Machine Learning – Lecture 10

Model Combination & Boosting

06.06.2016

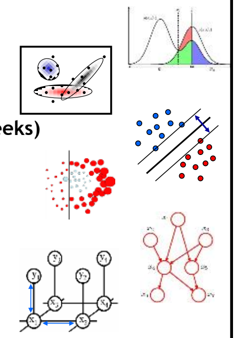
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

Many slides adapted from B. Schiele

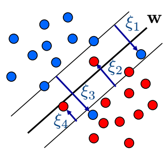
Course Outline

- 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



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.
  - $\xi_n = |t_n - y(\mathbf{x}_n)|$ for all other points.



Point on decision boundary: $\xi_n = 1$

Misclassified point: $\xi_n > 1$

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

$\Rightarrow$ They are jointly optimized together with $\mathbf{w}$.

Recap: SVM - New Dual Formulation

- New SVM Dual: Maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n)$$

under the conditions

$$0 \cdot a_n \cdot C$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

This is all that changed!

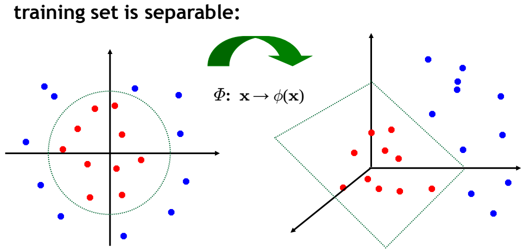
- This is again a quadratic programming problem

$\Rightarrow$ Solve as before...

Slide adapted from Bernt Schiele

Recap: Nonlinear SVMs

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



Slide credit: Raymond Mooney

Recap: The Kernel Trick

- Important observation
  - $\phi(\mathbf{x})$ only appears in the form of dot products $\phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{y})$:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) + b$$
$$= \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}) + b$$

  - Define a so-called kernel function $k(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{y})$.
  - Now, in place of the dot product, use the kernel instead:

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}) + b$$

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

Recap: Nonlinear SVM - Dual Formulation

- SVM Dual: Maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_m, \mathbf{x}_n)$$

under the conditions

$$0 \cdot\ a_n \cdot\ C$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

- Classify new data points using

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}) + b$$

SVM - Analysis

- Traditional soft-margin formulation

$$\min_{\mathbf{w}\in\mathbb{R}^D,\, \xi_n \in \mathbb{R}^+} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

"Maximize the margin"

subject to the constraints

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n$$

"Most points should be on the correct side of the margin"

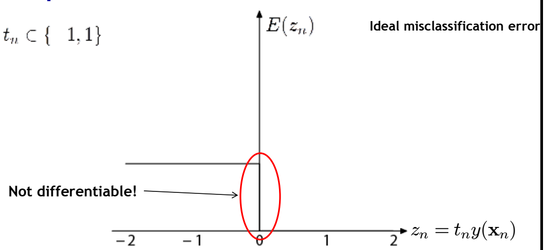
- Different way of looking at it
  - We can reformulate the constraints into the objective function.

$$\min_{\mathbf{w}\in\mathbb{R}^D} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{L_2 \text{ regularizer}} + C\underbrace{\sum_{n=1}^{N}[1 - t_n y(\mathbf{x}_n)]_+}_{\text{"Hinge loss"}}$$

where $[x]_+ := \max\{0, x\}$.

Recap: Error Functions

$t_n \subset \{\ 1, 1\}$

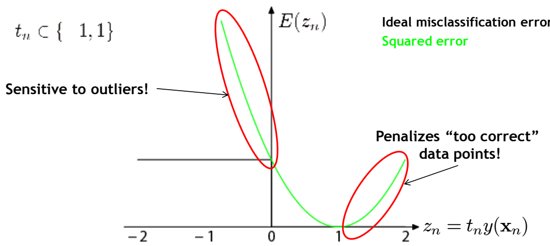


- Ideal misclassification error function (black)
  - This is what we want to approximate,
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.

  ⇒ We cannot minimize it by gradient descent.

