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.

Announcements (2)

- Test exam on Thursday
  - During the regular lecture slot
  - Duration: 1h (instead of 2h as for the real exam)
  - Purpose: prepare you for the questions you can expect
  - All bonus points!

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

- 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: \( R_1, R_2, R_3, \ldots \)

Recap: Classifying with Loss Functions

- 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

Recap: Minimizing the Expected Loss

- Optimal solution minimizes the loss.
  - But: loss function depends on the true class, which is unknown.

  Solution: Minimize the expected loss

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

  This can be done by choosing the regions \( R_j \) such that

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

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

Recap: The Reject Option

- Classification errors arise from regions where the largest posterior probability \( p(C_k|x) \) is significantly less than 1.
  - These are the regions where we are relatively uncertain about class membership.
  - For some applications, it may be better to reject the automatic decision entirely in such a case and e.g. consult a human expert.

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 \( \Sigma \)

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

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: 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 (i.e. minimize the negative log-likelihood)
  - Take the derivative and set it to zero.
  \[ \frac{\partial}{\partial \theta} E(\theta) = -\sum_{n=1}^{N} \frac{p(x_n|\theta)}{p(x_n|\theta)} = 0 \]

Recap: Bayesian Learning Approach

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

Recap: Histograms

- Basic idea:
  - Partition the data space into distinct bins with widths \( \Delta \), and count the number of observations, \( n_i \), in each bin.
  \[ p_i = \frac{n_i}{N\Delta} \]
  - Often, the same width is used for all bins, \( \Delta = \Delta \).
  - This can be done, in principle, for any dimensionality \( D \).

Recap: Kernel Density Estimation

- Approximation formula:
  \[ p(x) \approx \frac{K}{N} \]
  - \( K \) determine \( V \)
  - \( \Delta \) determine \( K \)
- Kernel Methods
  - \( \Delta \) nearest neighbor
  - \( K \) nearest neighbor
- \( K \) nearest neighbor
  - Increase the volume \( V \) until the \( K \) next data points are found.

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: Mixture of Gaussians (MoG)

- "Generative model"

\[ p(j) = \pi_j \]

"Weight" of mixture component

\[ p(x) \]

\[ p(x|\theta_j) \]

Mixture component

\[ p(x) = \sum_{j=1}^{M} p(x|\theta_j)p(j) \]

Recap: MoG - Iterative Strategy

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

\[ f(x) \]

ML for Gaussian #1

ML for Gaussian #2

\[ h(j = 1|x_n) = \begin{cases} 1 & 111 \\ 0 & 000 \end{cases} \]

\[ h(j = 2|x_n) = \begin{cases} 22 & 2 \\ 11 & 1 \end{cases} \]

Recap: MoG - Iterative Strategy

- Assuming we knew the mixture components...

\[ f(x) \]

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

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

Bayes decision rule: Decide \( j = 1 \) if

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

Recap: EM Algorithm

- Expectation-Maximization (EM) Algorithm

  E-Step: softly assign samples to mixture components

  \[ \gamma_j(x_n) \leftarrow \frac{\pi_j N(x_n|\mu_j, \Sigma_j)}{\sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \Sigma_k)} \quad \forall j = 1, \ldots, K; \quad n = 1, \ldots, N \]

  M-Step: re-estimate the parameters (separately for each mixture component) based on the soft assignments

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

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

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

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: Linear Discriminant Functions

- Basic idea
  - Directly encode decision boundary
  - Minimize misclassification probability directly.
- Linear discriminant functions
  \[ y(x) = w^T x + w_0 \]
  weight vector, "bias" (= threshold)
  - \( w, w_0 \) 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.

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 monotonous (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 \( \phi_j(\cdot) \):
  \[ y_k(x) = \sum_{j=1}^{M} w_k \phi_j(x) + w_{k0} \]
- 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.
  \( \Rightarrow \) Minimization with Gradient Descent
Recap: Classification as Dimensionality Reduction

- Classification as dimensionality reduction
  - Interpret linear classification as a projection onto a lower-dim. space. 
  - Learning problem: Try to find the projection vector $\mathbf{w}$ that maximizes class separation.

