Announcements

- Tentative Exam Dates
  - Planning with the following dates:
    - 1st date: Thursday, 13.08., afternoon
    - 2nd date: Friday, 11.09., afternoon
  - We tried to avoid overlaps with other Computer Science Master lectures as much as possible.
  - Exact slot durations and rooms will still be announced.
  - Does anybody still have conflicts with both exam dates?

Course Outline

- Fundamentals (2 weeks)
  - Bayes Decision Theory
  - Probability Density Estimation

- Discriminative Approaches (5 weeks)
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Randomized Trees, Forests & Ferns

- Generative Models (4 weeks)
  - Bayesian Networks
  - Markov Random Fields

Applications of SVMs: Text Classification

- Problem:
  - Classify a document in a number of categories

- Representation:
  - “Bag-of-words” approach
  - Histogram of word counts (on learned dictionary)
    - Very high-dimensional feature space (~10,000 dimensions)
    - Few irrelevant features
  - This was one of the first applications of SVMs
    - T. Joachims (1997)

Example Application: Text Classification

- Results:

<table>
<thead>
<tr>
<th></th>
<th>Bayes</th>
<th>Rocchio</th>
<th>CA</th>
<th>Lk</th>
<th>NN</th>
<th>SVM (poly)</th>
<th>SVM (RBF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>95.0</td>
<td>96.1</td>
<td>98.2</td>
<td>98.2</td>
<td>98.2</td>
<td>98.2</td>
<td>98.2</td>
</tr>
<tr>
<td>acq</td>
<td>91.5</td>
<td>92.1</td>
<td>90.3</td>
<td>90.0</td>
<td>90.0</td>
<td>90.0</td>
<td>90.0</td>
</tr>
<tr>
<td>money-fx</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
<td>62.6</td>
</tr>
<tr>
<td>crime</td>
<td>72.5</td>
<td>75.5</td>
<td>80.1</td>
<td>80.0</td>
<td>80.0</td>
<td>80.0</td>
<td>80.0</td>
</tr>
<tr>
<td>crime</td>
<td>81.0</td>
<td>81.5</td>
<td>83.5</td>
<td>83.0</td>
<td>83.0</td>
<td>83.0</td>
<td>83.0</td>
</tr>
<tr>
<td>trade</td>
<td>50.0</td>
<td>50.0</td>
<td>52.0</td>
<td>53.0</td>
<td>54.0</td>
<td>55.0</td>
<td>56.0</td>
</tr>
<tr>
<td>interest</td>
<td>50.0</td>
<td>50.0</td>
<td>52.0</td>
<td>53.0</td>
<td>54.0</td>
<td>55.0</td>
<td>56.0</td>
</tr>
<tr>
<td>ship</td>
<td>78.7</td>
<td>83.1</td>
<td>80.9</td>
<td>80.0</td>
<td>80.0</td>
<td>80.0</td>
<td>80.0</td>
</tr>
<tr>
<td>wheat</td>
<td>66.6</td>
<td>70.4</td>
<td>78.0</td>
<td>76.0</td>
<td>76.0</td>
<td>76.0</td>
<td>76.0</td>
</tr>
<tr>
<td>corn</td>
<td>47.3</td>
<td>62.2</td>
<td>67.7</td>
<td>77.9</td>
<td>80.0</td>
<td>80.0</td>
<td>80.0</td>
</tr>
<tr>
<td>microalg</td>
<td>73.6</td>
<td>79.9</td>
<td>79.4</td>
<td>82.3</td>
<td>84.2</td>
<td>86.1</td>
<td>86.1</td>
</tr>
</tbody>
</table>

This is also how you could implement a simple spam filter...
Example Application: OCR

- Handwritten digit recognition
  - US Postal Service Database
  - Standard benchmark task for many learning algorithms

Historical Importance

- USPS benchmark
  - 2.5% error: human performance

- Different learning algorithms
  - 16.2% error: Decision tree (C4.5)
  - 5.9% error: (best) 2-layer Neural Network
  - 5.1% error: LeNet 1 - (massively hand-tuned) 5-layer network

- Different SVMs
  - 4.0% error: Polynomial kernel (p=3, 274 support vectors)
  - 4.1% error: Gaussian kernel (p=0.3, 291 support vectors)

Example Application: OCR

- Results
  - Almost no overfitting with higher-degree kernels.