Recap: Error Functions

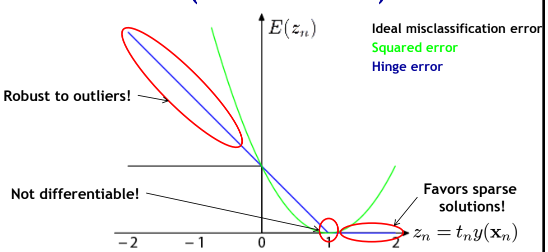
$t_n \subset \{\ 1, 1\}$



- 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.

Error Functions (Loss Functions)



- "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.

SVM - Discussion

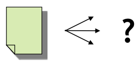
- SVM optimization function

$$\min_{\mathbf{w}\in\mathbb{R}^D} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{L_2 \text{ regularizer}} + C\underbrace{\sum_{n=1}^{N}[1 - t_n y(\mathbf{x}_n)]_+}_{\text{Hinge loss}}$$

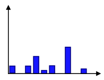
- Hinge loss enforces sparsity
  - Only a subset of training data points actually influences the decision boundary.
  - This is different from sparsity obtained through the regularizer! There, only a subset of input dimensions are used.
  - Unconstrained optimization, but non-differentiable function.
  - Solve, e.g. by *subgradient descent*
  - Currently most efficient: *stochastic gradient descent*

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)

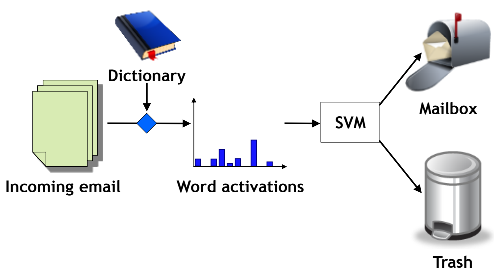
Example Application: Text Classification

- **Results:**

	Bayes	Rocchio	C4.5	k-NN	SVM (poly) degree $d =$ 1	2	3	4	5	SVM (rbf) width $\gamma =$ 0.6	0.8	1.0	1.2
earn	95.9	96.1	96.1	97.3	98.2	98.4	**98.5**	98.4	98.3	**98.5**	98.5	98.4	98.3
acq	91.5	92.1	85.3	92.0	92.6	94.6	**95.2**	95.2	95.3	95.0	95.3	95.3	**95.4**
money-fx	62.9	67.6	69.4	78.2	66.9	72.5	75.4	74.9	**76.2**	74.0	75.4	**76.3**	75.9
grain	72.5	79.5	89.1	82.2	91.3	93.1	**92.4**	91.3	89.9	**93.1**	91.9	91.9	90.6
crude	81.0	81.5	75.5	85.7	86.0	87.3	88.6	**88.9**	87.8	**88.9**	89.0	88.9	88.2
trade	50.0	77.4	59.2	77.4	69.2	75.5	76.6	77.3	**77.1**	76.9	78.0	**77.8**	76.8
interest	58.0	72.5	49.1	74.0	69.8	63.3	67.9	73.1	**76.2**	74.4	75.0	**76.2**	76.1
ship	78.7	83.1	80.9	79.2	82.0	85.4	86.0	**86.5**	86.0	**85.4**	86.5	87.6	87.1
wheat	60.6	79.4	85.5	76.6	83.1	84.5	85.2	**85.9**	83.8	**85.2**	85.9	85.9	85.9
corn	47.3	62.2	87.7	77.9	86.0	86.5	85.3	**85.7**	83.9	**85.1**	85.7	85.7	84.5
microavg.	**72.0**	**79.9**	**79.4**	**82.3**	84.2	85.1	85.9	86.2	85.9	86.4	86.5	86.3	86.2
					combined: **86.0**					combined: **86.4**			

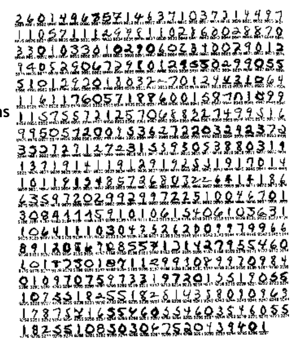
Example Application: Text Classification

- **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 (σ=0.3, 291 support vectors)

