Machine Learning - Lecture 16

Repetition

10.07.2012

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de
leibe@umic.rwth-aachen.de

Announcements
• Today, I’ll summarize the most important points from the lecture.
  - It is an opportunity for you to ask questions...
  - ...or get additional explanations about certain topics.
  - So, please do ask.
• Today’s slides are intended as an index for the lecture.
  - But they are not complete, won’t be sufficient as only tool.
  - Also look at the exercises - they often explain algorithms in detail.
• Oral exam procedure
  - Oral exam, form depends on B.Sc./M.Sc./Diplom specifics
  - Procedure: 4 questions, will have to answer 3 of them
  - Special rule for Diplom V4 exam

Course Outline
• Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM
• Discriminative Approaches
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees
• Generative Models
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference

Recap: Bayes Decision Theory
• Optimal decision rule
  - Decide for \( C_1 \) if
    \[
    p(C_1|x) > p(C_2|x)
    \]
  - This is equivalent to
    \[
    p(x|C_1)p(C_1) > p(x|C_2)p(C_2)
    \]
  - Which is again equivalent to (Likelihood-Ratio test)
    \[
    \frac{p(x|C_1)}{p(x|C_2)} > \frac{p(C_2)}{p(C_1)}
    \]
  - Decision threshold \( \theta \)

Recap: Bayes Decision Theory
• Decision regions: \( R_1, R_2, R_3 \)
In general, we can formalize this by introducing a loss matrix $L_{kj}$

$$L_{kj} = \text{loss for decision } C_j \text{ if truth is } C_k.$$ 

- Example: cancer diagnosis

$$L_{\text{cancer diagnosis}} = \begin{pmatrix} \text{cancer} & \text{normal} \\ 0 & 1000 \\ 1 & 0 \end{pmatrix}$$

Discriminative Approaches

- Bayesian Networks
- Covariance
- Ensemble Methods & Boosting

Example: cancer diagnosis

Exact Inference

But: loss function depends on the true class, which is unknown.

Solution: Minimize the expected loss

$$E[L] = \sum_k \sum_j \int_{R_j} L_{kj} p(x, C_k) \, dx$$

This can be done by choosing the regions $R_j$ such that

$$E[L] = \sum_k L_{kj} p(C_k|x)$$

which is easy to do once we know the posterior class probabilities $p(C_k|x)$.

- Linear Discriminant Functions
- Statistical Learning Theory & SVMs
- Ensemble Methods & Boosting
- Decision Trees & Randomized Trees

Generative Models

- Bayesian Networks
- Markov Random Fields
- Exact Inference

Recap: Gaussian (or Normal) Distribution

- One-dimensional case
  - Mean $\mu$
  - Variance $\sigma^2$

$$N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x - \mu)^2}{2\sigma^2}\right\}$$

- Multi-dimensional case
  - Mean $\mu$
  - Covariance $\Sigma$

$$N(x|\mu, \Sigma) = \frac{1}{(2\pi)^d |\Sigma|^{1/2}} \exp\left\{-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right\}$$

Recap: Maximum Likelihood Approach

- Computation of the likelihood
  - Single data point: $p(x_n|\theta)$
  - Assumption: all data points $X = \{x_1, \ldots, x_n\}$ are independent

$$L(\theta) = p(X|\theta) = \prod_{n=1}^{N} p(x_n|\theta)$$

Log-likelihood

$$E(\theta) = -\ln L(\theta) = -\sum_{n=1}^{N} \ln p(x_n|\theta)$$

- Estimation of the parameters $\theta$ (Learning)
  - Maximize the likelihood (= minimize the negative log-likelihood)
    - Take the derivative and set it to zero.

$$\frac{\partial}{\partial \theta} E(\theta) = -\sum_{n=1}^{N} \frac{\partial}{\partial \theta} p(x_n|\theta) = 0$$
Recap: Bayesian Learning Approach

Bayesian view:
- Consider the parameter vector $\theta$ as a random variable.
- When estimating the parameters, what we compute is
  \[ p(x|X) = \int p(x, \theta|X) \, d\theta \]
  This is entirely determined by the parameter $\theta$ (i.e. by the parametric form of the pdf).

Assumption: given $\theta$, this doesn’t depend on $X$ anymore

Discussion
- The more uncertain we are about $\theta$, the more we average over all possible parameter values.

Recap: Histograms

Basic idea:
- Partition the data space into distinct bins with widths $\Delta_i$ and count the number of observations, $n_i$, in each bin.
- Often, the same width is used for all bins, $\Delta_i = \Delta$.
- This can be done, in principle, for any dimensionality $D$...

...but the required number of bins grows exponentially with $D$.

Recap: Kernel Density Estimation

Approximation formula:
- $p(x) \approx \frac{K}{N \Delta}$

Kernel methods
- Place a kernel window $k$ at location $x$ and count how many data points fall inside it.

K-Nearest Neighbor
- Increase the volume $V$ until the $K$ next data points are found.

