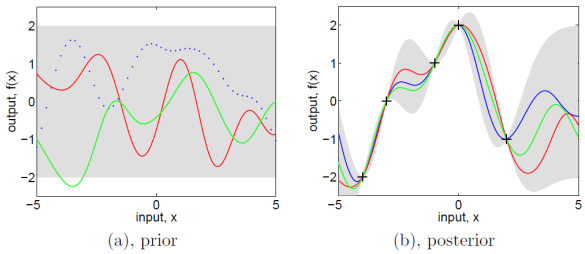
  Here, $k(\cdot,\cdot)$ is called **covariance function**

- Second, Gaussian Processes define an observation probability

$$P(y|x, f) = \mathcal{N}(y|f(x), \sigma^2)$$

6:10

Gaussian Processes

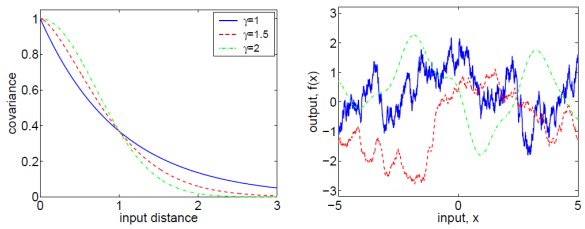


(a), prior (b), posterior

(from Rasmussen & Williams)

6:11

GP: different covariance functions



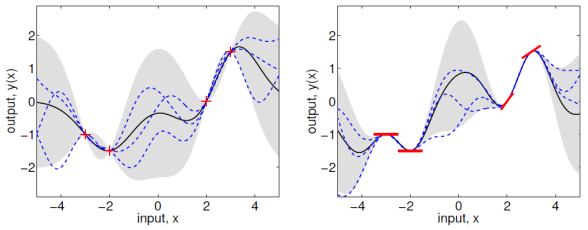
(from Rasmussen & Williams)

- These are examples from the γ-exponential covariance function

$$k(x, x') = \exp\{-|(x - x')/l|^\gamma\}$$

6:12

GP: derivative observations



(from Rasmussen & Williams)

6:13

- Bayesian Kernel Ridge Regression = Gaussian Process
- GPs have become a standard regression method
- If exact GP is not efficient enough, many approximations exist, e.g. sparse and pseudo-input GPs

6:14

Bayesian (Ridge) Logistic Regression

6:15

Bayesian Logistic Regression

- f now defines a logistic probability over $y \in \{0,1\}$:

$$P(X) = \text{arbitrary}$$
$$P(\beta) = \mathcal{N}(\beta|0, \frac{2}{\lambda}) \propto \exp\{-\lambda\|\beta\|^2\}$$
$$P(Y=1 \mid X, \beta) = \sigma(\beta^\top \phi(x))$$

- Recall

$$L^{\text{logistic}}(\beta) = -\sum_{i=1}^n \log p(y_i \mid x_i) + \lambda\|\beta\|^2$$

- Again, the parameter posterior is

$$P(\beta|D) \propto P(D \mid \beta)\ P(\beta) \propto \exp\{-L^{\text{logistic}}(\beta)\}$$

6:16

Bayesian Logistic Regression

- Use **Laplace approximation** (2nd order Taylor for L) at $\beta^* = \text{argmin}_\beta L(\beta)$:

$$L(\beta) \approx L(\beta^*) + \bar\beta^\top \nabla + \frac{1}{2}\bar\beta^\top H \bar\beta\ , \quad \bar\beta = \beta - \beta^*$$
$$P(\beta|D) \propto \exp\{-\bar\beta^\top \nabla - \frac{1}{2}\bar\beta^\top H \bar\beta\}$$
$$= \mathcal{N}[\bar\beta| - \nabla, H] = \mathcal{N}(\bar\beta| - H^{-1}\nabla, H^{-1})$$
$$= \mathcal{N}(\beta|\beta^*, H^{-1}) \qquad \text{(because } \nabla = 0 \text{ at } \beta^*)$$

- Then the predictive distribution of the *discriminative function* is also Gaussian!

$$P(f(x) \mid D) = \int_\beta P(f(x) \mid \beta)\ P(\beta \mid D)\ d\beta$$
$$= \int_\beta \mathcal{N}(f(x) \mid \phi(x)^\top\beta, 0)\ \mathcal{N}(\beta \mid \beta^*, H^{-1})\ d\beta$$
$$= \mathcal{N}(f(x) \mid \phi(x)^\top\beta^*, \phi(x)^\top H^{-1}\phi(x)) \ =:\ \mathcal{N}(f(x) \mid f^*, s^2)$$

- The predictive distribution over the label $y \in \{0,1\}$:

$$P(y(x)=1 \mid D) = \int_{f(x)} \sigma(f(x))\ P(f(x)|D)\ df$$
$$\approx \varphi(\sqrt{1 + s^2\pi/8}\, f^*)$$

the approximation replaced σ by the probit function $\varphi(x) = \int_{-\infty}^x \mathcal{N}(0,1)dx$.

6:17

Kernelized Bayesian Logistic Regression

- As with Kernel Logistic Regression, the MAP discriminative function f^* can be found iterating the Newton method $\leftrightarrow$ iterating GP estimation on a *re-weighted* data set.
- The rest is as above.

6:18

Kernel Bayesian Logistic Regression

is equivalent to Gaussian Process Classification

- GP classification became a standard classification method, if the prediction needs to be a meaningful probability that takes the *model uncertainty* into account.

6:19

Bayesian Neural Networks

6:20

General non-linear models

- Above we always assumed $f(x) = \phi(x)^\top\beta$ (or kernelized)
- Bayesian Learning also works for non-linear function models $f(x,\beta)$
- Regression case:

  $P(X)$ is arbitrary.

  $P(\beta)$ is Gaussian: $\beta \sim \mathcal{N}(0, \frac{\sigma^2}{\lambda}) \propto e^{-\frac{\lambda}{2\sigma^2}\|\beta\|^2}$

  $P(Y \mid X, \beta)$ is Gaussian: $y = f(x,\beta) + \epsilon\ , \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$

6:21

General non-linear models

- To compute $P(\beta|D)$ we first compute the most likely

$$\beta^* = \underset{\beta}{\text{argmin}}\, L(\beta) = \underset{\beta}{\text{argmax}}\, P(\beta|D)$$

- Use Laplace approximation around β^*: 2nd-order Taylor of $f(x,\beta)$ and then of $L(\beta)$ to estimate a Gaussian $P(\beta|D)$
- Neural Networks:

  – The Gaussian prior $P(\beta) = \mathcal{N}(\beta|0, \frac{\sigma^2}{\lambda})$ is called **weight decay**

  – This pushes "sigmoids to be in the linear region".

6:22

Conclusions

- Probabilistic inference is a very powerful concept!
  - Inferring about the world given data
  - Learning, decision making, reasoning can view viewed as forms of (probabilistic) inference
- We introduced Bayes' Theorem as the fundamental form of probabilistic inference
- Marrying Bayes with (Kernel) Ridge (Logisic) regression yields
  - Gaussian Processes
  - Gaussian Process classification

6:23

7 Graphical Models

Bayesian Networks, conditional independence, examples

The need for modelling

- Given a real world problem, translating it to a well-defined learning problem is non-trivial.
- The "framework" of plain regression/classification is rather restricted: input x, output y.
- Graphical models (probabilstic models with multiple random variables and dependencies) are a more general framework for modelling "problems"; regression & classification become a special case; Reinforcement Learning, decision making, but also language processing, image segmentation, are special cases.

7:1

Graphical Models

- The core difficulty in modelling is specifying

  What are the relevant variables?

  How do they depend on each other?

  (Or how *could* they depend on each other → learning)
- **Graphical models** are a simple, graphical notation for

  1) which random variables exist

  2) which random variables are "directly coupled"

  Thereby they *describe a joint probability distribution* $P(X_1, .., X_n)$ over n random variables.
- 2 basic variants:

  – Bayesian Networks (aka. directed model, belief network)

  – Factor Graphs (aka. undirected model, Markov Random Field)

7:2

Bayesian Networks

- A **Bayesian Network** is a

  – directed acyclic graph (DAG)

  – where each node represents a random variable X_i

  – for each node we have a conditional probability distribution

  $$P(X_i \,|\, \text{Parents}(X_i))$$
- In the simplest case (discrete RVs), the conditional distribution is represented as a conditional probability table (**CPT**)

7:3

Example

drinking red wine → longevity?

7:4

Bayesian Networks

- DAG → we can sort the RVs; edges only go from lower to higher index
- **The joint distribution can be factored as**

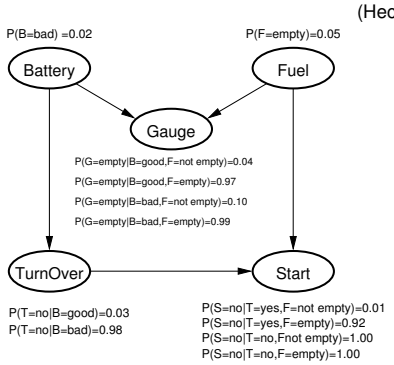
$$P(X_{1:n}) = \prod_{i=1}^{n} P(X_i \,|\, \text{Parents}(X_i))$$

- Missing links imply conditional independence
- Ancestral simulation to sample from joint distribution

7:5

Example

(Heckermann 1995)



$$\Longleftrightarrow \; P(S,T,G,F,B) = P(B)\, P(F)\, P(G|F,B)\, P(T|B)\, P(S|T,F)$$

- Table sizes: LHS = $2^5 - 1 = 31$ RHS = $1 + 1 + 4 + 2 + 4 = 12$

7:6

Constructing a Bayes Net

1. Choose a relevant set of variables X_i that describe the domain
2. Choose an ordering for the variables
3. While there are variables left

   (a) Pick a variable X_i and add it to the network

   (b) Set Parents(X_i) to some minimal set of nodes already in the net

   (c) Define the CPT for X_i

7:7

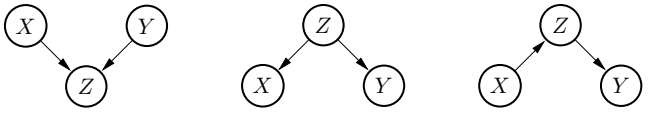
- This procedure is guaranteed to produce a DAG
- Different orderings may lead to **different Bayes nets representing the same joint distribution**:
- To ensure maximum sparsity choose a wise order ("root causes"first) Counter example: construct DAG for the car example using the ordering S, T, G, F, B

- "Wrong" ordering will give same joint distribution, but will require the specification of more numbers than otherwise necessary

7:8

Bayes Nets & conditional independence

- Independence: $Indep(X,Y) \iff P(X,Y) = P(X)\ P(Y)$
- Conditional independence:
  $Indep(X,Y|Z) \iff P(X,Y|Z) = P(X|Z)\ P(Y|Z)$



(head-to-head) (tail-to-tail) (head-to-tail)

$Indep(X,Y)$	$\neg Indep(X,Y)$	$\neg Indep(X,Y)$
$\neg Indep(X,Y\|Z)$	$Indep(X,Y\|Z)$	$Indep(X,Y\|Z)$

7:9

- Head-to-head: $Indep(X,Y)$
  $P(X,Y,Z) = P(X)\ P(Y)\ P(Z|X,Y)$
  $P(X,Y) = P(X)\ P(Y)\ \sum_Z P(Z|X,Y) = P(X)\ P(Y)$

- Tail-to-tail: $Indep(X,Y|Z)$
  $P(X,Y,Z) = P(Z)\ P(X|Z)\ P(Y|Z)$
  $P(X,Y|Z) = P(X,Y,Z) = P(Z) = P(X|Z)\ P(Y|Z)$

- Head-to-tail: $Indep(X,Y|Z)$
  $P(X,Y,Z) = P(X)\ P(Z|X)\ P(Y|Z)$
  $P(X,Y|Z) = \frac{P(X,Y,Z)}{P(Z)} = \frac{P(X,Z)\ P(Y|Z)}{P(Z)} = P(X|Z)\ P(Y|Z)$

7:10

General rules for determining conditional independence in a Bayes net:

- Given three groups of random variables X, Y, Z

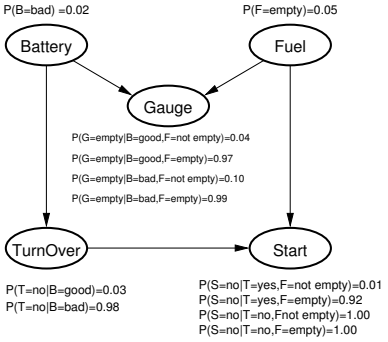
$Indep(X,Y|Z) \iff$ every path from X to Y is "blocked by Z"

- A path is "blocked by Z" $\iff$
  – $\exists$ a node in Z that is head-to-tail w.r.t. the path, or
  – $\exists$ a node in Z that is tail-to-tail w.r.t. the path, or
  – $\exists$ another node A which is head-to-head w.r.t. the path and neither A nor any of its descendants are in Z

7:11

Example

(Heckermann 1995)



$Indep(T,F)$? $Indep(B,F|S)$? $Indep(B,S|T)$?