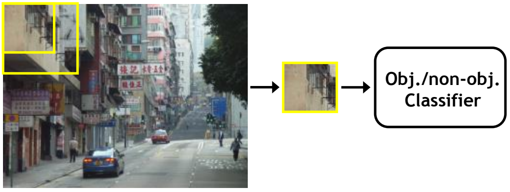
Example Application: OCR

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

degree of polynomial	dimensionality of feature space	support vectors	raw error
1	256	282	8.9
2	≈ 33000	227	4.7
3	$\approx 1 \times 10^6$	274	4.0
4	$\approx 1 \times 10^9$	321	4.2
5	$\approx 1 \times 10^{12}$	374	4.3
6	$\approx 1 \times 10^{14}$	377	4.5
7	$\approx 1 \times 10^{16}$	422	4.5

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.

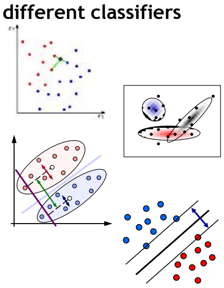
[Dalal & Triggs, CVPR 2005]

Example Application: Pedestrian Detection

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

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…

Slide adapted from Bernt Schiele

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

Methods for obtaining a set of classifiers

Methods for combining different classifiers

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	train	test	train
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 mutliple runs of the learning algorithm with different random initializations.

Slide adapted from Bernt Schiele

Topics of This Lecture

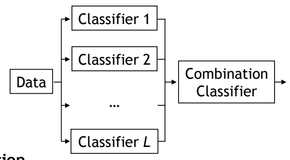
- Ensembles of Classifiers
- Constructing Ensembles
  - Cross-validation
  - Bagging
- Combining Classifiers
  - Stacking
  - Bayesian Model Averaging
  - Boosting
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
- Applications

Methods for obtaining a set of classifiers

Methods for combining different classifiers

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.

Slide credit: Bernt Schiele

Stacking

- Why can this be useful?
  - Simplicity
    - We may already have several existing classifiers available.
    - ⇒ No need to retrain those, they can just be combined with the rest.
  - Correlation between classifiers
    - The combination classifier can learn the correlation.
    - ⇒ Better results than simple Naïve Bayes combination.
  - Feature combination
    - E.g. combine information from different sensors or sources (vision, audio, acceleration, temperature, radar, etc.).
    - We can get good training data for each sensor individually, but data from all sensors together is rare.
    - ⇒ Train each of the L classifiers on its own input data. Only combination classifier needs to be trained on combined input.

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:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

  - For i.i.d. data, we write the marginal probability of a data set $\mathbf{X} = \{\mathbf{x}_1,\ldots,\mathbf{x}_N\}$ in the form:

$$p(\mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{x}_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

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

$$p(\mathbf{X}) = \sum_{h=1}^{H} p(\mathbf{X}|h)p(h)$$

- 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 $p(\mathbf{X}|h)$ becomes focused on just one of the models.

Note the Different Interpretations!

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

  ⇒ One latent variable $\mathbf{z}_n$ for each data point:

$$p(\mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{x}_n) = \prod_{n=1}^{N} \sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n)$$

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

  ⇒ One latent variable $\mathbf{z}$ for the entire data set:

$$p(\mathbf{X}) = \sum_{\mathbf{z}} p(\mathbf{X}, \mathbf{z})$$

Model Averaging: Expected Error

- Combine M predictors $y_m(\mathbf{x})$ for target output $h(\mathbf{x})$.
  - E.g. each trained on a different bootstrap data set by bagging.
  - The committee prediction is given by

$$y_{COM}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} y_m(\mathbf{x})$$

  - The output can be written as the true value plus some error.

$$y(\mathbf{x}) = h(\mathbf{x}) + \epsilon(\mathbf{x})$$

  - Thus, the expected sum-of-squares error takes the form

$$\mathbb{E}_{\mathbf{x}} = \left[\{y_m(\mathbf{x}) - h(\mathbf{x})\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

Model Averaging: Expected Error

- Average error of individual models

$$\mathbb{E}_{AV} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

- Average error of committee

$$\mathbb{E}_{COM} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x})\right\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$

- Assumptions
  - Errors have zero mean: $\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\right] = 0$
  - Errors are uncorrelated: $\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\epsilon_j(\mathbf{x})\right] = 0$
- Then: $\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$

Isn't this spectacular?