Recap: Mixture of Gaussians (MoG)

"Generative model"
- $p(j) = \pi_j$

"Weight" of mixture component

Mixture component

Mixture density

Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM

- Discriminative Approaches
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees

- Generative Models
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference
Recap: MoG - Iterative Strategy

- Assuming we knew the values of the hidden variable...

\[ f(x) \]

ML for Gaussian #1

ML for Gaussian #2

assumed known

\[ h(j = 1|x_n) = \frac{1}{2} \quad \sum_{n=1}^{N} h(j = 1|x_n) x_n \]

\[ h(j = 2|x_n) = \frac{1}{2} \quad \sum_{n=1}^{N} h(j = 2|x_n) x_n \]

\[ \mu_1 = \frac{\sum_{n=1}^{N} h(j = 1|x_n) x_n}{\sum_{n=1}^{N} h(j = 1|x_n)} \quad \mu_2 = \frac{\sum_{n=1}^{N} h(j = 2|x_n) x_n}{\sum_{n=1}^{N} h(j = 2|x_n)} \]

Recap: MoG - Iterative Strategy

- Assuming we knew the mixture components...

\[ f(x) \]

\[ p(j = 1|x_n) \]

\[ p(j = 2|x_n) \]

Bayes decision rule: Decide \( j \) if

\[ p(j = 1|x_n) > p(j = 2|x_n) \]

Recap: EM Algorithm

- Expectation-Maximization (EM) Algorithm

\[ \gamma_j(x_n) \leftarrow \frac{p(j = 1|x_n) p(j = 1)}{\sum_{k=1}^{K} p(j = k|x_n)} \quad \forall j = 1, \ldots, K \]

\[ N_j \leftarrow \sum_{n=1}^{N} \gamma_j(x_n) = \text{soft number of samples labeled } j \]

\[ \hat{\mu}_j^{\text{new}} \leftarrow \frac{\sum_{n=1}^{N} \gamma_j(x_n) x_n}{N_j} \]

\[ \hat{\Sigma}_j^{\text{new}} \leftarrow \frac{1}{N_j} \sum_{n=1}^{N} \gamma_j(x_n) (x_n - \hat{\mu}_j^{\text{new}})(x_n - \hat{\mu}_j^{\text{new}})^T \]

Recap: Linear Discriminant Functions

- Basic idea

\[ y(x) = w^T x + w_0 \]

- Linear discriminant functions

\[ y(x) > 0 \quad y(x) = 0 \quad y(x) < 0 \]

weight vector “bias”

\( (= \text{threshold}) \)

- Define a hyperplane in \( \mathbb{R}^D \).

- If a data set can be perfectly classified by a linear discriminant, then we call it linearly separable.

Course Outline

- Fundamentals

- Bayes Decision Theory

- Probability Density Estimation

- Mixture Models and EM

- Discriminative Approaches

- Linear Discriminant Functions

- Statistical Learning Theory & SVMs

- Ensemble Methods & Boosting

- Decision Trees & Randomized Trees

- Generative Models

- Bayesian Networks

- Markov Random Fields

- Exact Inference
Recap: Least-Squares Classification

- Simplest approach
  - Directly try to minimize the sum-of-squares error
  \[
  E(w) = \sum_{n=1}^{N} (y(x_n; w) - t_n)^2
  \]
  \[
  E_D(\hat{W}) = \frac{1}{2} \text{Tr} \left\{ (\hat{X}\hat{W} - \mathbf{T})^T(\hat{X}\hat{W} - \mathbf{T}) \right\}
  \]
  - Setting the derivative to zero yields
  \[
  \hat{W} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T\mathbf{T}
  \]
  - We then obtain the discriminant function as
  \[
  y(x) = \hat{W}^T\hat{x} = \mathbf{T}^T(\hat{X}^T\hat{X})^{-1}\hat{X}^T\mathbf{T}
  \]
  \[
  \Rightarrow \text{Exact, closed-form solution for the discriminant function parameters.}
  \]

Recap: Problems with Least Squares

- Least-squares is very sensitive to outliers!
  - The error function penalizes predictions that are “too correct”.

Recap: Generalized Linear Models

- Generalized linear model
  \[
  y(x) = g(w^T x + w_0)
  \]
  - \( g(\cdot) \) is called an activation function and may be nonlinear.
  - The decision surfaces correspond to
    \[
    y(x) = \text{const.} \iff w^T x + w_0 = \text{const.}
    \]
  - If \( g \) is monotous (which is typically the case), the resulting decision boundaries are still linear functions of \( x \).

- Advantages of the non-linearity
  - Can be used to bound the influence of outliers and “too correct” data points.
  - When using a sigmoid for \( g(\cdot) \), we can interpret the \( y(x) \) as posterior probabilities.

Recap: Extension to Nonlinear Basis Fcts.

- Generalization
  - Transform vector \( x \) with \( M \) nonlinear basis functions \( \phi_j(x) \):
    \[
    y_k(x) = \sum_{j=1}^{M} w_{kj}\phi_j(x) + w_{k0}
    \]

- Advantages
  - Transformation allows non-linear decision boundaries.
  - By choosing the right \( \phi_j \), every continuous function can (in principle) be approximated with arbitrary accuracy.

- Disadvantages
  - The error function can in general no longer be minimized in closed form.
  \[
  \Rightarrow \text{Minimization with Gradient Descent}
  \]

Recap: Classification as Dim. Reduction

- Classification as dimensionality reduction
  - Interpret linear classification as a projection onto a lower-dim. space.
  \[
  y = w^T x
  \]
  \[
  \Rightarrow \text{Learning problem: Try to find the projection vector } w \text{ that maximizes class separation.}
  \]
Recap: Fisher's Linear Discriminant Analysis

- Maximize distance between classes
- Minimize distance within a class
- Criterion: $J(w) = w^{T}S_{B}w / w^{T}S_{W}w$
- $S_{B}$ ... between-class scatter matrix
- $S_{W}$ ... within-class scatter matrix
- The optimal solution for $w$ can be obtained as:
  \[ w \propto S_{W}^{-1}(m_{2} - m_{1}) \]
- Classification function:
  \[ y(x) = w^{T}x + w_{0} \]
  where $w_{0} = -w^{T}m$

Recap: Probabilistic Discriminative Models

- Consider models of the form
  \[ p(C_{1} | \phi) = \sigma(w^{T} \phi) \]
  \[ p(C_{2} | \phi) = 1 - p(C_{1} | \phi) \]
- This model is called logistic regression.
- Properties
  - Probabilistic interpretation
  - But discriminative method: only focus on decision hyperplane
  - Advantageous for high-dimensional spaces, requires less parameters than explicitly modeling $p(\phi | C_{i})$ and $p(C_{i})$.