7:12

What can we do with Bayes nets?

- **Inference:** Given some pieces of information (prior, observed variabes) what is the implication (the implied information, the posterior) on a non-observed variable

- **Learning:**
  – Fully Bayesian Learning: Inference over parameters (e.g., β)
  – Maximum likelihood training: Optimizing parameters

- **Structure Learning** (Learning/Inferring the graph structure itself): Decide which model (which graph structure) fits the data best; thereby uncovering conditional independencies in the data.

7:13

Inference

- Inference: Given some pieces of information (prior, observed variabes) what is the implication (the implied information, the posterior) on a non-observed variable

- In a Bayes Nets: Assume there is three groups of RVs:
  – Z are observed random variables
  – X and Y are hidden random variables
  – We want to do inference about X, not Y

Given some observed variables Z, compute the **posterior marginal** $P(X \mid Z)$ for some hidden variable X.

$$P(X \mid Z) = \frac{P(X,Z)}{P(Z)} = \frac{1}{P(Z)} \sum_Y P(X,Y,Z)$$

where Y are all hidden random variables except for X

- Inference requires summing over *(eliminating)* hidden variables.

7:14

Example: Holmes & Watson

- Mr. Holmes lives in Los Angeles. One morning when Holmes leaves his house, he realizes that his grass is wet. Is it due to rain, or has he forgotten to turn off his sprinkler?

  – Calculate $P(R|H)$, $P(S|H)$ and compare these values to the prior probabilities.
  – Calculate $P(R,S|H)$.
  Note: R and S are marginally independent, but conditionally dependent

- Holmes checks Watson's grass, and finds it is also wet.

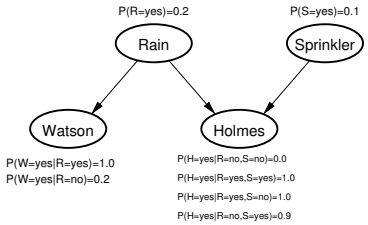
  – Calculate $P(R|H,W)$, $P(S|H,W)$
  – This effect is called explaining away

JavaBayes: run it from the html page

http://www.cs.cmu.edu/~javabayes/Home/applet.html

7:15

Example: Holmes & Watson



$$P(H,W,S,R) = P(H|S,R)\ P(W|R)\ P(S)\ P(R)$$

$$P(R|H) = \sum_{W,S} \frac{P(R,W,S,H)}{P(H)} = \frac{1}{P(H)} \sum_{W,S} P(H|S,R)\ P(W|R)\ P(S)\ P(R)$$
$$= \frac{1}{P(H)} \sum_{S} P(H|S,R)\ P(S)\ P(R)$$

$$P(R{=}1\,|\,H{=}1) = \frac{1}{P(H{=}1)}(1.0 \cdot 0.2 \cdot 0.1 + 1.0 \cdot 0.2 \cdot 0.9) = \frac{1}{P(H{=}1)} 0.2$$

$$P(R{=}0\,|\,H{=}1) = \frac{1}{P(H{=}1)}(0.9 \cdot 0.8 \cdot 0.1 + 0.0 \cdot 0.8 \cdot 0.9) = \frac{1}{P(H{=}1)} 0.072$$

7:16

- These types of calculations can be automated

  → Variable Elimination Algorithm

7:17

Example: Bavarian dialect

- Two binary random variables(RVs): B (bavarian) and D (dialect)
- Given:

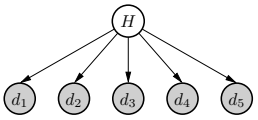
  $P(D,B) = P(D\,|\,B)\ P(B)$

  $P(D{=}1\,|\,B{=}1) = 0.4,\ P(D{=}1\,|\,B{=}0) = 0.01,\ P(B{=}1) = 0.15$

- **Notation:** Grey shading usually indicates "observed"

7:18

Example: Coin flipping



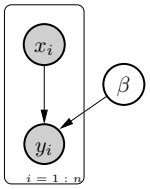
- One binary RV H (hypothesis), 5 RVs for the coin tosses $d_1, .., d_5$
- Given:

  $P(D,H) = \prod_i P(d_i\,|\,H)\ P(H)$

  $P(H{=}1) = \frac{999}{1000}$, $P(d_i{=}\mathtt{H}\,|\,H{=}1) = \frac{1}{2}$, $P(d_i{=}\mathtt{H}\,|\,H{=}2) = 1$

7:19

Example: Ridge regression



- One multi-variate RV β, $2n$ RVs $x_{1:n}$, $y_{1:n}$ (observed data)
- Given:

  $P(D,\beta) = \prod_i \Big[P(y_i\,|\,x_i,\beta)\ P(x_i)\Big]\ P(\beta)$

  $P(\beta) = \mathcal{N}(\beta\,|\,0, \frac{\sigma^2}{\lambda})$, $P(y_i\,|\,x_i,\beta) = \mathcal{N}(y_i\,|\,x_i^\top\beta, \sigma^2)$

- **Plate notation:** Plates (boxes with index ranges) mean "copy n-times"

7:20

8 Inference in Graphical Models

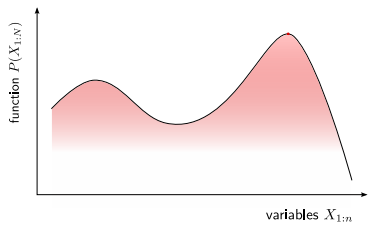
Sampling methods (Rejection, Importance, Gibbs), Variable Elimination, Factor Graphs, Message passing, Loopy Belief Propagation, Junction Tree Algorithm

Common inference methods in graphical models

- **Sampling:**
  - Rejection samping, importance sampling, Gibbs sampling
  - More generally, Markov-Chain Monte Carlo (MCMC) methods
- **Message passing:**
  - Exact inference on trees (includes the Junction Tree Algorithm)
  - Belief propagation
- **Other approximations/variational methods**
  - Expectation propagation
  - Specialized variational methods depending on the model
- **Reductions:**
  - Mathematical Programming (e.g. LP relaxations of MAP)
  - Compilation into Arithmetic Circuits (Darwiche at al.)

8:1

Probabilistic Inference & Optimization



- Note the relation of the Boltzmann distribution $p(x) \propto e^{-f(x)/T}$ with the cost function (energy) $f(x)$ and temperature T.

8:2

Sampling

- Read Andrieu et al: *An Introduction to MCMC for Machine Learning* (Machine Learning, 2003)
- Here I'll discuss only thee basic methods:
  - Rejection sampling
  - Importance sampling
  - Gibbs sampling

8:3

Rejection Sampling

- We have a Bayesian Network with RVs $X_{1:n}$, some of which are observed: $X_{obs} = y_{obs}$, $obs \subset \{1:n\}$
- The goal is to compute marginal posteriors $P(X_i \,|\, X_{obs} = y_{obs})$ conditioned on the observations.
- Our strategy is to generate a set of K (joint) samples of all variables

$$\mathcal{S} = \{(x_1^k, x_2^k, .., x_n^k)\}_{k=1}^K$$

Note: Each sample $x_{1:n}^k$ is a list of instantiation of all RVs.

8:4

Rejection Sampling

- To generate a single sample $x_{1:n}^k$:
  1. Sort all RVs in topological order; start with $i = 1$
  2. Sample a value $x_i^k \sim P(X_i \,|\, x_{\text{Parents}(i)}^k)$ for the ith RV conditional to the previous samples $x_{1:i-1}^k$
  3. If $i \in obs$ compare the sampled value x_i^k with the observation y_i. *Reject* and repeat from a) if the sample is not equal to the observation.
  4. Repeat with $i \leftarrow i + 1$ from 2.

8:5

Rejection Sampling

- Since computers are fast, we can sample for large K
- The sample set

$$\mathcal{S} = \{(x_1^k, x_2^k, .., x_n^k)\}_{k=1}^K$$

implies the necessary information:

– We can compute the marginal probabilities from these statistics:

$$P(X_i = x \,|\, X_{obs} = y_{obs}) \approx \frac{\text{count}_{\mathcal{S}}(x_i^k = x)}{K}$$

– or pair-wise marginals:

$$P(X_i = x, X_j = x' \,|\, X_{obs} = y_{obs}) \approx \frac{\text{count}_{\mathcal{S}}(x_i^k = x \wedge x_j^k = x')}{K}$$

– etc

8:6

Importance sampling (with likelihood weighting)

- Rejecting whole samples may become very inefficient in large Bayes Nets!

- New strategy: We generate a **weighted** sample set

$$\mathcal{S} = \{(w^k, x_1^k, x_2^k, .., x_n^k)\}_{k=1}^K$$

where each samples $x_{1:n}^k$ is associated with a weight w^k

- In our case, we will choose the weights proportional to the likelihood $P(X_{obs} = y_{obs} \,|\, X_{1:n} = x_{1:n}^k)$ of the observations conditional to the sample $x_{1:n}^k$

8:7

Importance sampling

- To generate a single sample $(w^k, x_{1:n}^k)$:
  1. Sort all RVs in topological order; start with $i = 1$ and $w^k = 1$
  2. a) If $i \notin obs$, sample a value $x_i^k \sim P(X_i \,|\, x_{\text{Parents}(i)}^k)$ for the ith RV conditional to the previous samples $x_{1:i\text{-}1}^k$
     b) If $i \in obs$, set the value $x_i^k = y_i$ and update the weight according to likelihood

$$w^k \leftarrow w^k \; P(X_i = y_i \,|\, x_{1:i\text{-}1}^k)$$

  3. Repeat with $i \leftarrow i + 1$ from 2.

8:8

Importance Sampling (with likelihood weighting)

- From the weighted sample set

$$\mathcal{S} = \{(w^k, x_1^k, x_2^k, .., x_n^k)\}_{k=1}^K$$

– we can compute the marginal probabilities:

$$P(X_i = x \,|\, X_{obs} = y_{obs}) \approx \frac{\sum_{k=1}^K w^k [x_i^k = x]}{\sum_{k=1}^K w^k}$$

– and likewise pair-wise marginals, etc

Notation: $[\text{expr}] = 1$ if $expr$ is true and zero otherwise

8:9

Gibbs sampling

- In Gibbs sampling we also generate a sample set $\mathcal{S}$ – but in this case the samples are not independent from each other anymore. The next sample "modifies" the previous one:
- First, all observed RVs are *clamped* to their fixed value $x_i^k = y_i$ for any k.
- To generate the $(k+1)$th sample, iterate through the latent variables $i \notin obs$, updating:

$$\begin{aligned} x_i^{k+1} &\sim P(X_i \,|\, x_{1:n\setminus i}^k) \\ &\sim P(X_i \,|\, x_1^k, x_2^k, .., x_{i\text{-}1}^k, x_{i+1}^k, .., x_n^k) \\ &\sim P(X_i \,|\, x_{\text{Parents}(i)}^k) \prod_{j : i \in \text{Parents}(j)} P(X_j = x_j^k \,|\, X_i, x_{\text{Parents}(j)\setminus i}^k) \end{aligned}$$

That is, each x_i^{k+1} is *resampled* conditional to the other (neighboring) current sample values.

8:10

Gibbs sampling

- As for rejection sampling, Gibbs sampling generates an unweighted sample set $\mathcal{S}$ which can directly be used to compute marginals.
  In practice, one often discards an initial set of samples (burn-in) to avoid starting biases.

- Gibbs sampling is a special case of **MCMC sampling.**
  Roughly, MCMC means to invent a sampling process, where the next sample may stochastically depend on the previous (Markov property), *such that* the final sample set is guaranteed to correspond to $P(X_{1:n})$.
  → *An Introduction to MCMC for Machine Learning*

8:11

Sampling – conclusions

- Sampling algorithms are very simple, very general and very popular
  – they equally work for continuous & discrete RVs
  – one only needs to ensure/implement the ability to sample from conditional distributions, no further algebraic manipulations
  – MCMC theory can reduce required number of samples

- In many cases exact and more efficient approximate inference is possible by actually computing/manipulating whole distributions in the algorithms instead of only samples.