Model Averaging: Expected Error

- Average error of committee

$$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$$

  - 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!
  - Spectacular indeed...
  - This sounds almost too good to be true...
- And it is... Can you see where the problem is?
  - Unfortunately, this result depends on the assumption that the errors are all uncorrelated.
  - In practice, they will typically be highly correlated.
  - Still, it can be shown that

$$\mathbb{E}_{COM} \cdot \mathbb{E}_{AV}$$

Discussion: Ensembles of Classifiers

- Set of simple methods for improving classification
  - Often effective in practice.
- Apparent contradiction
  - We have stressed before that a classifier should be trained on samples from the distribution on which it will be tested.
  - Resampling seems to violate this recommendation.
  - *Why can a classifier trained on a weighted data distribution do better than one trained on the i.i.d. sample?*
- Explanation
  - We do not attempt to model the full category distribution here.
  - Instead, try to find the decision boundary more directly.
  - Also, increasing number of component classifiers broadens the class of implementable decision functions.

Topics of This Lecture

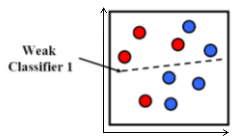
- Ensembles of Classifiers
- Constructing Ensembles
  - Cross-validation
  - Bagging
- Combining Classifiers
  - Stacking
  - Bayesian model averaging
  - Boosting
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions
- Applications

AdaBoost – "Adaptive Boosting"

- Main idea [Freund & Schapire, 1996]
  - 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
  - $h_m(\mathbf{x})$: "weak" or base classifier
    - Condition: <50% training error over any distribution
  - $H(\mathbf{x})$: "strong" or final classifier
- AdaBoost:
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:

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

AdaBoost: Intuition



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

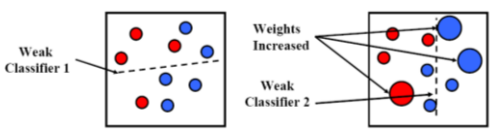
Each weak classifier splits the training examples with at least 50% accuracy.

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

Slide credit: Kristen Grauman

Figure adapted from Freund & Schapire

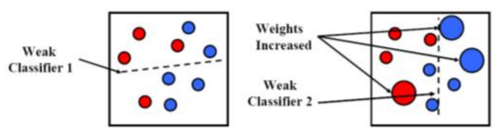
AdaBoost: Intuition

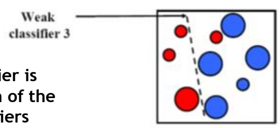


Slide credit: Kristen Grauman

Figure adapted from Freund & Schapire

AdaBoost: Intuition





Final classifier is combination of the weak classifiers

Slide credit: Kristen Grauman

Figure adapted from Freund & Schapire

AdaBoost – Formalization

- 2-class classification problem
  - Given: training set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with target values $\mathbf{T} = \{t_1, \ldots, t_N\}$, $t_n \in \{-1,1\}$.
  - Associated weights $\mathbf{W}=\{w_1, \ldots, w_N\}$ for each training point.
- Basic steps
  - In each iteration, AdaBoost trains a new weak classifier $h_m(\mathbf{x})$ based on the current weighting coefficients $\mathbf{W}^{(m)}$.
  - We then adapt the weighting coefficients for each point
    - Increase w_n if $\mathbf{x}_n$ was misclassified by $h_m(\mathbf{x})$.
    - Decrease w_n if $\mathbf{x}_n$ was classified correctly by $h_m(\mathbf{x})$.
  - Make predictions using the final combined model

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

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 $h_m(\mathbf{x})$ using the current weighting coefficients $\mathbf{W}^{(m)}$ by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n) \qquad I(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{else} \end{cases}$$

   b) Estimate the weighted error of this classifier on $\mathbf{X}$:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

   c) Calculate a weighting coefficient for $h_m(\mathbf{x})$:

$$\alpha_m = \ ?$$

   d) Update the weighting coefficients:

$$w_n^{(m+1)} = \ ?$$

How should we do this exactly?