Recap: Logistic Regression

- Let's consider a data set $\{ \phi_{n}, t_{n} \}$ with $n = 1, \ldots, N$ 
  where $\phi_{n} = \phi(x_{n})$ and $t_{n} \in \{0, 1\}$, $t = (t_{1}, \ldots, t_{N})^{T}$.
- With $y_{n} = p(C_{1} | \phi_{n})$, we can write the likelihood as
  \[ p(t | w) = \prod_{n=1}^{N} y_{n}^{t_{n}} (1 - y_{n})^{1-t_{n}} \]
- Define the error function as the negative log-likelihood
  \[ E(w) = -\ln p(t | w) = -\sum_{n=1}^{N} (t_{n} \ln y_{n} + (1 - t_{n}) \ln(1 - y_{n})) \]
  This is the so-called cross-entropy error function.

Recap: Iteratively Reweighted Least Squares

- Update equations
  \[ w^{(r+1)} = w^{(r)} - (\Phi^{T}R\Phi)^{-1}\Phi^{T}(y - t) \]
  \[ = (\Phi^{T}R\Phi)^{-1}\Phi^{T}R\Phi w^{(r)} - \Phi^{T}(y - t) \]
  \[ = (\Phi^{T}R\Phi)^{-1}\Phi^{T}Rz \]
  with $z = \Phi w^{(r)} - R^{-1}(y - t)$
- Very similar form to pseudo-inverse (normal equations)
  - But now with non-constant weighing matrix $R$ (depends on $w$).
  - Need to apply normal equations iteratively.
  \[ \text{Iteratively Reweighted Least-Squares (IRLS) \footnote{\textit{Machine Learning, Summer'12}}} \]

Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM
- Discriminative Approaches
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees
- Generative Models
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference

Recap: Iterative Methods for Estimation

- Gradient Descent (1st order)
  \[ w^{(r+1)} = w^{(r)} - \eta \nabla E(w) \]
  - Simple and general
  - Relatively slow to converge, has problems with some functions
- Newton-Raphson (2nd order)
  \[ w^{(r+1)} = w^{(r)} - H^{-1} \nabla E(w) \]
  where $H = \nabla^{2}E(w)$ is the Hessian matrix, i.e. the matrix of second derivatives.
  - Local quadratic approximation to the target function
  - Faster convergence
Goal: predict class labels of new observations
- Train classification model on limited training set.
- The further we optimize the model parameters, the more the training error will decrease.
- However, at some point the test error will go up again.
  ⇒ Overfitting to the training set!

Recap: Statistical Learning Theory
- Idea
  - Compute an upper bound on the actual risk based on the empirical risk
    \[ R(\alpha) \leq R_{\text{emp}}(\alpha) + \epsilon(N, p^*, h) \]
  - where
    \[ N: \text{number of training examples} \]
    \[ p^*: \text{probability that the bound is correct} \]
    \[ h: \text{capacity of the learning machine ("VC-dimension")} \]
- VC dimension
  - Vapnik-Chervonenkis dimension
    Measure for the capacity of a learning machine.
  - Formal definition:
    - If a given set of \( L \) points can be labeled in all possible \( 2^L \) ways, and for each labeling, a member of the set \( \{ f(\alpha) \} \) can be found which correctly assigns those labels, we say that the set of points is shattered by the set of functions.
    - The VC dimension for the set of functions \( \{ f(\alpha) \} \) is defined as the maximum number of training points that can be shattered by \( \{ f(\alpha) \} \).

Recap: Upper Bound on the Risk
- Important result (Vapnik 1979, 1995)
  - With probability \((1-\eta)\), the following bound holds
    \[ R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\frac{h \log(2N/h) + 1}{N}} \log(\eta/4) \]
  - This bound is independent of \( P_N,Y(x,y) \)
  - If we know \( h \) (the VC dimension), we can easily compute the risk bound
    \[ R(\alpha) \leq R_{\text{emp}}(\alpha) + \epsilon(N, p^*, h) \]

Recap: Structural Risk Minimization
- How can we implement Structural Risk Minimization?
  \[ R(\alpha) \leq R_{\text{emp}}(\alpha) + \epsilon(N, p^*, h) \]
- Classic approach
  - Keep \( \epsilon(N, p^*, h) \) constant and minimize \( R_{\text{emp}}(\alpha) \).
  - \( \epsilon(N, p^*, h) \) can be kept constant by controlling the model parameters.
- Support Vector Machines (SVMs)
  - Keep \( R_{\text{emp}}(\alpha) \) constant and minimize \( \epsilon(N, p^*, h) \).
  - In fact: \( R_{\text{emp}}(\alpha) = 0 \) for separable data.
  - Control \( \epsilon(N, p^*, h) \) by adapting the VC dimension (controlling the “capacity” of the classifier).
Recap: Support Vector Machine (SVM)

- Basic idea
  - The SVM tries to find a classifier which maximizes the margin between pos. and neg. data points.
  - Up to now: consider linear classifiers $w^T x + b = 0$
- Formulation as a convex optimization problem
  - Find the hyperplane satisfying
    $$\arg \min_{w,b} \frac{1}{2} ||w||^2$$
  - under the constraints
    $$t_n (w^T x_n + b) \geq 1 \quad \forall n$$
  - based on training data points $x_n$ and target values $t_n \in \{-1, 1\}$.