<table>
<thead>
<tr>
<th>degree of polynomial</th>
<th>dimensionality of feature space</th>
<th>support vectors</th>
<th>raw error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>256</td>
<td>282</td>
<td>8.9</td>
</tr>
<tr>
<td>2</td>
<td>≈ 33000</td>
<td>227</td>
<td>4.7</td>
</tr>
<tr>
<td>3</td>
<td>≈ 1 \times 10^6</td>
<td>274</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>≈ 1 \times 10^9</td>
<td>321</td>
<td>4.2</td>
</tr>
<tr>
<td>5</td>
<td>≈ 1 \times 10^{12}</td>
<td>374</td>
<td>4.3</td>
</tr>
<tr>
<td>6</td>
<td>≈ 1 \times 10^{14}</td>
<td>377</td>
<td>4.3</td>
</tr>
<tr>
<td>7</td>
<td>≈ 1 \times 10^{16}</td>
<td>422</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Example Application: Object Detection

- Sliding-window approach
  - E.g. histogram representation (HOG)
    - Map each grid cell in the input window to a histogram of gradient orientations.
    - Train a linear SVM using training set of pedestrian vs. non-pedestrian windows.

Example Application: Pedestrian Detection

N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Many Other Applications

- Lots of other applications in all fields of technology
  - OCR
  - Text classification
  - Computer vision
  - ...
  - High-energy physics
  - Monitoring of household appliances
  - Protein secondary structure prediction
  - Design on decision feedback equalizers (DFE) in telephony
So Far...

- We’ve seen already a variety of different classifiers
  - k-NN
  - Bayes classifiers
  - Linear discriminants
  - SVMs
- Each of them has their strengths and weaknesses...
  - Can we improve performance by combining them?

Topics of This Lecture

- Ensembles of Classifiers
- Constructing Ensembles
  - Cross-validation
  - Bagging
- Combining Classifiers
  - Stacking
  - Bayesian model averaging
  - Boosting
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
- Applications

Ensembles of Classifiers

- Intuition
  - Assume we have $K$ classifiers.
  - They are independent (i.e., their errors are uncorrelated).
  - Each of them has an error probability $p < 0.5$ on training data.
    - Why can we assume that $p$ won’t be larger than 0.5?
  - Then a simple majority vote of all classifiers should have a lower error than each individual classifier...

- Example
  - $K$ classifiers with error probability $p = 0.3$.
  - Probability that exactly $L$ classifiers make an error:
    $$p^L(1-p)^{K-L}$$
  - The probability that 11 or more classifiers make an error is 0.026.

Constructing Ensembles

- How do we get different classifiers?
  - Simplest case: train same classifier on different data.
  - But... where shall we get this additional data from?
    - Recall: training data is very expensive!
- Idea: Subsample the training data
  - Reuse the same training algorithm several times on different subsets of the training data.
- Well-suited for “unstable” learning algorithms
  - Unstable: small differences in training data can produce very different classifiers
    - E.g., Decision trees, neural networks, rule learning algorithms...
  - Stable learning algorithms
    - E.g., Nearest neighbor, linear regression, SVMs...
Constructing Ensembles

- Cross-Validation
  - Split the available data into \( N \) disjunct subsets.
  - In each run, train on \( N-1 \) subsets for training a classifier.
  - Estimate the generalization error on the held-out validation set.

- E.g. 5-fold cross-validation

```
train train train train test
train train test train train
train test train train train
test train train train train
```

Constructing Ensembles

- Bagging = “Bootstrap aggregation” (Breiman 1996)
  - In each run of the training algorithm, randomly select \( M \) samples from the full set of \( N \) training data points.
  - If \( M = N \), then on average, 63.2% of the training points will be represented. The rest are duplicates.