Recap: Properties

- Need to apply normal equations iteratively.
- Faster convergence
- Optimal solution $\mathbf{w}$

Maximize distance between classes

Criterion: $J(\mathbf{w}) = \mathbf{w}^T \mathbf{S}_B \mathbf{w} / \mathbf{w}^T \mathbf{S}_W \mathbf{w}$

- $\mathbf{S}_B$: between-class scatter matrix
- $\mathbf{S}_W$: within-class scatter matrix
- The optimal solution $\mathbf{w}$ can be obtained as:

$$\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

Classification function:

$$y(x) = \mathbf{w}^T \mathbf{x} + m_2 \quad \text{Class } 2$$

Recap: Logistic Regression

- Let's consider a data set $\{\phi_i, t_i\}$ with $i = 1, \ldots, N$.
- $t_i \in \{0, 1\}$, $\mathbf{t} = (t_1, \ldots, t_N)^T$.
- With $y_i = p(C_2|\phi_i)$, we can write the likelihood as

$$p(\mathbf{t}|\mathbf{w}) = \prod_{i=1}^N y_i^{t_i} (1 - y_i)^{1 - t_i}$$

- Define the error function as the negative log-likelihood

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w})$$

$$= -\sum_{i=1}^N \{t_i \ln y_i + (1 - t_i) \ln(1 - y_i)\}$$

This is the so-called cross-entropy error function.

Recap: Iterative Methods for Estimation

- Gradient Descent (1st order)

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \eta \nabla E(\mathbf{w}) \bigg|_{\mathbf{w}^{(r)}}$$

- Simple and general
- Relatively slow to converge, has problems with some functions

- Newton-Raphson (2nd order)

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \eta \mathbf{H}^{-1} \nabla E(\mathbf{w}) \bigg|_{\mathbf{w}^{(r)}}$$

where $\mathbf{H} = \nabla \nabla E(\mathbf{w})$ is the Hessian matrix, i.e. the matrix of second derivatives.

- Local quadratic approximation to the target function
- Faster convergence

Recap: Iteratively Reweighted Least Squares

- Update equations

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - (\mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi})^{-1} \mathbf{\Phi}^T (\mathbf{y} - \mathbf{t})$$

$$= (\mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi})^{-1} \left[ \mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi} \mathbf{w}^{(r)} - \mathbf{\Phi}^T (\mathbf{y} - \mathbf{t}) \right]$$

$$= (\mathbf{\Phi}^T \mathbf{R}^2 \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{R} \mathbf{z}$$

with $\mathbf{z} = \mathbf{\Phi} \mathbf{w}^{(r)} - \mathbf{R}^{-1} (\mathbf{y} - \mathbf{t})$

- Very similar form to pseudo-inverse (normal equations)
  - But now with non-constant weighting matrix $\mathbf{R}$ (depends on $\mathbf{w}$)
  - Need to apply normal equations iteratively.

$\Rightarrow$ Iteratively Reweighted Least-Squares (IRLS)
The further we optimize the model parameters, the more the
actual risk will increase. However, at some point the training error will go up again.
\[ \Rightarrow \text{Overfitting to the training set} \]

**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
    - \( N \): number of training examples
    - \( p^* \): probability that the bound is correct
    - \( h \): capacity of the learning machine ("VC-dimension")

**Recap: Upper Bound on the Risk**

- **Important result (Vapnik 1979, 1995)**
  - With probability \( 1-\eta \), the following bound holds
  \[ R(\alpha) \leq R_{\text{emp}}(\alpha) + \epsilon(N, p^*, h) \]
  - This bound is independent of \( P_{X,Y}(x,y) \).
  - 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: VC Dimension**

- **Vapnik-Chervonenkis dimension**
  - Measure for the capacity of a learning machine.
- **Formal definition:**
  - If a given set of \( \ell \) points can be labeled in all possible \( 2^\ell \) 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: Structural Risk Minimization