Recap: SVM - Primal Formulation

- Lagrangian primal form
  $$L_p = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} a_n \left(t_n(w^T x_n + b) - 1\right)$$
  $$= \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} a_n \left(t_n y(x_n) - 1\right)$$
- The solution of $L_p$ needs to fulfill the KKT conditions
  - Necessary and sufficient conditions
    $$a_n \geq 0$$
    $$t_n y(x_n) - 1 \geq 0$$
    $$\nabla f(x) = 0$$
    $$\lambda \geq 0$$

Recap: SVM - Support Vectors

- The training points for which $a_n > 0$ are called "support vectors".
- Graphical interpretation:
  - The support vectors are the points on the margin.
  - They define the margin and thus the hyperplane.
  - All other data points can be discarded!

Recap: SVM - Solution

- Solution for the hyperplane
  - Computed as a linear combination of the training examples
    $$w = \sum_{n=1}^{N} a_n t_n x_n$$
  - Sparse solution: $a_n \neq 0$ only for some points, the support vectors
  - Only the SVs actually influence the decision boundary!
  - Compute $b$ by averaging over all support vectors:
    $$b = \frac{1}{|S|} \sum_{n \in S} \left(t_n - \sum_{m \in S} a_m t_m x_m^T x_n\right)$$

Recap: SVM - Dual Formulation

- Maximize
  $$L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m)$$
  under the conditions
    $$\sum_{n=1}^{N} a_n t_n = 0$$
    $$a_n \geq 0 \quad \forall n$$
- Comparison
  - $L_p$ is equivalent to the primal form $L_p$, but only depends on $a_n$.
  - $L_d$ scales with $O(N^2)$.
  - $L_d$ scales with $O(N^2)$ - in practice between $O(N)$ and $O(N^2)$. 
Recap: SVM for Non-Separable Data

- Slack variables
  - One slack variable $\xi_i \geq 0$ for each training data point.
- Interpretation
  - $\xi_i = 0$ for points that are on the correct side of the margin.
  - $\xi_i = |y_i - y(x_i)|$ for all other points.

We do not have to set the slack variables ourselves!
⇒ They are jointly optimized together with $w$.  

Recap: Nonlinear SVMs

- General idea: The original input space can be mapped to some higher-dimensional feature space where the training set is separable: 

Recap: Kernels Fulfilling Mercer’s Condition

- Polynomial kernel
  
- Radial Basis Function kernel
  
- Hyperbolic tangent kernel
    
- And many, many more, including kernels on graphs, strings, and symbolic data...

Recap: SVM - New Dual Formulation

- New SVM Dual: Maximize
  
under the conditions

- This is again a quadratic programming problem
⇒ Solve as before...

Recap: The Kernel Trick

- Important observation
  - $\phi(x)$ only appears in the form of dot products $\phi(x)^T \phi(y)$:
    
  - Define a so-called kernel function $k(x, y) = \phi(x)^T \phi(y)$.
    
  - Now, in place of the dot product, use the kernel instead:
    
- The kernel function implicitly maps the data to the higher-dimensional space (without having to compute $\phi(x)$ explicitly)!

Recap: Nonlinear SVM - Dual Formulation

- SVM Dual: Maximize
  
under the conditions

- Classify new data points using

Recap: Classifier Combination

- We’ve seen already a variety of different classifiers
  - k-NN
  - Bayes classifiers
  - Fisher’s Linear Discriminant
  - SVMs

- Each of them has their strengths and weaknesses...
  - Can we improve performance by combining them?

Recap: Stacking

- Idea
  - Learn $L$ classifiers (based on the training data)
  - Find a meta-classifier that takes as input the output of the $L$ first-level classifiers.

- Example
  - Learn $L$ classifiers with leave-one-out.
  - Interpret the prediction of the $L$ classifiers as $L$-dimensional feature vector.
  - Learn “level-2” classifier based on the examples generated this way.

Recap: Bayesian Model Averaging

- Model Averaging
  - Suppose we have $H$ different models $h = 1,...,H$ with prior probabilities $p(h)$.
  - Construct the marginal distribution over the data set
    \[ p(X) = \sum_{h=1}^{H} p(X|h)p(h) \]
  - Average error of committee
    \[ E_{COM} = \frac{1}{M} E_{AV} \]
    - This suggests that the average error of a model can be reduced by a factor of $M$ simply by averaging $M$ versions of the model!
    - Unfortunately, this assumes that the errors are all uncorrelated.
    - In practice, they will typically be highly correlated.

Recap: Boosting (Schapire 1989)

- Algorithm: (3-component classifier)
  1. Sample $N_1 < N$ training examples (without replacement) from training set $D$ to get set $D_1$.
     - Train weak classifier $C_1$ on $D_1$.
  2. Sample $N_2 < N$ training examples (without replacement), half of which were misclassified by $C_1$, to get set $D_2$.
     - Train weak classifier $C_2$ on $D_2$.
  3. Choose all data in $D$ on which $C_1$ and $C_2$ disagree to get set $D_3$.
     - Train weak classifier $C_3$ on $D_3$.
  4. Get the final classifier output by majority voting of $C_1$, $C_2$, and $C_3$.
    (Recursively apply the procedure on $C_3$ to $C_0$)
Recap: AdaBoost - “Adaptive Boosting”

- **Main idea**
  - Instead of resampling, reweight misclassified training examples.
  - Increase the chance of being selected in a sampled training set.
  - Or increase the misclassification cost when training on the full set.