8:12

Variable Elimination

8:13

Variable Elimination example

$$\begin{aligned} &P(x_5) \\ &= \textstyle\sum_{x_1,x_2,x_3,x_4,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_4|x_2)\, P(x_5|x_3)\, P(x_6|x_2, \\ &= \textstyle\sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5) \textstyle\sum_{x_4} \underbrace{P(}_{F_1} \\ &= \textstyle\sum_{x_1,x_2,x_3,x_6} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, P(x_6|x_2,x_5)\, \mu_1(x_2) \\ &= \textstyle\sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3) \textstyle\sum_{x_6} \underbrace{P(x_6|x_2,x_5)}_{F_2(x_2,x_5,x_6)} \\ &= \textstyle\sum_{x_1,x_2,x_3} P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)\, P(x_5|x_3)\, \mu_2(x_2,x_5) \\ &= \textstyle\sum_{x_2,x_3} P(x_5|x_3)\, \mu_2(x_2,x_5) \textstyle\sum_{x_1} \underbrace{P(x_1)\, P(x_2|x_1)\, P(x_3|x_1)}_{F_3(x_1,x_2,x_3)} \\ &= \textstyle\sum_{x_2,x_3} P(x_5|x_3)\, \mu_2(x_2,x_5)\, \mu_3(x_2,x_3) \\ &= \textstyle\sum_{x_3} P(x_5|x_3) \textstyle\sum_{x_2} \underbrace{\mu_2(x_2,x_5)\, \mu_3(x_2,x_3)}_{F_4(x_3,x_5)} \end{aligned}$$

$= \sum_{x_3} P(x_5|x_3)\ \mu_4(x_3, x_5)$
$= \sum_{x_3} \underbrace{P(x_5|x_3)\ \mu_4(x_3, x_5)}_{F_5(x_3, x_5)}$
$= \mu_5(x_5)$

8:14

Variable Elimination example – lessons learnt

- There is a dynamic programming principle behind Variable Elimination:
  – For eliminating $X_{5,4,6}$ we use the solution of eliminating $X_{4,6}$
  – The "sub-problems" are represented by the F terms, their solutions by the *remaining* μ *terms*
  – We'll continue to discuss this 4 slides later!

- The factorization of the joint
  – determines in which order Variable Elimination is efficient
  – determines what the terms $F(...)$ and $\mu(...)$ depend on

- We can automate Variable Elimination. For the automation, all that matters is the factorization of the joint.

8:15

Factor graphs

- In the previous slides we introduces the box ■ notation to indicate *terms* that depend on some variables. That's exactly what factor graphs represent.

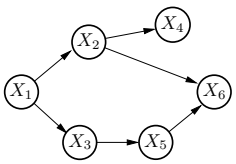
- A **Factor graph** is a
  – bipartite graph
  – where each circle node represents a random variable X_i
  – each box node represents a **factor** f_k, which is a function $f_k(X_{\partial k})$
  – the joint probability distribution is given as

$$P(X_{1:n}) = \prod_{k=1}^{K} f_k(X_{\partial k})$$

Notation: ∂k is shorthand for Neighbors(k)

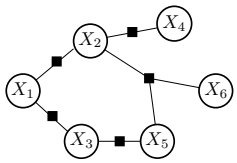
8:16

Bayes Net $\rightarrow$ factor graph



- Bayesian Network:

$P(x_{1:6}) = P(x_1)\ P(x_2|x_1)\ P(x_3|x_1)\ P(x_4|x_2)\ P(x_5|x_3)\ P(x_6|x_2, x_5)$



- Factor Graph:

$P(x_{1:6}) = f_1(x_1, x_2)\ f_2(x_3, x_1)\ f_3(x_2, x_4)\ f_4(x_3, x_5)\ f_5(x_2, x_5, x_6)$

$\rightarrow$ each CPT in the Bayes Net is just a factor (we neglect the special semantics of a CPT)

8:17

Variable Elimination Algorithm

- `eliminate_single_variable`(F, i)

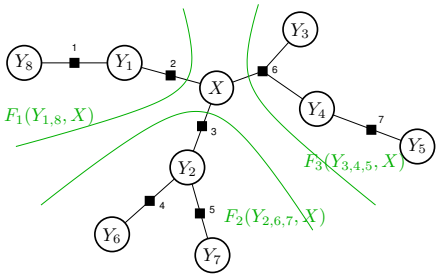
1: **Input:** list F of factors, variable id i
2: **Output:** list F of factors
3: find relevant subset $\hat{F} \subseteq F$ of factors coupled to i: $\hat{F} = \{k : i \in \partial k\}$
4: create new factor $\hat{k}$ with neighborhood $\partial\hat{k}$ = all variables in $\hat{F}$ except i
5: compute $\mu_{\hat{k}}(X_{\partial\hat{k}}) = \sum_{X_i} \prod_{k\in\hat{F}} f_k(X_{\partial k})$
6: remove old factors $\hat{F}$ and append new factor $\mu_{\hat{k}}$ to F
7: return F

- `elimination_algorithm`(μ, F, M)

1: **Input:** list F of factors, tuple M of desired output variables ids
2: **Output:** single factor μ over variables X_M
3: define all variables present in F: $V = \text{vars}(F)$
4: define variables to be eliminated: $E = V \setminus M$
5: for all $i \in E$: `eliminate_single_variable`(F, i)
6: for all remaining factors, compute the product $\mu = \prod_{f\in F} f$
7: return μ

8:18

Variable Elimination on trees



The subtrees w.r.t. X can be described as

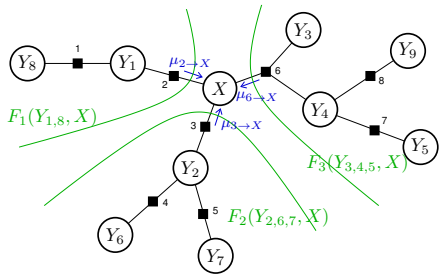
$F_1(Y_{1,8}, X) = f_1(Y_8, Y_1)\ f_2(Y_1, X)$
$F_2(Y_{2,6,7}, X) = f_3(X, Y_2)\ f_4(Y_2, Y_6)\ f_5(Y_2, Y_7)$
$F_3(Y_{3,4,5}, X) = f_6(X, Y_3, Y_4)\ f_7(Y_4, Y_5)$

The joint distribution is:

$P(Y_{1:8}, X) = F_1(Y_{1,8}, X)\ F_2(Y_{2,6,7}, X)\ F_3(Y_{3,4,5}, X)$

8:19

Variable Elimination on trees



We can eliminate each tree independently. The remaining terms (**messages**) are:

$$\mu_{F_1\to X}(X) = \textstyle\sum_{Y_{1,8}} F_1(Y_{1,8}, X)$$
$$\mu_{F_2\to X}(X) = \textstyle\sum_{Y_{2,6,7}} F_2(Y_{2,6,7}, X)$$
$$\mu_{F_3\to X}(X) = \textstyle\sum_{Y_{3,4,5}} F_3(Y_{3,4,5}, X)$$

The marginal $P(X)$ is the **product of subtree messages**

$$P(X) = \mu_{F_1\to X}(X)\, \mu_{F_2\to X}(X)\, \mu_{F_3\to X}(X)$$

8:20

Variable Elimination on trees – lessons learnt

- The "remaining terms" μ's are called **messages**

  Intuitively, **messages subsume information from a subtree**

- Marginal = product of messages, $P(X) = \prod_k \mu_{F_k\to X}$, is very intuitive:

  – *Fusion of independent information* from the different subtrees

  – Fusing independent information $\leftrightarrow$ multiplying probability tables

- Along a (sub-) tree, messages can be computed recursively

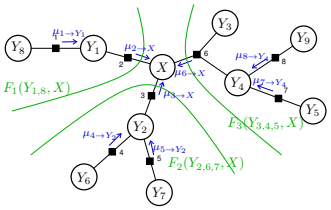
8:21

Message passing

- General equations (**belief propagation (BP)**) for recursive message computation (writing $\mu_{k\to i}(X_i)$ instead of $\mu_{F_k\to X}(X)$):

$$\mu_{k\to i}(X_i) = \sum_{X_{\partial k\setminus i}} \underbrace{f_k(X_{\partial k}) \prod_{j\in\partial k\setminus i} \bar\mu_{j\to k}(X_j)}_{F\text{(subtree)}} \quad \text{(factor-to-variable)}$$

where $\bar\mu_{j\to k}(X_j) = \prod_{k'\in\partial j\setminus k} \mu_{k'\to j}(X_j)$ (variable-to-factor)



Example messages:

$\mu_{2\to X} = \sum_{Y_1} f_2(Y_1, X)\, \mu_{1\to Y_1}(Y_1)$

$\mu_{6\to X} = \sum_{Y_3,Y_4} f_6(Y_3, Y_4, X)\, \mu_{7\to Y_4}(Y_4)$

$\mu_{3\to X} = \sum_{Y_2} f_3(Y_2, X)\, \mu_{4\to Y_2}(Y_2)\, \mu_{5\to Y_2}(Y_2)$

8:22

Message passing remarks

- Computing these messages recursively on a tree does nothing else than Variable Elimination.

  $\Rightarrow P(X_i) = \prod_{k\in\partial i} \mu_{k\to i}(X_i)$ is the correct posterior marginal

- However, since it stores all "intermediate terms", we can compute ANY marginal $P(X_i)$ for any i.

- Message passing exemplifies how to exploit the factorization structure of the joint distribution for the algorithmic implementation

- Note: These are recursive equations. They can be resolved exactly if and only if the dependency structure (factor graph) is a tree. If the factor graph had loops, this would be a "loopy recursive equation system"...

8:23

Message passing variants

- Message passing has many important applications:

  – Many models are actually trees: In particular chains

  esp. *Hidden Markov Models*

  – Message passing can also be applied on non-trees ($\leftrightarrow$ loopy graphs)

  $\rightarrow$ approximate inference *(Loopy Belief Propagation)*

  – Bayesian Networks can be "squeezed" to become trees

  $\rightarrow$ exact inference in Bayes Nets! *(Junction Tree Algorithm)*

8:24

Loopy Belief Propagation

- If the graphical model is not a tree (=has loops):

  – The recursive message equations cannot be resolved.

  – However, we could try to just iterate them as update equations...

- Loopy BP update equations: (initialize with $\mu_{k\to i} = 1,\ \mu_{j\to k} = 1$)

$$\mu^{\text{new}}_{k\to i}(X_i) = \sum_{X_{\partial k\setminus i}} f_k(X_{\partial k}) \prod_{j\in\partial k\setminus i} \mu^{\text{old}}_{j\to k}(X_j) \quad \text{(factor-to-variable)}$$

$$\mu^{\text{new}}_{j\to k}(X_j) = \prod_{k'\in\partial j\setminus k} \mu^{\text{old}}_{k'\to j}(X_j) \quad \text{(variable-to-factor)}$$

8:25

Loopy BP remarks

- Problem of loops intuitively:
  loops $\Rightarrow$ branches of a node to not represent independent information!
  – BP is multiplying (=fusing) messages from dependent sources of information

- No convergence guarantee, but if it converges, then to a state of **marginal consistency**

$$\sum_{X_{\partial k \setminus i}} b(X_{\partial k}) = \sum_{X_{\partial k' \setminus i}} b(X_{\partial k'}) = b(X_i)$$

  and to the minimum of the **Bethe approximation** of the free energy (Yedidia, Freeman, & Weiss, 2001)

- We shouldn't be overly disappointed:
  – if BP was exact on loopy graphs we could efficiently solve NP hard problems...
  – loopy BP is a *very* interesting approximation to solving an NP hard problem
  – is hence also applied in context of combinatorial optimization (e.g., SAT problems)

- Ways to tackle the problems with BP convergence:
  – Damping (Heskes, 2004: *On the uniqueness of loopy belief propagation fixed points*)
  – CCCP (Yuille, 2002: *CCCP algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation*)
  – Tree-reweighted MP (Kolmogorov, 2006: *Convergent tree-reweighted message passing for energy minimization*)

8:26

Loopy BP application: Markov/Conditional Random Fields for computer vision

- In CRFs in Computer Vision (slides 03:19,20)

8:27

Junction Tree Algorithm

- Many models have loops

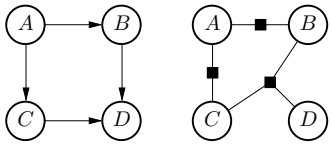
  Instead of applying loopy BP in the hope to get a good approximation, it is possible to convert every model into a tree by redefinition of RVs. The Junction Tree Algorithms converts a loopy model into a tree.

- Loops are resolved by defining larger variable groups (*separators*) on which messages are defined

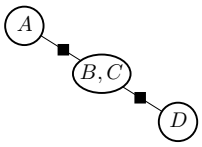
8:28

Junction Tree Example

- Example:



- Join variable B and C to a single **separator**



  This can be viewed as a variable substitution: rename the tuple (B, C) as a single random variable

- A single random variable may be part of multiple separators – but only along a *running intersection*

8:29

Junction Tree Algorithm

- Standard formulation: *Moralization & Triangulation*

  A **clique** is a fully connected subset of nodes in a graph.