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.
    - Note: margin for boosting is *not* the same as margin for SVM.
    - A bit like retrofitting the theory…
  - 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

- Exponential error function

$$E = \sum_{n=1}^{N} \exp\left\{-t_n f_m(\mathbf{x}_n)\right\}$$

  - where $f_m(\mathbf{x})$ is a classifier defined as a linear combination of base classifiers $h_l(\mathbf{x})$:

$$f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l h_l(\mathbf{x})$$

- Goal
  - Minimize E with respect to both the weighting coefficients α_l and the parameters of the base classifiers $h_l(\mathbf{x})$.

AdaBoost – Minimizing Exponential Error

- Sequential Minimization
  - Suppose that the base classifiers $h_1(\mathbf{x}),\ldots, h_{m-1}(\mathbf{x})$ and their coefficients $\alpha_1,\ldots,\alpha_{m-1}$ are fixed.

  ⇒ Only minimize with respect to α_m and $h_m(\mathbf{x})$.

$$\begin{aligned} E &= \sum_{n=1}^{N} \exp\left\{-t_n f_m(\mathbf{x}_n)\right\} \quad \text{with} \quad f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{m}\alpha_l h_l(\mathbf{x}) \\ &= \sum_{n=1}^{N} \exp\left\{\underbrace{-t_n f_{m-1}(\mathbf{x}_n)}_{= \textit{const.}} - \frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\} \\ &= \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\} \end{aligned}$$

AdaBoost – Minimizing Exponential Error

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\}$$

- Observation:
  - Correctly classified points: $t_n h_m(\mathbf{x}_n) = +1$ ⇒ collect in $\mathcal{T}_m$
  - Misclassified points: $t_n h_m(\mathbf{x}_n) = -1$ ⇒ collect in $\mathcal{F}_m$
- Rewrite the error function as

$$E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{F}_m} w_n^{(m)}$$

$$= \left(e^{\alpha_m/2} \qquad \right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n)$$

AdaBoost - Minimizing Exponential Error

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\}$$

- Observation:
  - Correctly classified points: $t_n h_m(\mathbf{x}_n) = +1$ $\Rightarrow$ collect in $\mathcal{T}_m$
  - Misclassified points: $t_n h_m(\mathbf{x}_n) = -1$ $\Rightarrow$ collect in $\mathcal{F}_m$
- Rewrite the error function as

$$E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{F}_m} w_n^{(m)}$$

$$= \left(e^{\alpha_m/2} - e^{-\alpha_m/2}\right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$

AdaBoost - Minimizing Exponential Error

- Minimize with respect to $h_m(\mathbf{x})$: $\frac{\partial E}{\partial h_m(\mathbf{x}_n)} \stackrel{!}{=} 0$

$$E = \underbrace{\left(e^{\alpha_m/2} - e^{-\alpha_m/2}\right)}_{=\ const.} \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) + \underbrace{e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}}_{=\ const.}$$

$\Rightarrow$ This is equivalent to minimizing

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)$$

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

$\Rightarrow$ *We're on the right track. Let's continue…*

AdaBoost - Minimizing Exponential Error

- Minimize with respect to α_m: $\frac{\partial E}{\partial \alpha_m} \stackrel{!}{=} 0$

$$E = \left(e^{\alpha_m/2} - e^{-\alpha_m/2}\right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$

$$\left(\frac{\cancel{1}}{\cancel{2}} e^{\alpha_m/2} + \frac{\cancel{1}}{\cancel{2}} e^{-\alpha_m/2}\right) \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) \stackrel{!}{=} \frac{\cancel{1}}{\cancel{2}} e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}$$

weighted error

$$\epsilon_m := \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} = \frac{e^{-\alpha_m/2}}{e^{\alpha_m/2} + e^{-\alpha_m/2}}$$

$$\epsilon_m = \frac{1}{e^{\alpha_m} + 1}$$

$\Rightarrow$ *Update for the α coefficients:* $\alpha_m = \ln\left\{\frac{1 - \epsilon_m}{\epsilon_m}\right\}$

AdaBoost - Minimizing Exponential Error

- Remaining step: update the weights
  - Recall that

$$E = \sum_{n=1}^{N} \underbrace{w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\}}_{\text{This becomes } w_n^{(m+1)} \text{ in the next iteration.}}$$