- **Components**
  - $h_m(x)$: “weak” or base classifier
    - Condition: <50% training error over any distribution
  - $H(x)$: “strong” or final classifier

- **AdaBoost**
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:
    $$H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)$$

Recap: AdaBoost - Intuition

Consider a 2D feature space with positive and negative examples.

Each weak classifier splits the training examples with at least 50% accuracy.

Examples misclassified by a previous weak learner are given more emphasis at future rounds.

Recap: AdaBoost - Algorithm

1. Initialization: Set $w_1^{(1)} = \frac{1}{N}$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$ iterations
   a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W(m)$ by minimizing the weighted error function
      $$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)$$
      $$f_t = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{else} \end{cases}$$
   b) Estimate the weight of this classifier on $X$:
      $$\alpha_m = \frac{1}{2} \ln \frac{1}{J_m}$$
   c) Calculate a weighting coefficient for $h_m(x)$:
      $$\alpha_m = \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)$$
   d) Update the weighting coefficients:
      $$w_{n}^{(m+1)} = w_{n}^{(m)} \exp \left( \alpha_m f_t(n, h_m(x_n) \neq t_n) \right)$$
Recap: Comparing Error Functions

- Ideal classification error function
- "Hinge error" used in SVMs
- Exponential error function
- "Cross-entropy error" – Similar to exponential error for $z > 0$.
  - Only grows linearly with large negative values of $z$.
  - Make AdaBoost more robust by switching to "GentleBoost"

Recap: Decision Trees

- Example:
  - "Classify Saturday mornings according to whether they’re suitable for playing tennis."

Recap: CART Framework

- Six general questions
  1. Binary or multi-valued problem? (I.e. how many splits should there be at each node?)
  2. Which property should be tested at a node? (I.e. how to select the query attribute?)
  3. When should a node be declared a leaf? (I.e. when to stop growing the tree?)
  4. How can a grown tree be simplified or pruned? (Goal: reduce overfitting.)
  5. How to deal with impure nodes? (I.e. when the data itself is ambiguous.)
  6. How should missing attributes be handled?

Recap: Picking a Good Splitting Feature

- Goal
  - Select the query (split) that decreases impurity the most

- Impurity measures
  - Entropy impurity (information gain):
    \[ i(N) = -\sum_{c_i} p(c_i|N) \log_2 p(c_i|N) \]
  - Gini impurity:
    \[ i(N) = \sum_{c_i} p(c_i|N)p(c_i|N) = \frac{1}{2} \left[ 1 - \sum_{i \neq j} p_i p_j (c_i|N) \right] \]

Recap: Overfitting Prevention (Pruning)

- Two basic approaches for decision trees
  - Prepruning: Stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
    - Cross-validation
    - Chi-square test
    - MDL
  - Postpruning: Grow the full tree, then remove subtrees that do not have sufficient evidence.
    - Merging nodes
    - Rule-based pruning

- In practice often preferable to apply post-pruning.
Recap: ID3 Algorithm

- **ID3 (Quinlan 1986)**
  - One of the first widely used decision tree algorithms.
  - Intended to be used with nominal (unordered) variables.
  - Real variables are first binned into discrete intervals.
  - General branching factor
    - Use gain ratio impurity based on entropy (information gain) criterion.

- **Algorithm**
  - Select attribute \( a \) that best classifies examples, assign it to root.
  - For each possible value \( v_i \) of \( a \),
    - Add new tree branch corresponding to test \( a = v_i \).
    - If \( \text{example_list}(v_i) \) is empty, add leaf node with most common label in \( \text{example_list}(a) \).
    - Else, recursively call ID3 for the subtree with attributes \( A \setminus a \).

Recap: C4.5 Algorithm

- **C4.5 (Quinlan 1993)**
  - Improved version with extended capabilities.
  - Ability to deal with real-valued variables.
  - Multiway splits are used with nominal data
    - Using gain ratio impurity based on entropy (information gain) criterion.
  - Heuristics for pruning based on statistical significance of splits.
  - Rule post-pruning
    - Main difference to CART
    - Strategy for handling missing attributes.
    - When missing feature is queried, C4.5 follows all possible answers.
    - Decision is made based on all possible outcomes, weighted by decision probabilities at node \( N \).

Recap: Computational Complexity

- **Given**
  - Data points \( \{x_1, \ldots, x_N\} \)
  - Dimensionality \( D \)
- **Complexity**
  - Storage: \( O(N) \)
  - Test runtime: \( O(\log N) \)
  - Training runtime: \( O(DN^2 \log N) \)
    - Most expensive part.
    - Critical step: selecting the optimal splitting point.
    - Need to check \( D \) dimensions, for each need to sort \( N \) data points.
      \( O(DN \log N) \)

Recap: Decision Trees - Summary

- **Properties**
  - Simple learning procedure, fast evaluation.
  - Can be applied to metric, nominal, or mixed data.
  - Often yield interpretable results.