  1) Generate the factor graph (classically called "moralization")

  2) Translate each factor to a clique: Generate the undirected graph

     where undirected edges connect all RVs of a factor

  3) Triangulate the undirected graph

  4) Translate each clique back to a factor; identify the separators between factors

- Formulation in terms of *variable elimination:*

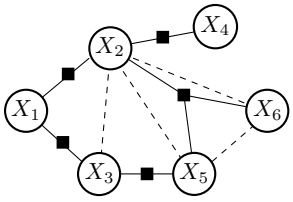
  1) Start with a factor graph

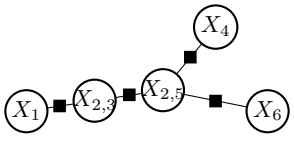
  2) Choose an order of variable elimination

  3) Keep track of the "remaining μ terms" (slide 14): which RVs would they depend on? $\rightarrow$ this identifies the separators

8:30

Junction Tree Algorithm Example





- If we eliminate in order 4, 6, 5, 1, 2, 3, we get remainder terms

$$(X_2),\ (X_2, X_5),\ (X_2, X_3),\ (X_2, X_3),\ (X_3)$$

  which translates to the Junction Tree on the right

8:31

Summary – what we covered

- Basic sampling methods for inference (rejection, importance, Gibbs)

- Message passing for
  - exact inference on (junction) trees
  - approximate inference ("loopy BP") on loopy graphs

8:32

Summary – what didn't cover

- **Maximum a-Posteriori (MAP) inference:**

  Sometimes we want to compute the most likely globa assignment

  $$X_{1:n}^{\text{MAP}} = \underset{x_{1:n}}{\text{argmax}}\, P(X_{1:n})$$

  of all random variables. ($\leftrightarrow$ Global optimization.)
  This is called MAP inference and can be solved by replacing all $\sum$ by $\max$ in the message passing equations – the algorithm is called **Max-Product Algorithm.**
  Max-Product is a generalization of Dynamic Programming methods like *Viterbi* or *Dijkstra*

- A very promising line of research is solving inference problems using mathematical programming. This unifies research in the areas of optimization, mathematical programming and probabilistic inference.

  Linear Programming relaxations of MAP inference and CCCP methods are great examples.

8:33

9 Learning with Graphical Models

Maximum likelihood learning, Expectation Maximization, Gaussian Mixture Models, Hidden Markov Models, Free Energy formulation, Discriminative Learning

Fully Bayes vs. ML learning

- **Fully Bayesian learning** (learning = inference)

  Every model "parameter" (e.g., the β of Ridge regression) is a RV. $\rightarrow$ We have a prior over this parameter; learning = computing the posterior

- **Maximum-Likelihood learning** (incl. **Expectation Maximization**)

  The conditional probabilities have parameters θ (e.g., the entries of a CPT). We find parameters θ to *maximize the likelihood of the data under the model*.

9:1

Likelihood, ML, fully Bayesian, MAP

We have a probabilistic model $P(X_{1:n};\theta)$ that depends on parameters θ. (Note: the ';' is used to indicate parameters.) We have data D.

- **The likelihood of the data under the model:**

$$P(D;\theta)$$

- **Maximum likelihood (ML) parameter estimate:**

$$\theta^{\text{ML}} := \text{argmax}_\theta\, P(D;\theta)$$

- If we treat θ as RV with prior $P(\theta)$ and $P(X_{1:n},\theta) = P(X_{1:n}|\theta)P(\theta)$:

  Fully Bayesian estimate:

$$P(\theta|D) \propto P(D|\theta)\; P(\theta)$$

  Bayesian prediction: $P(\text{prediction}|D) = \int_\theta P(\text{prediction}|\theta)\; P(\theta|D)$

- **Maximum a posteriori (MAP) parameter estimate:**

$$\theta^{\text{MAP}} = \text{argmax}_\theta\, P(\theta|D)$$

9:2

Maximum likelihood learning

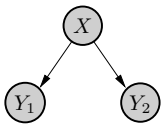
– fully observed case

– with latent variables ($\rightarrow$ Expectation Maximization)

9:3

Maximum likelihood parameters – fully observed

- An example: Given a model with three random varibles X, Y_1, Y_2:



  – We have unknown parameters

$$P(X=x) \equiv a_x,\; P(Y_1=y|X=x) \equiv b_{yx},\; P(Y_2=y|X=x) \equiv c_{yx}$$

  – Given a data set $\{(x_i, y_{1,i}, y_{2,i})\}_{i=1}^n$

  how can we learn the parameters $\theta = (a, b, c)$?

- Answer:

$$a_x = \frac{\sum_{i=1}^n [x_i=x]}{n}$$
$$b_{yx} = \frac{\sum_{i=1}^n [y_{1,i}=y \wedge x_i=x]}{na}$$
$$c_{yx} = \frac{\sum_{i=1}^n [y_{2,i}=y \wedge x_i=x]}{na}$$

9:4

Maximum likelihood parameters – fully observed

- In what sense is this the correct answer?

  $\rightarrow$ These parameters maximize *observed data log-likelihood*

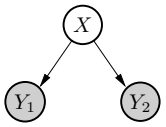
$$\log P(x_{1:n}, y_{1,1:N}, y_{2,1:n};\theta) = \sum_{i=1}^n \log P(x_i, y_{1,i}, y_{2,i}\,;\theta)$$

- When all RVs are fully observed, learning ML parameters is usually simple. This also holds when some random variables might be Gaussian (see mixture of Gaussians example).

9:5

Maximum likelihood parameters with missing data

- Given a model with three random varibles X, Y_1, Y_2:



  – We have unknown parameters

$$P(X=x) \equiv a_x,\; P(Y_1=y|X=x) \equiv b_{yx},\; P(Y_2=y|X=x) \equiv c_{yx}$$

  – Given a **partial** data set $\{(y_{1,i}, y_{2,i})\}_{i=1}^N$

  how can we learn the parameters $\theta = (a, b, c)$?

9:6

General idea

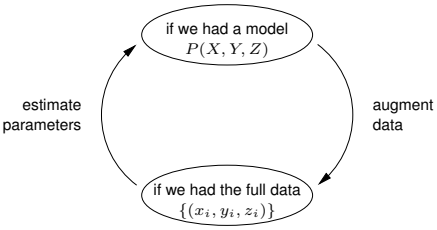
- We should somehow *fill in the missing data*

partial data $\{(y_{1:2,i})\}_{i=1}^n$ $\rightarrow$ augmented data $\{(\hat{x}_i, y_{1:2,i})\}_{i=1}^n$

- If we knew the model $P(X, Y_{1,2})$ already, we could use it to **infer** an $\hat{x}_i$ for each partial datum $(y_{1:2,i})$

  $\rightarrow$ then use this "augmented data" to train the model

- A chicken and egg situation:



9:7

Expectation Maximization (EM)

– Let X be a (set of) latent variables
– Let Y be a (set of) observed variables
– Let $P(X, Y; \theta)$ be a parameterized probabilistic model
– We have partial data $D = \{(y_i)\}_{i=1}^n$
– We start with some initial guess θ^{old} of parameters

Iterate:

- **Expectation-step:** Compute the posterior belief over the latent variables using θ^{old}

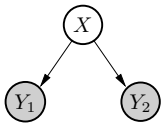
$$q_i(x) = P(x|y_i; \theta^{\text{old}})$$

- **Maximization-step:** Choose new parameters to maximize the expected data log-likelihood (expectation w.r.t. q)

$$\theta^{\text{new}} = \text{argmax}_\theta \underbrace{\sum_{i=1}^N \sum_x q_i(x) \log P(x, y_i; \theta)}_{=:Q(\theta, q)}$$

9:8

EM for our example



- **E-step:** Compute

$$q_i(x\,; \theta^{\text{old}}) = P(x|y_{1:2,i}; \theta^{\text{old}}) \propto a_x^{\text{old}}\, b_{y_{1,i}x}^{\text{old}}\, c_{y_{1,i}x}^{\text{old}}$$

- **M-step:** Update using the *expected* counts:

$$a_x^{\text{new}} = \frac{\sum_{i=1}^n q_i(x)}{n}$$
$$b_{yx}^{\text{new}} = \frac{\sum_{i=1}^n q_i(x)\, [y_{1,i}=y]}{na}$$
$$c_{yx}^{\text{new}} = \frac{\sum_{i=1}^n q_i(x)\, [y_{2,i}=y]}{na}$$

9:9

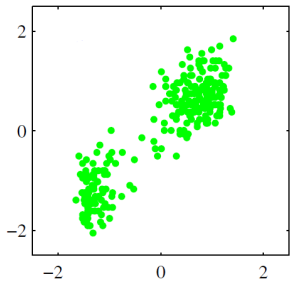
Examples for the application of EM

– Mixture of Gaussians
– Hidden Markov Models
– Outlier detection in regression

9:10

Gaussian Mixture Model – as example for EM

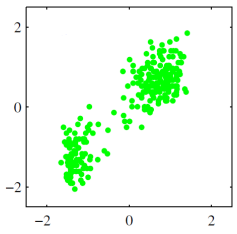
- What might be a *generative* model of this data?

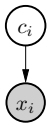


- There might be two "underlying" Gaussian distributions. A data point is drawn (here with about 50/50 chance) by one of the Gaussians.

9:11

Gaussian Mixture Model – as example for EM





- K different Gaussians with parmameters μ_k, Σ_k
- A latent random variable $c_i \in \{1, .., K\}$ for each data point with prior

$$P(c_i = k) = \pi_k$$

- The observed data point x_i

$$P(x_i \,|\, c_i = k; \mu_k, \Sigma_k) = \mathcal{N}(x_i \,|\, \mu_k, \Sigma_k)$$

9:12

Gaussian Mixture Model – as example for EM

Same situation as before:

- If we had full data $D = \{(c_i, x_i)\}_{i=1}^n$, we could estimate parameters

$$\pi_k = \frac{1}{n} \sum_{i=1}^n [c_i = k]$$
$$\mu_k = \frac{1}{n\pi_k} \sum_{i=1}^n [c_i = k]\, x_i$$

$$\Sigma_k = \frac{1}{n\pi_k} \sum_{i=1}^n [c_i = k]\; x_i x_i^\top - \mu_k \mu_k^\top$$

- If we knew the parameters $\mu_{1:K}, \Sigma_{1:K}$, we could infer the mixture labels

$$P(c_i = k \,|\, x_i; \mu_{1:K}, \Sigma_{1:K}) \propto \mathcal{N}(x_i \,|\, \mu_k, \Sigma_k)\; P(c_i = k)$$

9:13

Gaussian Mixture Model – as example for EM

- **E-step:** Compute

$$q(c_i = k) = P(c_i = k \,|\, x_i; , \mu_{1:K}, \Sigma_{1:K}) \propto \mathcal{N}(x_i \,|\, \mu_k, \Sigma_k)\; P(c_i = k)$$

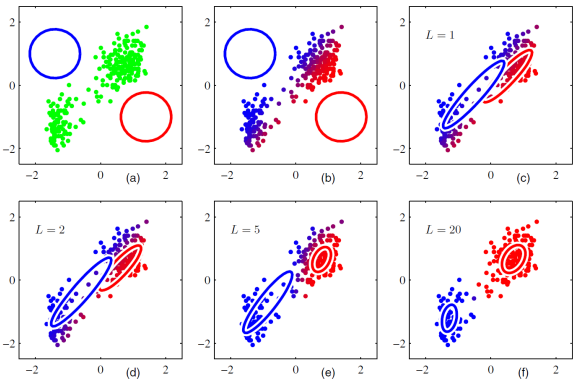
- **M-step:** Update parameters

$$\pi_k = \frac{1}{n} \sum_i q(c_i = k)$$
$$\mu_k = \frac{1}{n\pi_k} \sum_i q(c_i = k)\; x_i$$
$$\Sigma_k = \frac{1}{n\pi_k} \sum_i q(c_i = k)\; x_i x_i^\top - \mu_k \mu_k^\top$$

9:14

Gaussian Mixture Model – as example for EM

EM iterations for Gaussian Mixture model:



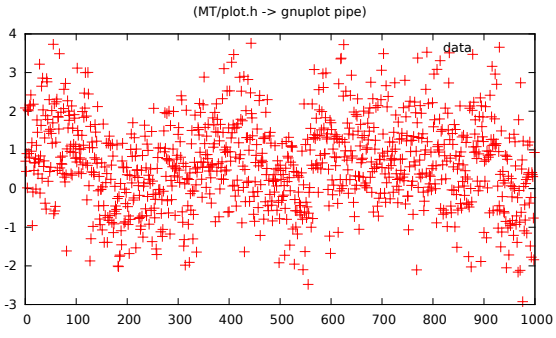
from Bishop

9:15

- Generally, the term **mixture** is used when there is a discrete latent variable (which indicates the *"mixture component"*).