  - Therefore

$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m h_m(\mathbf{x}_n)\right\}$$
$$= \ldots$$
$$= w_n^{(m)} \exp\left\{\alpha_m I(h_m(\mathbf{x}_n) \neq t_n)\right\}$$

$\Rightarrow$ *Update for the weight coefficients.*

AdaBoost - Final 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 $h_m(\mathbf{x})$ using the current weighting coefficients $\mathbf{W}^{(m)}$ by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)$$

   b) Estimate the weighted error of this classifier on $\mathbf{X}$:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

   c) Calculate a weighting coefficient for $h_m(\mathbf{x})$:

$$\alpha_m = \ln\left\{\frac{1 - \epsilon_m}{\epsilon_m}\right\}$$

   d) Update the weighting coefficients:

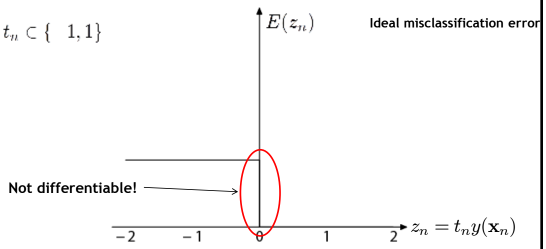
$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{\alpha_m I(h_m(\mathbf{x}_n) \neq t_n)\right\}$$

AdaBoost - Analysis

- Result of this derivation
  - We now know that AdaBoost minimizes an exponential error function in a sequential fashion.
  - This allows us to analyze AdaBoost's behavior in more detail.
  - In particular, we can see how robust it is to outlier data points.

Recap: Error Functions

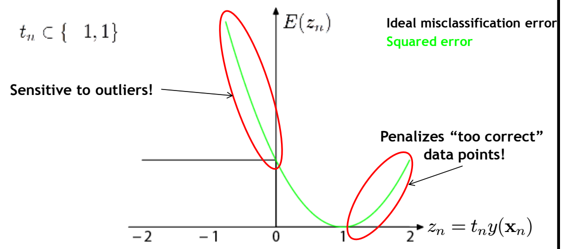
$t_n \subset \{\ 1, 1\}$



- 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.

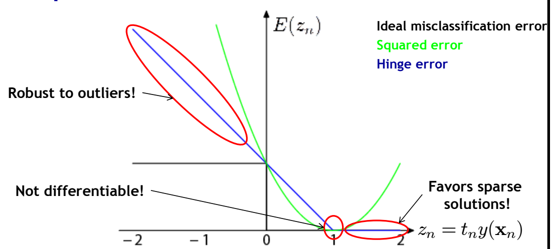
Recap: Error Functions

$t_n \subset \{\ 1, 1\}$



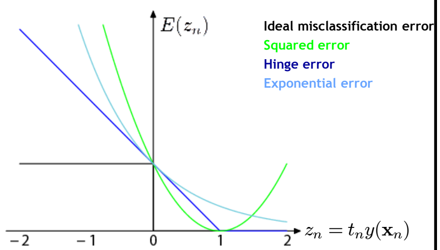
- Squared error used in Least-Squares Classification
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes “too correct” data points
  - ⇒ Generally does not lead to good classifiers.

Recap: Error Functions



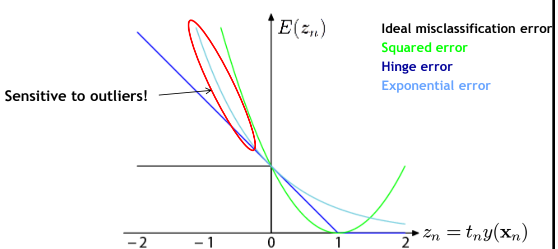
- “Hinge error” used in SVMs
  - Zero error for points outside the margin ($z_n > 1$) ⇒ sparsity
  - Linear penalty for misclassified points ($z_n < 1$) ⇒ robustness
  - Not differentiable around $z_n = 1$ ⇒ Cannot be optimized directly.