- **Limitations**
  - Often produce noisy (bushy) or weak (stunted) classifiers.
  - Do not generalize too well.
  - Training data fragmentation:
    - As tree progresses, splits are selected based on less and less data.
  - Overtraining and undert raining:
    - Deep trees: fit the training data well, will not generalize well to new test data.
    - Shallow trees: not sufficiently refined.
  - Stability:
    - Trees can be very sensitive to details of the training points.
      - If a single data point is only slightly shifted, a radically different tree may come out!
    - Result of discrete and greedy learning procedure.
  - Expensive learning step
    - Mostly due to costly selection of optimal split.

Course Outline

- **Fundamentals**
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM
- **Discriminative Approaches**
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees
- **Generative Models**
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference
Recap: Randomized Decision Trees

- Decision trees: main effort on finding good split
  - Training runtime: $O(DN^2 \log N)$
  - This is what takes most effort in practice.
  - Especially cumbersome with many attributes (large $D$).
- Idea: randomize attribute selection
  - No longer look for globally optimal split.
  - Instead randomly use subset of $K$ attributes on which to base the split.
  - Choose best splitting attribute e.g. by maximizing the information gain (= reducing entropy):
    $$\Delta E = \sum_{k=1}^{K} \left( \sum_{j=1}^{N} p_j \log_2(p_j) \right)$$

Recap: Ensemble Combination

- Ensemble combination
  - Tree leaves $(l,u)$ store posterior probabilities of the target classes.
  - Combine the output of several trees by averaging their posteriors (Bayesian model combination)
    $$p(C|x) = \frac{1}{L} \sum_{l=1}^{L} p_{l,u}(C|x)$$

Recap: Random Forests (Breiman 2001)

- General ensemble method
  - Idea: Create ensemble of many (50 - 1,000) trees.
- Empirically very good results
  - Often as good as SVMs (and sometimes better!)
  - Often as good as Boosting (and sometimes better!)
- Injecting randomness
  - Bootstrap sampling process
    - On average only 63% of training examples used for building the tree
    - Remaining 37% out-of-bag samples used for validation.
  - Random attribute selection
    - Randomly choose subset of $K$ attributes to select from at each node.
    - Faster training procedure.
- Simple majority vote for tree combination

Recap: Extremely Randomized Decision Trees

- Random queries at each node...
  - Tree gradually develops from a classifier to a flexible container structure.
  - Node queries define (randomly selected) structure.
  - Each leaf node stores posterior probabilities
- Learning
  - Patches are “dropped down” the trees.
    - Only pairwise pixel comparisons at each node.
    - Directly update posterior distributions at leaves
  - Very fast procedure, only few pixel-wise comparisons.
  - No need to store the original patches!
**Course Outline**

- **Fundamentals**
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM

- **Discriminative Approaches**
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees

- **Generative Models**
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference

**Recap: Graphical Models**

- Two basic kinds of graphical models
  - Directed graphical models or Bayesian Networks
  - Undirected graphical models or Markov Random Fields

- **Key components**
  - **Nodes**
    - Random variables
  - **Edges**
    - Directed or undirected
  - The value of a random variable may be known or unknown.

**Recap: Directed Graphical Models**

- Chains of nodes:
  
  \[
  p(a) \quad p(b|a) \quad p(c|b) \\
  \]

  - Knowledge about \( a \) is expressed by the prior probability:
    \( p(a) \)
  - Dependencies are expressed through conditional probabilities:
    \( p(b|a), p(c|b) \)
  - Joint distribution of all three variables:
    \[
    p(a, b, c) = p(c|a, b)p(a, b) \\
    = p(c|b)p(b|a)p(a)
    \]

**Recap: Factorization of the Joint Probability**

- Computing the joint probability:
  \[
  p(x_1, \ldots, x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3) \\
  p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5)
  \]

  General factorization
  \[
  p(x) = \prod_{k=1} \prod_{a_k} p(x_k|p_{a_k})
  \]

  We can directly read off the factorization of the joint from the network structure!

**Recap: Factorized Representation**

- **Reduction of complexity**
  - Joint probability of \( n \) binary variables requires us to represent values by brute force
    \[
    O(2^n) \quad \text{terms}
    \]
  - The factorized form obtained from the graphical model only requires
    \[
    O(n \cdot 2^k) \quad \text{terms}
    \]
    - \( k \): maximum number of parents of a node.

\[\Rightarrow \text{It’s the edges that are missing in the graph that are important! They encode the simplifying assumptions we make.}\]
Recap: Conditional Independence

- \( X \) is conditionally independent of \( Y \) given \( V \)
  - Definition: \( X \perp Y \mid V \Rightarrow p(X,Y,V) = p(X,V)p(Y|V) \)
  - Also: \( X \perp Y \mid V \Rightarrow p(X,Y|V) = p(X|V)p(Y|V) \)
- Special case: Marginal Independence
  \( X \perp Y \mid \emptyset \Rightarrow p(X,Y) = p(X)p(Y) \)

- Often, we are interested in conditional independence between sets of variables:
  \( X \perp Y \mid V \Rightarrow \{X \perp Y \mid \forall X \in X \text{ and } \forall Y \in Y\} \)

Recap: Conditional Independence

- Three cases
  - Divergent ("Tail-to-Tail")
    - Conditional independence when \( c \) is observed.
  - Chain ("Head-to-Tail")
    - Conditional independence when \( c \) is observed.
  - Convergent ("Head-to-Head")
    - Conditional independence when neither \( c \), nor any of its descendants are observed.