- How can we implement Structural Risk Minimization?
  \[ R(\alpha) = R_{\text{emp}}(\alpha) + \epsilon(N, p^*, h) \]

- 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
  \[ L_p = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} a_n \left( t_n (w^T x_n + b) - 1 \right) \]

  \[ = \frac{1}{2} ||w||^2 - \sum_{n=1}^{N} a_n \{ t_n y(x_n) - 1 \} \]

- The solution of \( L_p \) needs to fulfill the KKT conditions
  - Necessary and sufficient conditions
    \[ a_n \geq 0 \]
    \[ t_n y(x_n) - 1 \geq 0 \]
    \[ a_n \{ t_n y(x_n) - 1 \} = 0 \]
    \[ f(x) \geq 0 \]
    \[ \lambda f(x) = 0 \]

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
  \[ \Rightarrow \] Only the SVs actually influence the decision boundary!

  - Compute \( b \) by averaging over all support vectors:
    \[ b = \frac{1}{N_S} \sum_{n \in S} \left( t_n - \sum_{m \in S} a_m t_m x_m^T x_n \right) \]

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.
  \[ \Rightarrow \] All other data points can be discarded!
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
  
  \[ a_n \geq 0 \quad \forall n \]
  \[ \sum_{n=1}^{N} a_n t_n = 0 \]

- Comparison
  
  \( \phi(x) \) is equivalent to the primal form \( L_p \), but only depends on \( a_n \).
  
  \( L_d \) scales with \( O(D) \).
  
  \( L_d \) scales with \( O(N^2) \) - in practice between \( O(N) \) and \( O(N^2) \).

Recap: SVM - New Dual Formulation

- New SVM Dual: 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
  
  \[ 0 \leq a_n \leq C \]
  \[ \sum_{n=1}^{N} a_n t_n = 0 \]

  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 \( \phi(x)^T \phi(y) \):

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

  \[ = \sum_{n=1}^{N} a_n t_n \phi(x_n)^T \phi(x) + b \]

  - 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:

    \[ y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b \]

    The kernel function implicitly maps the data to the higher-dimensional space (without having to compute \( \phi(x) \) explicitly)!

Recap: Kernels Fulfilling Mercer’s Condition

- Polynomial kernel

  \[ k(x,y) = (x^T y + 1)^p \]

- Radial Basis Function kernel

  \[ k(x,y) = \exp \left\{ -\frac{(x - y)^2}{2\sigma^2} \right\} \]

  e.g. Gaussian

- Hyperbolic tangent kernel

  \[ k(x,y) = \tanh(\kappa x^T y + \delta) \]

  e.g. Sigmoid

  - And many, many more, including kernels on graphs, strings, and symbolic data...

Recap: SVM for Non-Separable Data

- Slack variables

  - One slack variable \( \xi_n \geq 0 \) for each training data point.
  
  - Interpretation

    \( \xi_n = 0 \) for points that are on the correct side of the margin.
    
    \( \xi_n = |y(x)| \) for all other points.

- We do not have to set the slack variables ourselves!

  ⇒ They are jointly optimized together with \( w \).
Recap: Kernels Fulfilling Mercer’s Condition

- Polynomial kernel
  \[ k(x, y) = (x^T y + 1)^p \]
- Radial Basis Function kernel
  \[ k(x, y) = \exp \left\{ -\frac{(x - y)^2}{2\sigma^2} \right\} \quad \text{e.g. Gaussian} \]
- Hyperbolic tangent kernel
  \[ k(x, y) = \tanh(ax^T y + b) \quad \text{e.g. Sigmoid} \]

And many, many more, including kernels on graphs, strings, and symbolic data...

Recap: Nonlinear SVM - Dual Formulation

- SVM Dual: Maximize
  \[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m y_n y_m k(x_n, x_m) \]
  under the conditions
  \[ \sum_{n=1}^{N} a_n = 0 \]
  \[ 0 \leq a_n \leq C \]

- Classify new data points using
  \[ y(x) = \sum_{n=1}^{N} a_n y_n k(x_n, x) + b \]

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.

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: Bayesian Model Averaging

- Model Averaging
  - Suppose we have $H$ different models $h = 1, \ldots, H$ 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
  - 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: 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 $w_m^{(1)} = \frac{1}{N}$ for $m = 1, \ldots, N$.
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
      \[ J_m = \sum_{n=1}^{N} w_m^{(m)} I(h_m(x) \neq t_n) \]
   b) Estimate the weighted error of this classifier on $X$:
      \[ \epsilon_m = \sum_{n=1}^{N} w_m^{(m)} I(h_m(x) \neq t_n) \]
   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:
      \[ w_{m+1}^{(m+1)} = w_m^{(m)} \exp \{ \alpha_m I(h_m(x) \neq t_n) \} \]
Recap: Comparing Error Functions

- Ideal misclassification error function
- “Hinge error” used in SVMs
- Exponential error function
  - Continuous approximation to ideal misclassification function.
  - Sequential minimization leads to simple Adaboost scheme.
  - Disadvantage: exponential penalty for large negative values!

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
    \[ \Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R) \]

- Impurity measures
  - Entropy impurity (information gain):
    \[ i(N) = - \sum_{C_j} p(C_j|N) \log_2 p(C_j|N) \]
  - Gini impurity:
    \[ i(N) = \sum_{C_j} p(C_j|N) p(C_j|N) = \frac{1}{2} \left( 1 - \sum_{C_j} p(C_j|N) \right) \]
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.
  - As tree progresses, splits are selected based on less and less data.
  - 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.

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):
    \[
    \Delta E = \sum_{k=1}^{K} \left( \frac{|S_k|}{N} \right) \sum_{j=1}^{N} p_j \log_2(p_j)
    \]

Recap: Ensemble Combination

- **Ensemble combination**
  - Tree leaves \((l, i)\) store posterior probabilities of the target classes.
    \[
    p_{l, i}(C|x)
    \]
  - Combine the output of several trees by averaging their posteriors (Bayesian model combination)
    \[
    p(C|x) = \frac{1}{T} \sum_{l=1}^{T} p_{l, i}(C|x)
    \]
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: A Graphical Interpretation

Different trees induce different partitions on the data.

By combining them, we obtain a finer subdivision of the feature space...

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!

Recap: Ferns

- Ferns are semi-naïve Bayes classifiers.
  - They assume independence between sets of features (between the ferns)...
  - ...and enumerate all possible outcomes inside each set.
- Interpretation
  - Combine the tests \( f_1, \ldots, f_{S/2} \) into a binary number.
  - Update the “fern leaf” corresponding to that number.

\[
p(f_1; \ldots; f_{S/2} | \mathcal{C}_k) \approx \prod_{j=1}^M p(F_j | \mathcal{C}_k)
\]

Recap: Ferns (Semi-Naïve Bayes Classifiers)

- Ferns
  - A fern \( F \) is defined as a set of \( S \) binary features \( \{f_1, \ldots, f_S\} \).
  - \( M \): number of ferns, \( N_j = S \cdot M \).
  - This represents a compromise:
    \[
p(f_1, \ldots, f_S | \mathcal{C}_k) \approx \prod_{j=1}^M p(F_j | \mathcal{C}_k)
= p(f_1 | \mathcal{C}_k) \cdot p(f_2 | \mathcal{C}_k) \cdot \cdots
\]