Discussion: AdaBoost Error Function



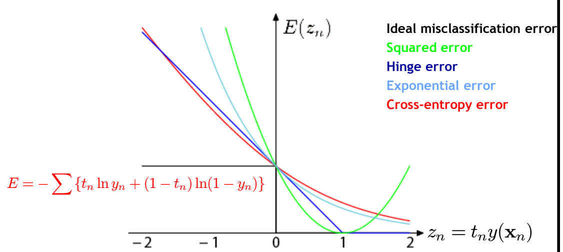
- Exponential error used in AdaBoost
  - Continuous approximation to ideal misclassification function.
  - Sequential minimization leads to simple AdaBoost scheme.
  - Properties?

Discussion: AdaBoost Error Function



- 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”

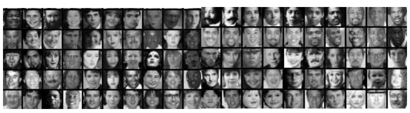
Summary: AdaBoost

- 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

Example Application: Face Detection

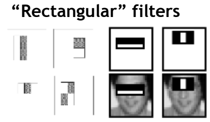
- Frontal faces are a good example of a class where global appearance models + a sliding window detection approach fit well:
  - Regular 2D structure
  - Center of face almost shaped like a "patch"/window



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

Feature extraction

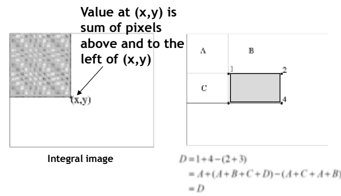
"Rectangular" filters



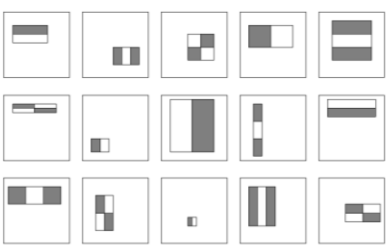
Feature output is difference between adjacent regions

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



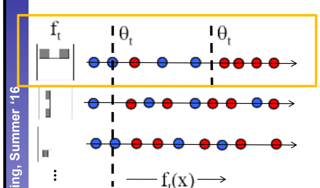
Considering all possible filter parameters: position, scale, and type:

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.



Outputs of a possible rectangle feature on faces and non-faces.

Resulting weak classifier:

$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$

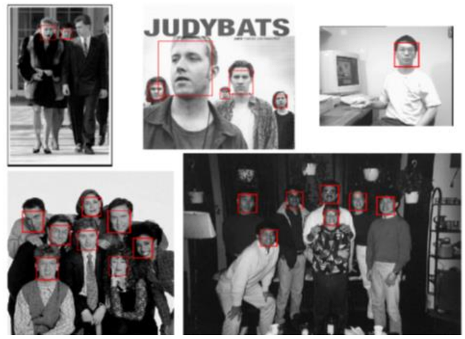
For next round, reweight the examples according to errors, choose another filter/threshold combo.

AdaBoost for Efficient Feature Selection

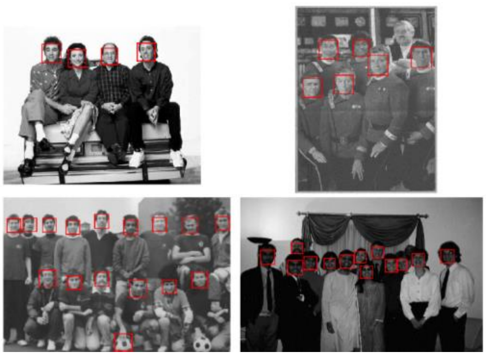
- Image features = weak classifiers
- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter (min error)
    - Sorted list can be quickly scanned for the optimal threshold
  - Select best filter/threshold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

P. Viola, M. Jones, Robust Real-Time Face Detection, IJCV, Vol. 57(2), 2004.
(first version appeared at CVPR 2001)

Viola-Jones Face Detector: Results



Viola-Jones Face Detector: Results

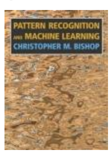


Viola-Jones Face Detector: Results

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:
  - J. Friedman, T. Hastie, R. Tibshirani, Additive Logistic Regression: a Statistical View of Boosting, *The Annals of Statistics*, Vol. 38(2), pages 337-374, 2000.