Recap: D-Separation

- Definition
  - Let \( A, B, \) and \( C \) be non-intersecting subsets of nodes in a directed graph.
  - A path from \( A \) to \( B \) is **blocked** if it contains a node such that either
    - The arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in the set \( C \), or
    - The arrows meet head-to-head at the node, and neither the node, nor any of its descendants, are in the set \( C \).
  - If all paths from \( A \) to \( B \) are blocked, \( A \) is said to be d-separated from \( B \) by \( C \).
- If \( A \) is d-separated from \( B \) by \( C \), the joint distribution over all variables in the graph satisfies \( A \perp B \mid C \).
- Read: "\( A \) is conditionally independent of \( B \) given \( C \)."

Recap: “Bayes Ball” Algorithm

- Graph algorithm to compute d-separation
  - Goal: Get a ball from \( X \) to \( Y \) without being blocked by \( V \).
  - Depending on its direction and the previous node, the ball can
    - Pass through (from parent to all children, from child to all parents)
    - Bounce back (from any parent/child to all parents/children)
    - Be blocked

- Game rules
  - An unobserved node \( (W \notin V) \) passes through balls from parents, but also bounces back balls from children.
  - An observed node \( (W \in V) \) bounces back balls from parents, but blocks balls from children.

Recap: The Markov Blanket

- Markov blanket of a node \( x \)
  - Minimal set of nodes that isolates \( x \), from the rest of the graph.
  - This comprises the set of
    - Parents,
    - Children, and
    - Co-parents of \( x \).

Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM
- Discriminative Approaches
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees
- Generative Models
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference
Recap: Undirected Graphical Models

- Undirected graphical models ("Markov Random Fields")
  - Given by undirected graph

- Conditional independence for undirected graphs
  - If every path from any node in set \( A \) to set \( B \) passes through at least one node in set \( C \), then \( A \perp B | C \).
  - Simple Markov blanket:

Recap: Factorization in MRFs

- Joint distribution
  - Written as product of potential functions over maximal cliques in the graph:
  
  \[ p(x) = \frac{1}{Z} \prod_C \psi_C(x_C) \]

  - The normalization constant \( Z \) is called the partition function.

- Remarks
  - BNs are automatically normalized. But for MRFs, we have to explicitly perform the normalization.
  - Presence of normalization constant is major limitation!
  - Evaluation of \( Z \) involves summing over \( O(K^M) \) terms for \( M \) nodes!

Factorization in MRFs

- Role of the potential functions
  - General interpretation
    - No restriction to potential functions that have a specific probabilistic interpretation as marginals or conditional distributions.
  - Convenient to express them as exponential functions ("Boltzmann distribution")
  
  \[ \psi_C(x_C) = \exp\{-E(x_C)\} \]

  - with an energy function \( E \).

  - Why is this convenient?
    - Joint distribution is the product of potentials \( \Rightarrow \) sum of energies.
    - We can take the log and simply work with the sums...

Recap: Converting Directed to Undirected Graphs

- Problematic case: multiple parents
  - Need to introduce additional links ("marry the parents").
  - This process is called moralization. It results in the moral graph.

Recap: Conversion Algorithm

- General procedure to convert directed \( \rightarrow \) undirected
  1. Add undirected links to marry the parents of each node.
  2. Drop the arrows on the original links \( \Rightarrow \) moral graph.
  3. Find maximal cliques for each node and initialize all clique potentials to 1.
  4. Take each conditional distribution factor of the original directed graph and multiply it into one clique potential.

  - Restriction
    - Conditional independence properties are often lost!
    - Moralization results in additional connections and larger cliques.

Recap: Computing Marginals

- How do we apply graphical models?
  - Given some observed variables, we want to compute distributions of the unobserved variables.
  - In particular, we want to compute marginal distributions, for example \( p(x_i) \).

  - How can we compute marginals?
    - Classical technique: sum-product algorithm by Judea Pearl.
    - In the context of (loopy) undirected models, this is also called (loopy) belief propagation [Weiss, 1997].
    - Basic idea: message-passing.
Recap: Message Passing on a Chain

- Idea
  - Pass messages from the two ends towards the query node $x_w$.
- Define the messages recursively:
  \[ \mu_a(x_n) = \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_a(x_{n-1}) \]
  \[ \mu_b(x_n) = \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \mu_b(x_{n+1}) \]
- Compute the normalization constant $Z$ at any node $x_m$,
  \[ Z = \sum_{x_m} \mu_a(x_m) \mu_b(x_m) \]

Recap: Message Passing on Trees

- General procedure for all tree graphs.
  - Root the tree at the variable that we want to compute the marginal of.
  - Start computing messages at the leaves.
  - Compute the messages for all nodes for which all incoming messages have already been computed.
  - Repeat until we reach the root.
- If we want to compute the marginals for all possible nodes (roots), we can reuse some of the messages.
- Computational expense linear in the number of nodes.
- We already motivated message passing for inference.
  - How can we formalize this into a general algorithm?

Recap: Factor Graphs

- Joint probability
  - Can be expressed as product of factors: $p(x) = \frac{1}{Z} \prod_i f_i(x_i)$
  - Factor graphs make this explicit through separate factor nodes.
- Converting a directed polytree
  - Conversion to undirected tree creates loops due to moralization!
  - Conversion to a factor graph again results in a tree!