\( \Rightarrow \) Model with \( M \cdot 2^S \) parameters (“Semi-Naïve”).
\( \Rightarrow \) Flexible solution that allows complexity/performance tuning.
Fundamentals
Random variables
• The value of a random variable may be
Discriminative Approaches
Reduction of complexity
Bayesian Networks
Two basic kinds of graphical models
Probability Density Estimation
Mixture Models and EM
Joint probability of
• The factorized form obtained from the graphical model only
Dependences are expressed through
• The value of a random variable may be known or unknown.
Convergent connections
Chains of nodes
• Here the value of c depends on both variables a and b.
• This is modeled with the conditional probability:
Directed graphical models
• Directed or undirected
Undirected graphical model
• The joint probability of all three variables is given as:
Generative Models
Joint distribution
• Knowledge about a is expressed by the prior probability:
Directed or undirected
Computing the joint probability
Statistical Learning Theory & SVMs
Bayes Decision Theory
Markov Random Fields
Random variables
• Dependencies are expressed through conditional probabilities:
• Joint distribution of all three variables:
Bayesian Networks
• Directed graphical models or Bayesian Networks
• Mixture models and EM
• Ensemble Methods & Boosting
• Markov Random Fields
• Exact Inference
• Bayesian Networks
• Markov Random Fields
• Decision Trees & Randomized Trees
• Linear Discriminant Functions
• Statistical Learning Theory & SVMs
• Ensemble Methods & Boosting
• Decision Trees & Randomized Trees
Course Outline
Recap: Directed Graphical Models
• Chains of nodes:
Recap: Factorized Representation
• Reduction of complexity
• Joint probability of n binary variables requires us to represent values by brute force

\[ O(2^n) \text{ terms} \]

• The factorized form obtained from the graphical model only requires

\[ O(n \cdot 2^k) \text{ terms} \]

\[ k: \text{ maximum number of parents of a node.} \]

\[ \Rightarrow \text{It's the edges that are missing in the graph that are important!} \]

\[ \text{They encode the simplifying assumptions we make.} \]
Recap: Conditional Independence

- **X** is conditionally independent of **Y** given **V**
  
  - **Definition:**
    \[ X \perp Y \mid V \iff p(X,Y \mid V) = p(X \mid V)p(Y \mid V) \]
  
  - **Also:**
    \[ X \perp Y \mid V \iff p(X,Y \mid V) = p(X \mid V)p(Y \mid V) \]
  
  - **Special case:** Marginal Independence
    \[ X \perp Y \mid \emptyset \iff p(X,Y) = p(X)p(Y) \]
  
- Often, we are interested in conditional independence between sets of variables:
  \[ X \perp Y \mid V \iff \{ X \perp Y \mid V, \forall X \in X \text{ and } \forall Y \in Y \} \]

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-head 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 ⊥⊥ B | 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** \( \in \not V \)) 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 \( A \perp B \mid C \).
  - 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 $x_n$
- Define the messages recursively:
  - $\mu_a(x_n) = \sum_{x_{n-1}} v_{n-1,n}(x_{n-1}, x_n) \mu_a(x_{n-1})$
  - $\mu_b(x_n) = \sum_{x_{n+1}} v_{n,n+1}(x_n, x_{n+1}) \mu_b(x_{n+1})$
- Compute the normalization constant $Z$ at any node $x_n$,
  - $Z = \sum_{x_n} \mu_a(x_n) \mu_b(x_n)$

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
  - 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.
  - $p(x) \propto \prod_{v \in \text{mark}(x)} \mu_{v^\text{to} \rightarrow v}(x)$
- Computational effort
  - Total number of messages $= 2 \cdot \text{number of graph edges}$
Recap: Max-Sum Algorithm

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

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

Recap: Sum-Product from Leaves to Root

Message definitions:
\[
\begin{align*}
\mu_{f_j \rightarrow a}(x) &= \sum_{x_i} f_j(x_i) \prod_{m \in \text{ne}(f_j)} \mu_{x_m \rightarrow f_j}(x_m) \\
\mu_{x_m \rightarrow f_j}(x_m) &= \prod_{l \in \text{ne}(x_m) \setminus f_j} \mu_{x_l \rightarrow f_j}(x_l)
\end{align*}
\]