9:16

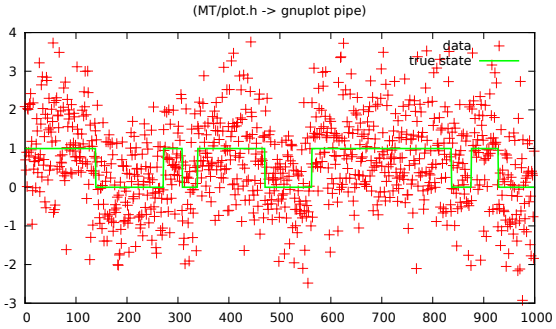
Hidden Markov Model – as example for EM



What probabilistic model could explain this data?

9:17

Hidden Markov Model – as example for EM



There could be a latent variable, with two states, indicating the high/low mean of the data.

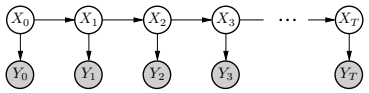
9:18

Hidden Markov Models

- We assume we have
  - observed (discrete or continuous) variables Y_t in each time slice
  - a discrete latent variable X_t in each time slice
  - some observation model $P(Y_t \,|\, X_t; \theta)$
  - some transition model $P(X_t \,|\, X_{t\text{-}1}; \theta)$

- A **Hidden Markov Model (HMM)** is defined as the joint distribution

$$P(X_{0:T}, Y_{0:T}) = P(X_0) \cdot \prod_{t=1}^T P(X_t|X_{t\text{-}1}) \cdot \prod_{t=0}^T P(Y_t|X_t)\ .$$



9:19

EM for Hidden Markov Models

- Given data $D = \{y_t\}_{t=0}^T$. How can we learn all parameters of a HMM?

- What are the parameters of an HMM?
  - The number K of discrete latent states x_t (the cardinality of dom(x_t))
  - The transition probability $P(x_t|x_{t\text{-}1})$ (a $K \times K$-CPT $P_{k'k}$)
  - The initial state distribution $P(x_0)$ (a K-table π_k)
  - Parameters of the output distribution $P(y_t \,|\, x_t = k)$

    e.g., the mean μ_k and covariance matrix Σ_k for each latent state k

- Learning K is very hard!
  – Try and evaluate many K
  – Have a distribution over K ($\rightarrow$ infinite HMMs)
  – We assume K fixed in the following

- Learning the other parameters is easier: **Expectation Maximization**

9:20

EM for Hidden Markov Models

The paramers of an HMM are $\theta = (P_{k'k}, \pi_k, \mu_k, \Sigma_k)$

- **E-step:** Compute the marginal posteriors (also pair-wise)

$$q(x_t) = P(x_t \,|\, y_{1:T}; \theta^{\text{old}})$$
$$q(x_t, x_{t+1}) = P(x_t, x_{t+1} \,|\, y_{1:T}; \theta^{\text{old}})$$

- **M-step:** Update the parameters

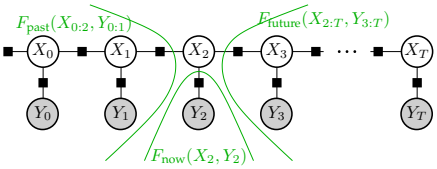
$$P_{k'k} \leftarrow \frac{\sum_{t=0}^{T-1} q(x_t=k, x_{t+1}=k')}{\sum_{t=0}^{T-1} q(x_t=k)}\,, \qquad \pi_k \leftarrow q(x_0=k)$$
$$\mu_k \leftarrow \sum_{t=0}^{T} w_{kt}\, y_t\,, \qquad \Sigma_k \leftarrow \sum_{t=0}^{T} w_{kt}\, y_t y_t^\top - \mu_k \mu_k^\top$$

with $w_{kt} = q(x_t=k)/(\sum_{t=0}^{T} q(x_t=k))$ (each row of w_{kt} is normalized)

compare with EM for Gaussian Mixtures!

9:21

E-step: Inference in an HMM – a tree!



- The marginal posterior $P(X_t \,|\, Y_{1:T})$ is the product of three messages

$$P(X_t \,|\, Y_{1:T}) \propto P(X_t, Y_{1:T}) = \underbrace{\mu_{\text{past}}(X_t)}_{\alpha}\ \underbrace{\mu_{\text{now}}(X_t)}_{\varrho}\ \underbrace{\mu_{\text{future}}(X_t)}_{\beta}$$

- For all $a < t$ and $b > t$
  – X_a conditionally independent from X_b given X_t
  – Y_a conditionally independent from Y_b given X_t

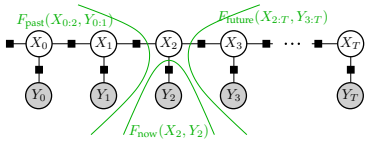
  "The future is independent of the past given the present"

  Markov property

(conditioning on Y_t does not yield any conditional independences)

9:22

Inference in HMMs



Applying the general message passing equations:

forward msg. $\quad \mu_{X_{t-1} \to X_t}(x_t) =: \alpha_t(x_t) = \sum_{x_{t-1}} P(x_t|x_{t-1})\ \alpha_{t-1}(x_{t-1})\ \varrho_{t-1}(x_{t-1})$

$\alpha_0(x_0) = P(x_0)$

backward msg. $\quad \mu_{X_{t+1} \to X_t}(x_t) =: \beta_t(x_t) = \sum_{x_{t+1}} P(x_{t+1}|x_t)\ \beta_{t+1}(x_{t+1})\ \varrho_{t+1}(x_{t+1})$

$\beta_T(x_0) = 1$

observation msg. $\quad \mu_{Y_t \to X_t}(x_t) =: \varrho_t(x_t) = P(y_t \,|\, x_t)$

posterior marginal $\quad q(x_t) \propto \alpha_t(x_t)\ \varrho_t(x_t)\ \beta_t(x_t)$

posterior marginal $\quad q(x_t, x_{t+1}) \propto \alpha_t(x_t)\ \varrho_t(x_t)\ P(x_{t+1}|x_t)\ \varrho_{t+1}(x_{t+1})\ \beta_{t+}$

9:23

Inference in HMMs – implementation notes

- The message passing equations can be implemented directly by looping over all variables, e.g., in case of the forward message:

1: $\texttt{for}_x$: $\alpha_0(x) = \pi(x)$
2: $\texttt{for}_{t=1:T}$: $\texttt{for}_x$: $\alpha_t(x) = 0$
3: $\texttt{for}_{t=1:T}$: $\texttt{for}_x$: $\texttt{for}_{x'}$: $\alpha_t(x)$ += $P(x|x')\alpha_{t-1}(x')\varrho_{t-1}(x')$

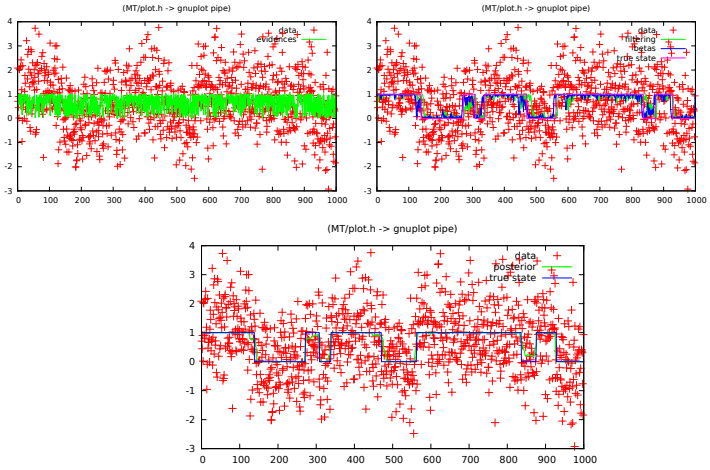
- Alternatively, the message passing equations can be implemented by reinterpreting them as matrix equations: Let $\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \boldsymbol{\varrho}_t$ be the vectors corresponding to the probability tables $\alpha_t(x_t), \beta_t(x_t), \varrho_t(x_t)$; and let $\boldsymbol{P}$ be the matrix with entries $P(x_t \,|\, x_{t-1})$. Then

forward msg.	$\boldsymbol{\alpha}_t = \boldsymbol{P}\,(\boldsymbol{\alpha}_{t-1} \cdot \boldsymbol{\varrho}_{t-1})$
backward msg.	$\boldsymbol{\beta}_t = \boldsymbol{P}^\top(\boldsymbol{\beta}_{t+1} \cdot \boldsymbol{\varrho}_{t+1})$
posterior marginal	$\boldsymbol{q}_t = \boldsymbol{\alpha}_t \cdot \boldsymbol{\varrho}_t \cdot \boldsymbol{\beta}_t$

where $\cdot$ is the *element-wise product* of vectors! Let $\boldsymbol{Q}_t$ be the matrix with entries $q(x_t, x_{t+1})$. The equation for $\boldsymbol{Q}_t$ cannot be written nicely in matrix notation. We have to fall back to explicit index notations and loop, e.g.:

1: $\texttt{for}_{t=1:T\text{-}1}$: $\texttt{for}_x$: $\texttt{for}_{x'}$: $Q_t(x, x') = \alpha_t(x)\varrho_t(x)P(x'|x)\varrho_{t+1}(x')\beta_{t+1}(x')$

9:24



9:25

Inference in HMMs: classical derivation

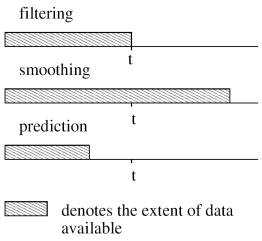
Given our knowledge of Belief propagation, inference in HMMs is simple. However, there is a more classical derivation:

$$\begin{aligned}
P(x_t \,|\, y_{0:T}) &= \frac{P(y_{0:T} \,|\, x_t)\; P(x_t)}{P(y_{0:T})} \\
&= \frac{P(y_{0:t} \,|\, x_t)\; P(y_{t+1:T} \,|\, x_t)\; P(x_t)}{P(y_{0:T})} \\
&= \frac{P(y_{0:t}, x_t)\; P(y_{t+1:T} \,|\, x_t)}{P(y_{0:T})} \\
&= \frac{\alpha_t(x_t)\; \beta_t(x_t)}{P(y_{0:T})} \\
\alpha_t(x_t) &:= P(y_{0:t}, x_t) = P(y_t|x_t)\; P(y_{0:t\text{-}1}, x_t) \\
&= P(y_t|x_t) \sum_{x_{t\text{-}1}} P(x_t \,|\, x_{t\text{-}1})\; \alpha_{t\text{-}1}(x_{t\text{-}1}) \\
\beta_t(x_t) &:= P(y_{t+1:T} \,|\, x_t) = \sum_{x+1} P(y_{t+1:T} \,|\, x_{t+1})\; P(x_{t+1} \,|\, x_t) \\
&= \sum_{x+1} \Big[\beta_{t+1}(x_{t+1})\; P(y_{t+1}|x_{t+1})\Big]\; P(x_{t+1} \,|\, x_t)
\end{aligned}$$

Note: α_t here is the same as $\alpha_t \cdot \varrho_t$ on all other slides!

9:26

Different inference problems in HMMs



- $P(x_t \,|\, y_{0:T})$ marginal posterior
- $P(x_t \,|\, y_{0:t})$ filtering
- $P(x_t \,|\, y_{0:a})$, $t > a$ prediction
- $P(x_t \,|\, y_{0:b})$, $t < b$ smoothing
- $P(y_{0:T})$ likelihood calculation

- Viterbi alignment: Find sequence $x^*_{0:T}$ that maximizes $P(x_{0:T} \,|\, y_{0:T})$ (This is done using max-product, instead of sum-product message passing.)

9:27

HMM remarks

- The computation of forward and backward messages along the Markov chain is also called **forward-backward algorithm**
- The EM algorithm to learn the HMM parameters is also called **Baum-Welch algorithm**
- If the latent variable x_t is **continuous** $x_t \in \mathbb{R}^d$ instead of discrete, then such a Markov model is also called *state space model*.
- If the continuous transitions are linear Gaussian

$$P(x_{t+1}|x_t) = \mathcal{N}(x_{t+1}|Ax_t + a, Q)$$

then the forward and backward messages α_t and β_t are also Gaussian.

→ forward filtering is also called **Kalman filtering**

→ smoothing is also called **Kalman smoothing**

- Sometimes, computing forward and backward messages (in discrete or continuous context) is also called **Bayesian filtering/smoothing**

9:28

HMM example: Learning Bach

- A machine "listens" (reads notes of) Bach pieces over and over again

  → It's supposed to learn how to write Bach pieces itself (or at least harmonize them).

- *Harmonizing Chorales in the Style of J S Bach* Moray Allan & Chris Williams (NIPS 2004)

- use an HMM

  – observed sequence $Y_{0:T}$ Soprano melody

  – latent sequence $X_{0:T}$ chord & and harmony:

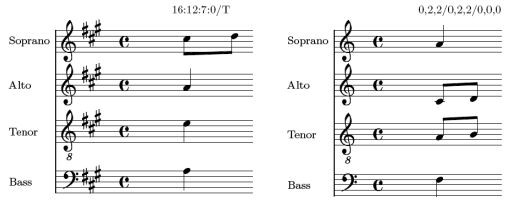


Figure 1: Hidden state representations (a) for harmonisation, (b) for ornamentation.

9:29

HMM example: Learning Bach

- results: http://www.anc.inf.ed.ac.uk/demos/hmmbach/

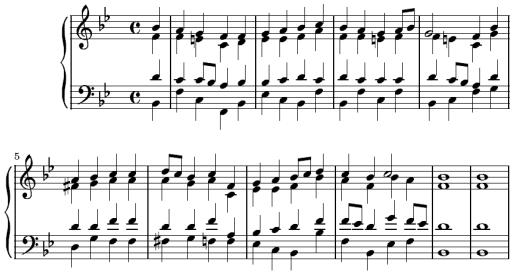


Figure 2: Most likely harmonisation under our model of chorale K4, BWV 48

- See also work by Gerhard Widmer http://www.cp.jku.at/people/widmer/

9:30

Free Energy formulation of EM

- We introduced EM rather heuristically: an iteration to resolve the chicken and egg problem.

  Are there convergence proofs?

  Are there generalizations?

  Are there more theoretical insights?

  → Free Energy formulation of Expectation Maximization

9:31

Free Energy formulation of EM

- Define a function (called "Free Energy")

$$F(q,\theta) = \log P(Y;\theta) - D\big(q(X)\,\big\|\,P(X|Y;\theta)\big)$$

where

$\log P(Y;\theta) = \log \sum_X P(X,Y;\theta)$ log-likelihood of *observed* data Y

$D\big(q(X)\,\big\|\,P(X|Y;\theta)\big) := \sum_X q(X) \log \frac{q(X)}{P(X\,|\,Y;\theta)}$ Kullback-Leibler div.

- The Kullback-Leibler divergence is sort-of a distance measure between two distributions: $D\big(q\,\big\|\,p\big)$ is always positive and zero only if $q = p$.

9:32

Free Energy formulation of EM

- We can write the free energy in two ways

$$\begin{aligned} F(q,\theta) &= \underbrace{\log P(Y;\theta)}_{\text{data log-like.}} - \underbrace{D\big(q(X)\,\big\|\,P(X|Y;\theta)\big)}_{\text{E-step: } q \leftarrow \text{argmin}} && (1)\\ &= \log P(Y;\theta) - \sum_X q(X) \log \frac{q(X)}{P(X\,|\,Y;\theta)} \\ &= \sum_X q(X) \log P(Y;\theta) + \sum_X q(X) \log P(X\,|\,Y;\theta) + H(q) \\ &= \underbrace{\sum_X q(X) \log P(X,Y;\theta)}_{\text{M-step: } \theta \leftarrow \text{argmax}} + H(q), && (2) \end{aligned}$$

- We actually want to maximize $P(Y;\theta)$ w.r.t. θ → but can't analytically
- Instead, we maximize the lower bound $F(q,\theta) \le \log P(Y;\theta)$
  - E-step: find q that maximizes $F(q,\theta)$ for fixed θ^{old} using (1)
    ($\to q(X)$ approximates $P(X|Y;\theta)$, makes lower bound tight)
  - M-step: find θ that maximizes $F(q,\theta)$ for fixed q using (2)
    ($\to \theta$ maximizes $Q(\theta,q) = \sum_X q(X) \log P(X,Y;\theta)$)
- **Convergence proof:** $F(q,\theta)$ increases in each step.

9:33

- The free energy formulation of EM
  - clarifies convergence of EM (to *local minima*)
  - clarifies that q need only to *approximate* the posterior
- *Why is it called free energy?*
  - Given a physical system in specific state (x,y), it has energy $E(x,y)$

    This implies a joint distribution $p(x,z) = \frac{1}{Z} e^{-E(x,y)}$ with partition function Z
  - If the DoFs y are "constrained" and x "unconstrained", then the *expected energy* is $\langle E\rangle = \sum_x q(x) E(x,y)$

    Note $\langle E\rangle = -Q(\theta,q)$ is the same as the neg. expected data log-likelihood
  - The unconstrained DoFs x have an entropy $H(q)$
  - Physicists call $A = \langle E\rangle - H$ (Helmholtz) free energy

    Which is the negative of our eq. (2): $F(q,\theta) = Q(\theta,q) + H(q)$

9:34

Discriminative (conditional) vs. generative models

(very briefly)

9:35

Maximum conditional likelihood

- We have two random variables X and Y and full data $D = \{(x_i,y_i)\}$

  Given a model $P(X,Y;\theta)$ of the joint distribution we can train it

$$\theta^{\text{ML}} = \operatorname*{argmax}_\theta \prod_i P(x_i,y_i;\theta)$$

- Assume the *actual goal* is to have a classifier $x \mapsto y$, or, the *conditional* distribution $P(Y|X)$.

  Q: Is it really necessary to first learn the full joint model $P(X,Y)$ when we only need $P(Y|X)$?

  Q: If $P(X)$ is very complicated but $P(Y|X)$ easy, then learning the joint $P(X,Y)$ is unnecessarily hard?

9:36

Maximum conditional Likelihood

Instead of likelihood maximization $\theta^{\text{ML}} = \operatorname{argmax}_\theta \prod_i P(x_i,y_i;\theta)$:

- **Maximum conditional likelihood:**

  Given a *conditional model* $P(Y|X;\theta)$, train

$$\theta^* = \operatorname*{argmax}_\theta \prod_i P(y_i|x_i;\theta)$$

- The little but essential difference: We don't care to learn $P(X)$!

9:37

Example: Conditional Random Field (03:23)

$$f(y,x) = \phi(y,x)^\top \beta \quad = \sum_{j=1}^k \phi_j(y_{\partial j},x)\beta_j$$

$$p(y|x) = e^{f(y,x) - Z(x,\beta)} \quad \propto \prod_{j=1}^k e^{\phi_j(y_{\partial j},x)\beta_j}$$

- "Random Field" $\leftrightarrow$ "Markov Random Field" $\leftrightarrow$ Factor Graph
  "Conditional" $\leftrightarrow$ we optimize for *conditional* likelihood.

- Logistic regression also maximizes the conditional log-likelihood
- Plain regression also maximizes the conditional log-likelihood! (with $\log P(y_i|x_i;\beta) \propto (y_i - \phi(x_i)^\top\beta)^2$)

9:38

10 Exercises

10.1 Exercise 1

10.1.1 Hastie, Tibshirani & Friedman

Read chapter 1 of Hastie et al.'s "Elements of Statistical Learning" (http://www-stat.stanford.edu/~tibs/ElemStatLearn/). Consider the DNA microarray data of Figure 1.3. Let's assume that samples 1-32 are taken from cancer cells whereas samples 33-64 from non-cancer cells. How could one analyze which genes are "involved with cancer"? No formal answers needed, but ceative ideas.

10.1.2 Matrix equations

a) Let X, A be arbitrary matrices, A invertible. Solve for X:

$$XA + A^\top = \mathbf{I}$$

b) Let X, A, B be arbitrary matrices, $(C - 2A^\top)$ invertible. Solve for X:

$$X^\top C = [2A(X + B)]^\top$$

c) Let $x \in \mathbb{R}^n, y \in \mathbb{R}^d, A \in \mathbb{R}^{d\times n}$. A obviously *not* invertible, but let $A^\top A$ be invertible. Solve for x:

$$(Ax - y)^\top A = \mathbf{0}_n^\top$$

d) As above, additionally $B \in \mathbb{R}^{n\times n}$, B positive-definite. Solve for x:

$$(Ax - y)^\top A + x^\top B = \mathbf{0}_n^\top$$

10.1.3 Vector derivatives

Let $x \in \mathbb{R}^n, y \in \mathbb{R}^d, A \in \mathbb{R}^{d\times n}$.

a) What is $\frac{\partial}{\partial x} x$? (Of what type/dimension is this thing?)

b) What is $\frac{\partial}{\partial x}[x^\top x]$?

c) Let B be symmetric (and pos.def.). What is the minimum of $(Ax - y)^\top(Ax - y) + x^\top Bx$ w.r.t. x?

10.1.4 Code

Future exercises will need you to code some Machine Learning methods. I'll support C++, but you are free to choose your programming language, which needs to support linear algebra and matrix manipulations.

For those using C++, download and test http://ipvs.informatik.uni-stuttgart.de/mlr/marc/source-code/libMLcourse.13.tgz (see README). In particular, have a look at test/array/main with many examples on how to use the array class. Report on problems with installation.

10.2 Exercise 2

10.2.1 Linear Ridge Regression

On the course webpage there are two simple data sets `dataLinReg1D.txt` and `dataLinReg2D.txt`. The data files can be plotted, e.g., using gnuplot with `plot 'dataLin` or `splot 'dataLinReg2D.txt'`. Each line contains a data entry (x, y) with $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$; the last entry in a line refers to y. Compute and report the optimal parameters β for a linear Ridge regression model (just linear features) for both data sets. Tips:

a) Write a routine that loads a data file and returns a matrix X containing all x_i as rows, and a vector y containing all y_i.

b) Write a routine that takes the raw X as input and returns a new $\boldsymbol{X}$ with a '1' pre-pended to each row. This routine simply computes the "linear features" including the constant 1. This routine can later be replaced by others to work with non-linear features.

c) Write a routine that returns the otpimal β from $\boldsymbol{X}$ and y.

d) Generate some test data points (along a grid) and collect them in a matrix Z. Apply routine b) to comupute features. Compute the predictions $\hat{y} = Z\beta$ (simple matrix multiplication) on the test data and plot it.

10.2.2 Non-linear features

Test regression with quadratic features on the data sets `dataQuadReg1D.txt` and `dataQuadReg2D.txt`. Compute and report the optimal parameters β for both data

sets. In principle, all you have to do is replace routine b) above (see slide 02:8).

10.2.3 Cross-validation

Implement cross-validation (slide 02:18) to evaluate the generalization performance of the linear and polynomial regression method for `dataLinReg2D.txt`. Report 1) the squared error when training on all data, 2) the mean squared error $\hat{\ell}$ from cross-validation, and 3) the standard deviation of $\hat{\ell}$.

Repeat this for different Ridge regularization parameters λ. (Ideally, generate a nice bar plot of the generalization error, including deviation, for various λ.)

10.3 Exercise 3

Comment to last exercise: The cross validation example wasn't well chosen. I added another data set `dataQuadReg2D_noisy.txt` for which the regularization—and cross validation—makes sense.

10.3.1 Log-likelihood gradient and Hessian

Consider a binary classification problem with data $D = \{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$. We define

$$f(x) = x^\top \beta \tag{3}$$

$$p(x) = \sigma(f(x)) \ , \quad \sigma(z) = 1/(1 + e^{-z}) \tag{4}$$

$$L(\beta) = -\sum_{i=1}^{n} \Big[y_i \log p(x_i) + (1 - y_i) \log[1 - p(x_i)] \Big] \tag{5}$$

where $\beta \in \mathbb{R}^d$ is a vector. (NOTE: the $p(x)$ we defined here is a short-hand for $p(y = 1|x)$ on slide 03:10.)

a) Compute the derivative $\frac{\partial}{\partial \beta} L(\beta)$. Tip: use the fact $\frac{\partial}{\partial z} \sigma(z) = \sigma(z)(1 - \sigma(z))$.

b) Compute the 2nd derivative $\frac{\partial^2}{\partial \beta^2} L(\beta)$.

10.3.2 Logistic Regression

On the course webpage there is a data set `data2Class.txt` for a binary classification problem. Each line contains a data entry (x, y) with $x \in \mathbb{R}^2$ and $y \in \{0, 1\}$.

a) Compute the optimal parameters β and the mean neg-log-likelihood ($\frac{1}{n} L(\beta)$) of logistic regression using linear features. Plot the probability $p(y = 1 \,|\, x)$ over a 2D grid of test points.

Useful gnuplot commands:

```
splot [-2:3][-2:3][-3:3.5] 'model' matrix \
   us ($1/20-2):($2/20-2):3 with lines notitle
plot [-2:3][-2:3] 'data2Class.txt' \
   us 1:2:3 with points pt 2 lc variable title 'train'
```

b) Compute and plot the same for quadratic features.

10.3.3 Handwritten Digit Classification

On the course webpage there is a data set in two files, `digit_pcs.txt` and `digit_label.txt`, the first containing the inputs x_i in each row, the second the label $y \in \{0, 1\}$ in each row. This data are handwritten digits, encoded using PCA components (explained later in the lecture), as illustrated in Figure 1.

Use the same code as above to learn a binary classifier on this data. What is the mean neg-log-likelihood you achieve with linear and with quadratic features? What the correct classification rate?