Recap: Sum-Product Algorithm

- Objectives
  - Efficient, exact inference algorithm for finding marginals.
- Procedure:
  - Pick an arbitrary node as root.
  - Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
  - Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
  - Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.
  \[ p(x) \propto \prod_{x \in \text{root}(x)} \mu_{f_i \rightarrow x}(x) \]
- Computational effort
  - Total number of messages $\approx 2 \cdot \text{number of graph edges}$.
Recap: Sum-Product from Leaves to Root

Message definitions:

\[ \mu_{f_i \rightarrow x}(x) \equiv \sum_{x_i} f_i(x_i) \prod_{m \in \text{net}(f_i) \setminus x} \mu_{x_m \rightarrow f_i}(x_m) \]
\[ \mu_{x_m \rightarrow f_i}(x_m) \equiv \prod_{l \in \text{net}(x_m) \setminus f_i} \mu_{x_l \rightarrow x_m}(x_l) \]
\[ \mu_{x_l \rightarrow x_m}(x_l) = 1 \quad \mu_{f_i \rightarrow x}(x) = f(x) \]

Recap: Sum-Product from Root to Leaves

Message definitions:

\[ \mu_{f_i \rightarrow x}(x) \equiv \sum_{x_i} f_i(x_i) \prod_{m \in \text{net}(f_i) \setminus x} \mu_{x_m \rightarrow f_i}(x_m) \]
\[ \mu_{x_m \rightarrow f_i}(x_m) \equiv \prod_{l \in \text{net}(x_m) \setminus f_i} \mu_{x_l \rightarrow x_m}(x_l) \]
\[ \mu_{x_l \rightarrow x_m}(x_l) = 1 \quad \mu_{f_i \rightarrow x}(x) = f(x) \]

Recap: Max-Sum Algorithm

- Objective: an efficient algorithm for finding
  - Value \( x^{\text{max}} \) that maximises \( p(x) \);
  - Value of \( p(x^{\text{max}}) \).
  \( \Rightarrow \) Application of dynamic programming in graphical models.

- Key ideas
  - We are interested in the maximum value of the joint distribution
  \( p(x^{\text{max}}) = \max_x p(x) \)
  \( \Rightarrow \) Maximize the product \( p(x) \).
  - For numerical reasons, use the logarithm.
  \( \ln \left( \max_x p(x) \right) = \max_x \ln p(x) \)
  \( \Rightarrow \) Maximize the sum (of log-probabilities).

Recap: Junction Tree Algorithm

- Motivation
  - Exact inference on general graphs.
  - Works by turning the initial graph into a junction tree and then running a sum-product-like algorithm.
  - Intractable on graphs with large cliques.

- Main steps
  1. If starting from directed graph, first convert it to an undirected graph by moralization.
  2. Introduce additional links by triangulation in order to reduce the size of cycles.
  3. Find cliques of the moralized, triangulated graph.
  4. Construct a new graph from the maximal cliques.
  5. Remove minimal links to break cycles and get a junction tree.
  \( \Rightarrow \) Apply regular message passing to perform inference.
Recap: Junction Tree Example

• Without triangulation step
  - The final graph will contain cycles that we cannot break without losing the running intersection property!

Recap: Junction Tree Example

• When applying the triangulation
  - Only small cycles remain that are easy to break.
  - Running intersection property is maintained.

Course Outline

• Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM

• Discriminative Approaches
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees

• Generative Models
  - Bayesian Networks
  - Markov Random Fields & Applications
  - Exact Inference

Recap: MRF Structure for Images

• Basic structure

Recap: How to Set the Potentials?

• Unary potentials
  - E.g. color model, modeled with a Mixture of Gaussians
  \[
  \phi(x_i, y_i; \theta_q) = \log \sum_k \theta_q(x_i, k) p(k|x_i) \mathcal{N}(y_i; \bar{y}_k, \Sigma_k)
  \]

  \[\Rightarrow \text{Learn color distributions for each label}\]

Recap: How to Set the Potentials?

• Pairwise potentials
  - Potts Model
    \[
    \psi(x_i, x_j; \theta_q) = \theta_q \delta(x_i \neq x_j)
    \]
  - Simplest discontinuity preserving model.
  - Discontinuities between any pair of labels are penalized equally.
  - Useful when labels are unordered or number of labels is small.

  • Extension: “contrast sensitive Potts model”
    \[
    \psi(x_i, x_j; g(y); \theta_q) = \theta_q g(y) \delta(x_i \neq x_j)
    \]
    where
    \[
    g(y) = e^{-\beta \| y_i - y_j \|^2}
    \]
    - Discourages label changes except in places where there is also a large change in the observations.
Recap: Graph Cuts for Binary Problems

“expected” intensities of object and background can be re-estimated

Recap: s-t-Mincut Equivalent to Maxflow

Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Repeat until no path can be found

Algorithms assume non-negative capacity

Recap: When Can s-t Graph Cuts Be Applied?

Regional term
Boundary term

Submodularity is the discrete equivalent to convexity.
- Implies that every local energy minimum is a global minimum.
- Solution will be globally optimal.

Recap: Simple Binary Image Denoising Model

- MRF Structure

Recap: Converting an MRF into an s-t Graph

Conversion:

Energy:

 Unary potentials are straightforward to set.
- Just insert \( x_i = 1 \) and \( x_i = 0 \) into the unary terms above...
Recap: Converting an MRF into an s-t Graph
• Conversion:

\[ E(x; y) = h \sum_i x_i + \eta \sum_{(i,j)} \delta(x_i \neq x_j) - \eta \sum_i x_i y_i \]

- Unary potentials are straightforward to set.
- Pairwise potentials are more tricky, since we don’t know \(x_i\).
  - Trick: the pairwise energy only has an influence if \(x_i \neq x_j\).
  - (Only!) in this case, the cut will go through the edge \(\{x_i, x_j\}\).

Any Questions?

So what can you do with all of this?

Mobile Object Detection & Tracking

Master Thesis: Image-Based Localization

Any More Questions?

Good luck for the exam!