Recap: Sum-Product from Root to Leaves

Message definitions:
\[
\begin{align*}
\mu_{f_j \rightarrow a}(x) &= \sum_{x_i} f_j(x_i) \prod_{m \in \text{ne}(f_j)} \mu_{x_m \rightarrow f_j}(x_m) \\
\mu_{x_m \rightarrow f_j}(x_m) &= \prod_{l \in \text{ne}(x_m) \setminus f_j} \mu_{x_l \rightarrow f_j}(x_l)
\end{align*}
\]

Recap: Max-Sum Algorithm

- Initialization (leaf nodes)
  \( \mu_{f_j \rightarrow a}(x) = \ln f(x) \)
- Recursion
  - Messages
    \[ \mu_{f_j \rightarrow a}(x) = \max_{x_i} \left[ \ln f(x, x_1, \ldots, x_i) + \sum_{m \in \text{ne}(x_i) \setminus f_j} \mu_{x_m \rightarrow f_j}(x_m) \right] \]
  - For each node, keep a record of which values of the variables gave rise to the maximum state:
    \[ \phi(x) = \arg \max_{x_i} \left[ \ln f(x, x_1, \ldots, x_i) + \sum_{m \in \text{ne}(x_i) \setminus f_j} \mu_{x_m \rightarrow f_j}(x_m) \right] \]

Recap: Max-Sum Algorithm

- Termination (root node)
  - Score of maximal configuration
    \[ z^{\text{max}} = \max_{x} \left[ \sum_{e \in \text{ne}(x)} \mu_{e \rightarrow a}(x) \right] \]
  - Value of root node variable giving rise to that maximum
    \[ z^{\text{max}} = \arg \max_x \left[ \sum_{e \in \text{ne}(x)} \mu_{e \rightarrow a}(x) \right] \]
  - Back-track to get the remaining variable values
    \[ z^{\text{max}_{n-1}} = \phi(z^{\text{max}}_{n-1}) \]

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.
  - 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!

• 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

  "True" image content
  Noisy observations

• Two components
  - Observation model
    - How likely is it that node $x_i$ has label $L_i$ given observation $y_i$?
    - This relationship is usually learned from training data.
  - Neighborhood relations
    - Simplest case: 4-neighborhood
    - Serve as smoothing terms.
    - Discourage neighboring pixels to have different labels.
    - This can either be learned or be set to fixed “penalties”.

Recap: How to Set the Potentials?

• Unary potentials
  - E.g. color model, modeled with a Mixture of Gaussians
    \[
    \phi(x_i, y_i; \theta_k) = \log \sum_k \theta_k \mathcal{N}(x_i, y_i; \mu_k, \Sigma_k)
    \]

  ⇒ Learn color distributions for each label

• Pairwise potentials
  - Potts Model
    \[
    \psi(x_i, x_j; \theta_\delta) = \theta_\delta \delta(x_i \neq x_j)
    \]
    - 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”
    \[
    \psi(x_i, x_j, y_i, y_j; \theta_\delta) = \theta_\delta g_i(y_j) \delta(x_i \neq x_j)
    \]
    - Where
      \[
      g_i(y) = e^{-|y - \bar{y}|^2 / \beta}
      \beta = 2 \text{ arg max } |y_i - y_j|
      \]
      - Discourages label changes except in places where there is also a large change in the observations.
Recap: Graph Cuts for Binary Problems

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

Recap: Converting an MRF to an s-t Graph

Recap: s-t-Mincut Equivalent to Maxflow

Recap: α-Expansion Move

Any Questions?

So what can you do with all of this?
Mobile Object Detection & Tracking

[Ess, Leibe, Schindler, Van Gool, CVPR'08]

Learning Person-Object Interactions

B. Leibe
[T. Baumgartner, D. Mitzel, B. Leibe, CVPR'13]

Semantic Segmentation

image       ground truth       Baseline       RF (HOG)

[127]

3D Labeling Results - Living Room

play video

[Hermans, Flores, Leibe, submission to ICCV'13]

Semantic Scene Segmentation

[B. Leibe
[G. Floros, B. Leibe, CVPR'12]

Any More Questions?

Good luck for the exam!