For further information on how this data was generated, see
`http://..../teaching/data/LogReg_digits_PCA_by_S`

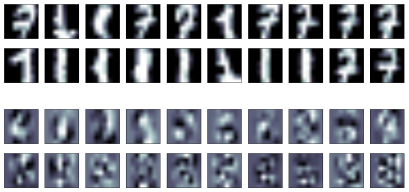


Figure 1: The top figure displays data examples of two classes of digits. The bottom figure displays the 20 first principle components of the whole (10 digit) data set – these were uses as inputs in the file `xdigit_pcs.txt`

10.4 Exercise 4

10.4.1 PCA and reconstruction on the Yale face database

On the webpage find and download the Yale face database http://ipvs.informatik.uni-stuttgart.de/mlr/marc/teaching/data/yalefaces.tgz. (Optinally use `yalefaces_cropBackground.tgz`) The file contains gif images of 166 faces.

a) Write a routine to load all images into a big data matrix $X \in \mathbb{R}^{165\times 77760}$, where each row contains a gray image.

In Octave, images can easily be read using `I=imread("subject01.gif");` and `imagesc(I);`. You can loop over files using `files=dir(".");` and `files(:).name`.

b) Compute the mean face $\mu = 1/n \sum_i x_i$ and center the whole data, $X \leftarrow X - \mathbf{1}_n \mu^\top$.

c) Compute the singular value decomposition $X = UDV^\top$ for the data matrix.[1] In Octave/Matlab, use the command `[U, S, V] = svd(X, "econ")`, where the `econ` ensures we don't run out of memory.

d) Map the whole data set to $\boldsymbol{Z} = \boldsymbol{X} V_p$, where $V_p \in \mathbb{R}^{77760\times p}$ contains only the first p columns of V. Assume $p = 60$. The Z represents each face as a p-dimensional vector, instead of a 77760-dimensional image.

e) Reconstruct all images by computing $\tilde{\boldsymbol{X}} = \mathbf{1}_n \mu^\top + \boldsymbol{Z} V_p^\top$. Display the reconstructed images (by reshaping each row of $\tilde{\boldsymbol{X}}$ to a 320×243-image) – do they look ok? Repeat for various PCA-dimensions $p = 1, 2, ...$

10.4.2 PLS for classification?

In the course we discussed Partial Least Squares as a method that, instead of just picking the p largest PCA components, it incrementally picks those components that are most correlated *with the output*. Meassuring correlation with the output is obvious in the regression case ($\hat{\boldsymbol{X}}^\top \boldsymbol{y}$ in the algorithm).

Do research to generalize PLS to the classification case. Report on your ideas, what you find in the web, and a pseudocode describing such an PLS-for-classification method.

10.5 Exercise 5

10.5.1 WEKA, decision trees and boosting

Weka http://www.cs.waikato.ac.nz/ml/weka/ is a Java toolbox for Machine Learning. Download and install it (an Ubuntu package exists).

Your task is to test various classifiers on the MNIST digit data set from http://yann.lecun.com/exdb/mnist/.

a) Download the datasets `digits-train.arg.gz` and `digits-test.arff.gz` from the course webpage. This is the result of a (painful) conversion of the original datasets to the arff format. `gunzip` the data. Start Weka with `weka -m 2g` (or more memory, otherwise you'll not be able to load the dataset. In explorer mode, load the dataset.

b) Try to find the best possible classifier using Weka. Focus on ensemble and decision tree methods. (You can choose from a large range of classifiers and change their parameters by right click on their name...) Compare with the results reported at http://yann.lecun.com/exdb/mnist/

c) Weka also allows you several preprocessing filters, including Principle Components. Test whether you can get better results using this preprocessing.

10.6 Exercise 6

10.6.1 SVMs

a) Draw a small dataset $\{(x_i, y_i)\}$, $x_i \in \mathbb{R}^2$ with two different classes $y_i \in \{0, 1\}$ such that a 1-nearest neighbor (1-NN) classifier has a lower leave-one-out cross validation error than a SVM classifier.

b) Draw a small dataset $\{(x_i, y_i)\}$, $x_i \in \mathbb{R}^2$ with two different classes $y_i \in \{0, 1\}$ such that 1-NN classifier

[1] This is alternative to what was discussed in the lecture: In the lecture we computed the SVD of $X^\top X = (UDV^\top)^\top (UDV^\top) = VD^2V^\top$, as U is orthonormal and $U^\top U = \mathbf{I}$. Decomposing the covariance matric $X^\top X$ is a bit more intuitive, decomposing X directly is more efficient and amounts to the same V.

has a higher leave-one-out cross validation error than a SVM classifier.

c) Proof that the constrained optimization problem (where $y_i \in \{-1, +1\}, C > 0$)

$$\min_{\beta,\xi} \|\beta\|^2 + C\sum_{i=1}^n \xi_i \quad \text{s.t.} \quad y_i(x_i^\top \beta) \geq 1 - \xi_i \ , \quad \xi_i \geq 0$$

can be rewritten as the unconstrained optimization problem

$$\min_{\beta} \|\beta\|^2 + C\sum_{i=1}^n \max\{0,\ 1 - y_i(x_i^\top \beta)\} \ .$$

(Note that the $\max$ term is the hinge loss.)

d) Explain why the optimal model (optimal β) will "not depend" on data points for which $\xi_i = 0$. Here "not depend" is meant in the variational sense: roughly, the derivative of β w.r.t. these data points is zero.

10.6.2 Neural Networks

(As preparation for the next lecture.)

A sober view on neural networks (NNs) is that they are an interesting class of parameterized functions $y = f(x, w)$ that map some input $x \in \mathbb{R}^n$ to some output $y \in \mathbb{R}$ depending on parameters w.

We've introduced the *logistic sigmoid function* as

$$\sigma(z) = \frac{e^z}{1+e^z} = \frac{1}{e^{-z}+1} \ , \quad \sigma'(z) = \sigma(z)(1-\sigma(z))$$

A 1-layer NN, with h_1 neurons in the hidden layer, is defined as

$$f(x, w) = w_1^\top \sigma(W_0 x) \ , \quad w_1 \in \mathbb{R}^{h_1}, W_0 \in \mathbb{R}^{h_1 \times d}$$

with parameters $w = (W_0, w_1)$, where $\sigma(z)$ is applied component-wise.

A 2-layer NN, with h_1, h_2 neurons in the hidden layers, is defined as

$$f(x, w) = w_2^\top \sigma(W_1 \sigma(W_0 x)) \ , \quad w_2 \in \mathbb{R}^{h_2}, W_1 \in \mathbb{R}^{h_2, h_1}, W_0 \in \mathbb{R}^{h_1 d}$$

with parameters $w = (W_0, W_1, w_2)$.

The weights of hidden neurons W_0, W_1 are usually trained using gradient descent. (The output weights w_1 or w_2 can be optimized analytically as for linear regression.)

a) Derive the gradient $\frac{\partial}{\partial W_0} f(x)$ for the 1-layer NN.

b) Derive the gradients $\frac{\partial}{\partial W_0} f(x)$ and $\frac{\partial}{\partial W_1} f(x)$ for the 2-layer NN.

10.7 Exercise 7

10.7.1 Independence

Write an algorithm (Octave or just pseudo code) that tests, for any given joint probability table $P(X, Y)$ over 2 random variables, whether the two random variables are independent. Test your algorithm on the following table:

	Y=0	Y=1	Y=2
X=0	.08	.12	.2
X=1	.12	.18	.3

(When writing pseudo code, be explicit what type/size all objects are.)

10.7.2 Sum of 3 dices

You have 3 dices (potentially fake dices where each one has a different probability table over the 6 values). You're given all three probability tables $P(D_1)$, $P(D_2)$, and $P(D_3)$. Write an algorithm that computes the conditional probability $P(S|D_1)$ of the sum of all three dices conditioned on the value of the first dice.

10.7.3 Conditionalized versions of product and Bayes rule

Prove from first principles the conditionalized version of the product rule (the same as the product rule, but every term is additionally conditioned on Z):

$$P(X, Y|Z) = P(Y|X, Z)\ P(X|Z) \ .$$

Also prove the conditionalized version of Bayes rule

$$P(X|Y, Z) = \frac{P(Y|X, Z)P(X|Z)}{P(Y|Z)} \ .$$

10.7.4 Product of Gaussians

Slide 06:25 defines a 1D and n-dimensional Gaussian distribution. See also Wikipedia, if necessary. Multiplying probability distributions is a fundamental operation, and multiplying two Gaussians is needed in many models. From the definition of a n-dimensional Gaussian, prove the general rule

$$\mathcal{N}(x \,|\, a, A)\, \mathcal{N}(x \,|\, b, B) \ \propto \ \mathcal{N}(x \,|\, (A^{-1}+B^{-1})^{-1}(A^{-1}a+B^{-1}b), (A^{-1}+B^{-1})^{-1}$$

where the proportionality $\propto$ allows you to drop all terms independent of x.

Note: The so-called canonical form of a Gaussian is defined as $\mathcal{N}[x \,|\, \bar{a}, \bar{A}] = \mathcal{N}(x \,|\, \bar{A}^{-1}\bar{a}, \bar{A}^{1})$; in this convention the product reads much nicher: $\mathcal{N}[x \,|\, \bar{a}, \bar{A}]\ \mathcal{N}[x \,|\, \bar{b}, \bar{B}] \propto \mathcal{N}[x \,|\, \bar{a}+\bar{b}, \bar{A}+\bar{B}]$. You can first prove this before proving the above, if you like.

10.8 Exercise 8

10.8.1 Gaussian Processes

Consider a Gaussian Process prior $P(f)$ over functions defined by the mean function $\mu(x) = 0$, the γ-exponential covariance function

$$k(x, x') = \exp\{-|(x - x')/l|^{\gamma}\}$$

and an observation noise $\sigma = 0.1$. We assume $x \in \mathbb{R}$ is 1-dimensional. First consider the standard squared exponential kernel with $\gamma = 2$ and $l = 0.2$.

a) Write a routine to draw 10 random functions from the prior $P(f)$. For this, discretize $x \in [-1, 1]$ to 100 points, compute the covariance matrix for these 100 points and sample.

b) Assume we have two data points $(-0.5, 0.3)$ and $(0.5, -0.1)$. Display the posterior $P(f|D)$. For this, compute the mean posterior function $\hat{f}(x)$ and the standard deviation function $\hat{\sigma}(x)$ (on the 100 grid points) exactly as on slide 07:9, using $\lambda = \sigma^2$. Then plot $\hat{f}$, $\hat{f} + \hat{\sigma}$ and $\hat{f} - \hat{s}$ to display the posterior mean and standard deviation.

c) Now display the posterior $P(y^*|x^*, D)$. This is only a tiny difference from the above (see slide 07:7). The mean is the same, but the variance of y^* includes additionally the obseration noise σ^2.

d) Sample 10 functions from the posterior $P(f|D)$ and display them.

e) Repeat a-d) for a kernel with $\gamma = 1$.

f) Optional: In case you have the code: compare the posterior $P(f|D)$ with standard Ridge regression using radial basis function features of width 0.2.

10.9 Exercise 9

10.9.1 Probabilistic modelling

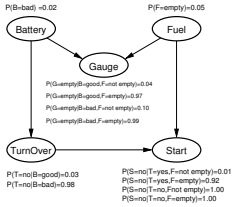
Formulate a probabilistic model for the price of used cars (e.g. on autoscout24 or so).

("Formulating a probabilistic model" includes: 1) which random variables exist, 2) how do they depend on each other: of which type are the conditional probability distributions and which parameters do they have.)

Let's assume your model is better than anybody else's model. How could you make money out of it?

10.9.2 Inference by hand

Consider the Bayesian network of binary random variables given below, which concerns the probability of a car starting.



Calculate $P(\text{Fuel=empty} \,|\, \text{Start=no})$, the probability of the fuel tank being empty conditioned on the observation that the car does not start. Do this calculation by hand. (First compute the joint probability $P(\text{Fuel}, \text{Start=no})$ by eliminating all latent (non-observed) variables except for Fuel.)

10.9.3 Inference by constructing the full joint

Consider the same example as above. This time write an algorithm that first computes the full joint probability table $P(S, T, G, F, B)$. From this compute $P(F|S)$.

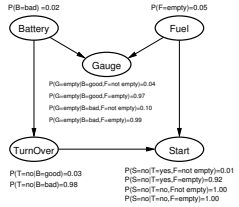
10.9.4 Constructing a Bayes net

On slide 08:12 general rules for determining $Indep(X, Y|Z)$ are given. Use these to construct the Bayesian network for the car example when random variables are added in the order S, T, G, F, B (see slides 08:8,9).

10.10 Exercise 10

10.10.1 Sampling

Consider again the Bayesian network of binary random variables given below, which concerns the probability of a car starting.



a) Condition on Start=no. Implement rejection sampling to collect a sample set $\mathcal{S} \sim P(B, F, G, T \,|\, S = \text{no})$ with K samples (e.g., $K = 1000$). Compute $P(F \,|\, S = \text{no})$.

b) Do the same using importance weighting instead of rejection sampling. Compare the results for varying K.

10.10.2 Sampling from a Gaussian

Consider the following simple model: There is a 1-dimensional Gaussian random variable x with $P(x) = \mathcal{N}(x|0, 1)$. There is a binary random variable Y with

$$P(Y=1 \,|\, x) = \begin{cases} .9 & x > 0 \\ .1 & \text{ortherwise} \end{cases}$$

.

Use rejection sampling to compute a sample set representing the posterior $P(x \,|\, Y=1)$. From this, compute an estimate of the posterior mean $\int_x x\; P(x \,|\, Y=1)$.

10.10.3 Message passing

Consider three random variables A, B and C with joint distribution $P(A, B, C) = P(A)\; P(B|A)\; P(C|B)$. Let each RV be binary. We assume the CPTs are

$$P(A) = \begin{bmatrix} 1/3 & 2/3 \end{bmatrix}$$
$$P(B|A) = \begin{bmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{bmatrix}$$
$$P(C|B) = \begin{bmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{bmatrix}$$

a) Draw the Bayes Net for the joint $P(A, B, C)$. Draw also the factor graph that corresponds to $P(A, B, C) = f_1(A)\; f_2(A, B)\; f_3(B, C)$.

b) Compute (by hand on paper) all messages in this factor graph. These are (forward) $\mu_{1\to A}(A)$, $\mu_{2\to B}(B)$, $\mu_{3\to C}(C)$ and (backward) $\mu_{3\to B}(B)$, $\mu_{2\to A}(A)$.

c) Compute the posterior marginals (also called "beliefs") $P(A)$, $P(B)$, and $P(C)$ for each variable.

d) Assume we have an additional factor $f_4(A, C) = \begin{bmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{bmatrix}$ that couples A and C. To what messages would loopy belief propagation converge to when we would iterate infinitely? Would it actually converge? Would it converge modulo a scaling of the messages? All these questions can be addressed by investigating the fixed point equations of loopy BP – what is the fixed point equation for, say, $\mu_{4\to A}$? (No numerical answers necessary for these questions.)

10.11 Exercise 11

10.11.1 Max. likelihood estimator for a multinomial distribution

Let X be a discrete variable with domain $\{1, .., K\}$. We parameterize the discrete distribution as

$$P(X=k; \pi) = \pi_k \tag{6}$$

with parameters $\pi = (\pi_1, .., \pi_K)$ that are constrained to fulfill $\sum_k \pi_k = 1$. Assume we have some data $D = \{x_i\}_{i=1}^n$

a) Write down the likelihood $\mathcal{L}(\pi)$ of the data under the model.

b) Prove that the maximum likelihood estimator for π is, just as intuition tells us,

$$\pi_k^{\text{ML}} = \frac{1}{n} \sum_{i=1}^{n} [x=k] \;. \tag{7}$$

Tip: Although this seems trivial, it is not. The pitfall is that the parameter π is constrained with $\sum_k \pi_k = 1$. You need to solve this using Lagrange multipliers—see Wikipedia or Bishop section 2.2.

10.11.2 Gaussians and Singular Value Decomposition

On the course homepage there is a data set `gauss.txt` containing $n = 1000$ 2-dimensional points. Load it in a data matrix $\boldsymbol{X} \in \mathbb{R}^{n\times 2}$.

a) Compute the mean (e.g., `mu = 1.0/n*sum(X,0)`)

b) Center the data ($\tilde{\boldsymbol{X}} = \boldsymbol{X} - \mathbf{1}_n \mu^\top$)

c) Compute the covariance matrix $C = \frac{1}{n}\tilde{\boldsymbol{X}}^\top \tilde{\boldsymbol{X}}$. Also compute $\frac{1}{n}\boldsymbol{X}^\top \boldsymbol{X} - \mu\mu^\top$ (using the uncentered data) and compare.

d) Compute the Singular Value Decomposition $C = UDV^\top$. Ouput the eigenvalues (diagonal of D) and eigenvectors (columns of V). Plot the data and the two line segments between μ and $\mu + \sqrt{\lambda_j} v_j$, $j = 1, 2$.

10.11.3 Mixture of Gaussians

Download the data set `mixture.txt` from the course webpage, containing $n = 300$ 2-dimensional points. Load it in a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times 2}$.

a) Implement the EM-algorithm for a Gaussian Mixture on this data set. Choose $K = 3$ and the prior $P(c_i = k) = 1/K$. Initialize by choosing the three means μ_k to be different randomly selected data points x_i (i random in $\{1, .., n\}$) and the covariances $\Sigma_k = \mathbf{I}$ (a more robust choice would be the covariance of the whole data). Iterate EM starting with the first E-step based on these initializations. Repeat with random restarts—how often does it converge to the optimum?

Tip: Store $q(c_i = k)$ as a $K \times n$-matrix with entries q_{ki}; equally $w_{ki} = q_{ki}/\pi_k$. Store μ_k's as $K \times d$-matrix and Σ_k's as $K \times d \times d$-array. Then the M-step update for μ_k is just a matrix multiplication. The update for each Σ_k can be written as $\boldsymbol{X}^\top \mathrm{diag}(w_{k,1:d}) \boldsymbol{X} - \mu_k \mu_k^\top$.

b) Do exactly the same, but this time initialize the posterior $q(c_i = k)$ randomly (i.e., assign each point to a random cluster, $q(c_i) = [c_i = \text{rand}(1 : K)]$); then start EM with the first M-step. Is this better or worse than the previous way of initialization?

11 Topic list

This list summarizes the lecture's content and is intended as a guide for preparation for the exam. (Going through all exercises is equally important!) References to the lectures slides are given in the format (lecture:slide).

11.1 Regression & Classification

- Linear regression with non-linear features
  - What is the function model $f(x)$? (2:6)
  - What are possible features $\phi(x)$? (2:7-10)
  - Why do we need regularization? (2:11)
  - Cost is composed of a (squared) loss plus regularization (2:15)
  - How does the regularization reduce estimator variance (2:12,13,16)
  - Be able to derive optimal parameters for squared loss plus ridge regularization (2:15,4)
  - What is Lasso and Ridge regression? Be able to explain their effect (2:19-21)
- Cross validation
  - How does cross validation estimate the generalization error? Be able to explain k-fold cross validation (2:17)
  - Be able to draw the "idealized" train & test error curves depending on λ (2:18)
- Kernel trick
  - Understand that the kernel trick is to rewrite solutions in terms of $k(x', x) := \phi(x')^\top \phi(x)$ (2:23,24)
  - What is the predictor for Kernel Ridge Regression? (2:24)
  - What is the kernel equivalent to polynomial features? (2:27)
- Classification
  - Concept of a *discriminative function* $f(y, x)$ to represent a classification model $x \mapsto y$ (3:2-4)
  - How are discriminative functions translated to class probabilities via the logistic function/log-ratios? (3:7,13)
  - Log-likelihood as a proper "loss function" to train discriminative functions (3:7); including a regularization (3:9,13).
  - No closed form of the optimal solution $\rightarrow$ Newton method to find optimal β (3:10,14)
  - Discriminative functions that are linear in arbitrary features (3:8,11,12)
- Conditional Random Fields
  - What are examples for complex structured y's? (3:16-18)
  - How is the discriminative function $f(y, x)$ parameterized in CRFs? (3:21)
  - Can you see that logistic regression ($y \in \{0, 1\}$) is a special case? (3:21)

11.2 ML ideas

- PCA & PLS
  - What is centering and whitening? (4:2)
  - How does PCA select dimensions given a dataset $D = \{x_i\}$? (4:5)
  - What are examples for PCA dimensions in image datasets? (4:7-9)
  - PCA can also be kernelized (4:10-12)
  - What is different between PCA and PLS? (4:14,15)
- Local & lazy learning
  - What is a *smoothing* kernel? And what are common smoothing kernels? (4:18,19)
  - What are kd-trees good for? (4:21)
  - How are kd-trees constructed? (4:22,23)
- Combining weak or randomized learners
  - What are the differences between model averaging, bootstrapping, and boosting? (4:25)
  - What is the simplest way to bootstrap? (4:26)
  - How does AdaBoost assign differently weighted data to different learners (qualitatively)? (4:32-34)
  - Given a set of (learned) models $f_m(x)$, how can you combine them optimally? (4:35)
  - How can Gradient Boosting define the full model as a combination of local gradients? (4:36)
  - How are decision trees defined? (4:40) How are the regions and local constants found? (4:41,42)
  - What is a random forest? (4:45)

- SVMs (Vien's slides)

  – What is the different between logistic regression and a SVM? (4:55, 3:9)

  – How is the hinge loss related to maximizing margins? (Exercise 6.1, 4:53-55)

- Deep learning & representations

  – What are "Neural Networks"? (4:64)

  – What is Jason Weston's approach to make learning in deep Neural Networks efficient? (4:63,65-66)

11.3 Bayesian Modelling

- Probability basics

  – Notion of a random variable & probability distribution (5:7,8)

  – Definition of a joint, conditional and marginal (5:9)

  – Derivation of Bayes' Theorem from the basic definitions (5:10)

  – Be able to apply Bayes' Theorem to basic examples (5:12-16)

  – Definition of a Gaussian distribution (5:18); also: what is the maximum-likelihood estimate of the parameters of a Gaussian distribution from data (9:13)

- Bayesian Regression & Classification

  – Generic understanding of learning as Bayesian inference (6:1)

  – Bayesian Ridge Regression: How to formulate a Bayesian model that eventually turns out equivalent to ridge regression (6:3-5)

  – Relation between prior $P(\beta)$ and regularization, likelihood $P(y|x,\beta)$ and squared loss (6:2,7)

  – Fully Bayesian prediction (6:6); Computing also the uncertainty ("predictive error bars") of a prediction

  – Kernel Bayesian Ridge Regression = Gaussian Process (6:7)

  – What is the function space view of Gaussian Processes? (6:10)

  – How can you sample random functions from a GP? (6:11) And what are typical covariance functions? (6:12)

  – Bayesian Logistic Regression: How to formulate a Bayesian model that turns out equivalent to ridge logistic regression? (6:16-17)

  – Kernel Bayesian Logistic Regression = Gaussian Process Classification (6:18,19)

11.4 Graphical Models

- Bayesian Networks

  – Definition (7:3,5)

  – Graphical Notation for a Bayes net; each factor is a conditional probability table (CPT). Be able to count number of parameters in a Bayes net (7:6)

  – Constructing a Bayes Net by adding RVs in a specified order (7:7)

  – Deciding (conditional) independence given a Bayes Net (7:9-12)

  – Be able to compute posterior marginals by hand in simple Bayes Nets (7:14,16)

- Factor graphs

  – Definition of a Factor Graph (8:16)

  – Be able to convert any Bayes Net into a factor graph (8:17)

11.5 Inference in Graphical models

- Sampling methods

  – Generally understand that sampling methods generate (weighted) sample sets $\mathcal{S}$ from which posterior marginals are estimated (8:6,9,11)

  – Pseudo code of Rejection Sampling (8:5)

  – Pseudo code of Importance Sampling (8:8)

  – Gibbs sampling (8:10)

- Variable Elimination

  – Understand the significance of the "remaining terms" when eliminating variables in a given order (8:14)

  – Variable Elimination Algorithm (8:18)

- Message Passing

  – Variable Elimination on a tree $\rightarrow$ posterior marginals are a product of *subtree messages* (8:20) This is interpreted as fusion of independent information (8:21) Messages $\mu_{\cdot\rightarrow X}(X)$ are remaining potentials over X when eliminating subtrees

– General Message passing equations for a factor graph (8:22)

– Be able to write down the BP equations and compute by hand the messages for tiny models

– Intuitively understand the problem with loopy graphs (fusing dependent information, propagating them around loops) (8:25,26)

- Junction Tree Algorithm

  – Joining variables to make them *separators* (8:29)

  – Using Variable Elimination to decide how to joint variables to generate a Junction Tree (8:30,31)

11.6 Learning in Graphical models

– Different types of learning (9:1)

- Maximum-likelihood learning

  – Directly assigning CPTs to maximum likelihood estimates (normalized statistics in the discrete case) if all variables are observed (9:4,5)

  – Definition of Expectation Maximization (9:8)

  – Be able to derive explicit E- and M-steps for simple latent variable models

  – Gaussian Mixture model, Hidden Markov Models as example applications of EM

  – Be able to define the joint probability distribution for a Gaussian Mixture model (9:12) and a HMM (9:19)

  – Inference in HMMs (9:23), forward and backward equations for computing messages and posterior

  – Free Energy formulation of Expectation Maximization (9:33)

- Discriminative learning

  – Understand the difference between maximum likelihood and maximum conditional likelihood (9:36,37)