- Injecting randomness
  - Many (iterative) learning algorithms need a random initialization (e.g. \( k \)-means, EM)
  - Perform multiple runs of the learning algorithm with different random initializations.

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 cross-validation.
  - Interpret the prediction of the \( L \) classifiers as \( L \)-dimensional feature vector.
  - Learn “level-2” classifier based on the examples generated this way.

Recap: Model Combination

- E.g. Mixture of Gaussians
  - Several components are combined probabilistically.
  - Interpretation: different data points can be generated by different components.
  - We model the uncertainty which mixture component is responsible for generating the corresponding data point:
    \[
p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)
    \]
  - For i.i.d. data, we write the marginal probability of a data set \( X = \{x_1, \ldots, x_N\} \) in the form:
    \[
p(X) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)
    \]
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
    \[
    p(X) = \sum_{h=1}^{H} p(X|h)p(h)
    \]

- **Interpretation**
  - Just one model is responsible for generating the entire data set.
  - The probability distribution over \( h \) just reflects our uncertainty which model that is.
  - As the size of the data set increases, this uncertainty reduces, and \( p(X|h) \) becomes focused on just one of the models.

Discussion: Ensembles of Classifiers

- **Model Combination**
  - Different data points generated by different model components.
  - Uncertainty is about which component created which data point.

\[
E_{\text{COM}} = \sum_{r=1}^{N} E_{\text{COM}}(x_{r}) = \sum_{r=1}^{N} \sum_{w_{m}} p(x_{r}, z_{w})
\]

- **Bayesian Model Averaging**
  - The whole data set is generated by a single model.
  - Uncertainty is about which model was responsible.

\[
p(X) = \sum_{z} p(X, z)
\]

Note the Different Interpretations!

- **Model Combination**
  - Different data points generated by different model components.
  - Uncertainty is about which component created which data point.

\[
E_{\text{COM}} = \sum_{r=1}^{N} E_{\text{COM}}(x_{r}) = \sum_{r=1}^{N} \sum_{w_{m}} p(x_{r}, z_{w})
\]

- **Bayesian Model Averaging**
  - The whole data set is generated by a single model.
  - Uncertainty is about which model was responsible.

\[
p(X) = \sum_{z} p(X, z)
\]
**AdaBoost**

- **Main idea**
  - Instead of resampling, reweight misclassified training examples.
  - Increase the chance of being selected in a sampled training 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:

$$H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)$$

---

**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.
AdaBoost - Algorithm

1. Initialization: Set \( w_n^{(1)} = \frac{1}{N} \) 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
      \[
      J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) \quad \text{if } d \text{ is true}
      \]
      \[
      b) \text{Estimate the weighted error of this classifier on } X:
      \]
      \[
      e_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) \quad \sum_{n=1}^{N} w_n^{(m)}
      \]
      \[
      c) \text{Calculate a weighting coefficient for } h_m(x); \quad \alpha_m = ?
      \]
      \[
      d) \text{Update the weighting coefficients: } w_n^{(m+1)} = ?
      \]

How should we do this exactly?

AdaBoost - Minimizing Exponential Error

- Exponential error function
  \[
  E = \sum_{n=1}^{N} \exp \left\{ -t_n f_m(x_n) \right\}
  \]
  where \( f_m(x) \) is a classifier defined as a linear combination of base classifiers \( h_l(x) \):
  \[
  f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l h_l(x)
  \]

- Goal
  Minimize \( E \) with respect to both the weighting coefficients \( \alpha_l \)
  and the parameters of the base classifiers \( h_l(x) \).

AdaBoost - Historical Development

- Originally motivated by Statistical Learning Theory
  - AdaBoost was introduced in 1996 by Freund & Schapire.
  - It was empirically observed that AdaBoost often tends not to overfit. (Breiman 96, Cortes & Drucker 97, etc.)
  - As a result, the margin theory (Schapire et al. 98) developed, which is based on loose generalization bounds.
    \[
    \text{Note: margin for boosting is not the same as margin for SVM.}
    \]
    \[
    \text{A bit like retrofitting the theory...}
    \]
    \[
    \text{However, those bounds are too loose to be of practical value.}
    \]
- Different explanation (Friedman, Hastie, Tibshirani, 2000)
  Interpretation as sequential minimization of an exponential error function ("Forward Stagewise Additive Modeling").
  Explains why boosting works well.
  Improvements possible by altering the error function.

AdaBoost - Minimizing Exponential Error

- Sequential Minimization
  - Suppose that the base classifiers \( h_1(x), \ldots, h_m(x) \) and their coefficients \( \alpha_1, \ldots, \alpha_m \) are fixed.
  \[
  \Rightarrow \text{Only minimize with respect to } \alpha_0 \text{ and } h_0(x).
  \]
  \[
  E = \sum_{n=1}^{N} \exp \left\{ -t_n f_m(x_n) \right\} \quad \text{with } f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l h_l(x)
  \]
  \[
  = \sum_{n=1}^{N} \exp \left\{ -t_n f_{m-1}(x_n) - \frac{1}{2} \alpha_0 h_0(x_n) \right\}
  \]
  \[
  = \sum_{n=1}^{N} w_n^{(m)} \exp \left\{ \frac{1}{2} \alpha_0 h_0(x_n) \right\}
  \]

AdaBoost - Minimizing Exponential Error

- Observation:
  Correctly classified points: \( t_n h_n(x_n) = +1 \) \( \Rightarrow \text{collect in } F_m \)
  Misclassified points: \( t_n h_n(x_n) = -1 \) \( \Rightarrow \text{collect in } F_m \)
  Rewrite the error function as
  \[
  E = e^{-\alpha_0/2} \sum_{n \in F_m} w_n^{(m)} + e^{\alpha_0/2} \sum_{n \in F_m} w_n^{(m)}
  \]
  \[
  = e^{\alpha_0/2} \sum_{n \in F_m} w_n^{(m)} I(h_m(x_n) \neq t_n)
  \]

AdaBoost - Minimizing Exponential Error

- Observation:
  Correctly classified points: \( t_n h_n(x_n) = +1 \) \( \Rightarrow \text{collect in } F_m \)
  Misclassified points: \( t_n h_n(x_n) = -1 \) \( \Rightarrow \text{collect in } F_m \)
  Rewrite the error function as
  \[
  E = e^{-\alpha_0/2} \sum_{n \in F_m} w_n^{(m)} + e^{\alpha_0/2} \sum_{n \in F_m} w_n^{(m)}
  \]
  \[
  = e^{-\alpha_0/2} \sum_{n \in F_m} w_n^{(m)} I(h_m(x_n) \neq t_n) + e^{-\alpha_0/2} \sum_{n \in F_m} w_n^{(m)}
  \]
AdaBoost - Minimizing Exponential Error

• Minimize with respect to $h_m(x)$:
  $$\frac{\partial E}{\partial h_m(x_n)} \downarrow 0$$

$$E = \left( e^{\alpha_n/2} - e^{-\alpha_n/2} \right) \left( \sum_{n=1}^{N} u^{(m)}_n I(h_m(x_n) \neq t_n) \right) + e^{-\alpha_n/2} \sum_{n=1}^{N} u^{(m)}_n$$

$$= \text{const.}$$

⇒ This is equivalent to minimizing

$$J_m = \sum_{n=1}^{N} u^{(m)}_n I(h_m(x_n) \neq t_n)$$

(our weighted error function from step 2a) of the algorithm

⇒ We’re on the right track. Let’s continue...

AdaBoost - Final Algorithm

1. Initialization: Set $w^{(1)}_n = \frac{1}{N}$ 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

$$J_m = \sum_{n=1}^{N} u^{(m)}_n I(h_m(x_n) \neq t_n)$$

b) Estimate the weighted error of this classifier on $X$:

$$\epsilon_m = \frac{\sum_{n=1}^{N} u^{(m)}_n I(h_m(x_n) \neq t_n)}{\sum_{n=1}^{N} u^{(m)}_n}$$

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:

$$w^{(m+1)}_n = w^{(m)}_n \exp \left( \alpha_m I(h_m(x_n) \neq t_n) \right)$$

### Recap: Error Functions

- $t_n \subset \{1, 1\}$
- Ideal misclassification error

Not differentiable!

- **Ideal misclassification error function (black)**
  - This is what we want to approximate.
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.
  - We cannot minimize it by gradient descent.
Recap: Error Functions

- Squared error used in Least-Squares Classification
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes “too correct” data points
    - Generally does not lead to good classifiers.

- “Hinge error” used in SVMs
  - Zero error for points outside the margin ($z_n > 1$) ⇒ sparsity
  - Linear penalty for misclassified points ($z_n < 1$) ⇒ robustness
  - Not differentiable around $z_n = 1$ ⇒ Cannot be optimized directly

Discussion: AdaBoost Error Function

- Exponential error used in AdaBoost
  - Continuous approximation to ideal misclassification function.
  - Sequential minimization leads to simple AdaBoost scheme.
  - Properties?

- Exponential error used in AdaBoost
  - No penalty for too correct data points, fast convergence.
  - Disadvantage: exponential penalty for large negative values!
    - Less robust to outliers or misclassified data points!

Discussion: Other Possible Error Functions

- “Cross-entropy error” used in Logistic Regression
  - Similar to exponential error for $z > 0$.
  - Only grows linearly with large negative values of $z$.
  - Make AdaBoost more robust by switching to this error function.
    - “GentleBoost”

Summary: AdaBoost

- Properties
  - Simple combination of multiple classifiers.
  - Easy to implement.
  - Can be used with many different types of classifiers.
    - None of them needs to be too good on its own.
    - In fact, they only have to be slightly better than chance.
    - Commonly used in many areas.
    - Empirically good generalization capabilities.

- Limitations
  - Original AdaBoost sensitive to misclassified training data points.
    - Because of exponential error function.
    - Improvement by GentleBoost
  - Single-class classifier
  - Multiclass extensions available
Topics of This Lecture

• Ensembles of Classifiers
• Constructing Ensembles
  - Cross-validation
  - Bagging
• Combining Classifiers
  - Stacking
  - Bayesian model averaging
  - Boosting
• AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
• Applications

Example Application: Face Detection

• Frontal faces are a good example of a class where global appearance models + a sliding window detection approach fit well:
  - Regular 2D structure
  - Center of face almost shaped like a "patch"/window

• Now we’ll take AdaBoost and see how the Viola-Jones face detector works

Feature extraction

“Rectangular” filters

Feature output is difference between adjacent regions

Value at (x,y) is sum of pixels above and to the left of (x,y)

Efficiently computable with integral image: any sum can be computed in constant time

Avoid scaling images → scale features directly for same cost

Large Library of Filters

Considering all possible filter parameters: position, scale, and type:

180,000+ possible features associated with each 24 x 24 window

Use AdaBoost both to select the informative features and to form the classifier

AdaBoost for Feature+Classifier Selection

• Want to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of weighted error.

Resulting weak classifier:

\[
    h(x) = \begin{cases} 
        +1 & \text{if } f_i(x) > 0_i \\
        -1 & \text{otherwise} 
    \end{cases}
\]

For next round, reweight the examples according to errors, choose another filter/threshold combo.

AdaBoost for Efficient Feature Selection

• Image features = weak classifiers
• For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter (min error)
    - Sorted list can be quickly scanned for the optimal threshold
  - Select best filter/threshold combination
  - Weight on this features is a simple function of error rate
  - Reweight examples

Viola-Jones Face Detector: Results

References and Further Reading

- More information on Classifier Combination and Boosting can be found in Chapters 14.1-14.3 of Bishop’s book.

Christopher M. Bishop
Pattern Recognition and Machine Learning
Springer, 2006

- A more in-depth discussion of the statistical interpretation of AdaBoost is available in the following paper: