6.891 Machine learning and neural networks

Tommi Jaakkola
MIT AI LAB
tommi@ai.mit.edu

LECTURE 8 (October 3, 2000)

Topics

- Combining multiple methods
  - Sequential (forward) fitting
  - Bagging
  - Boosting
- Support vector machines
  - optimal hyperplane
  - kernel function

Combination of multiple methods

- Why would we want to generate and combine multiple methods rather than use a single more complex method?
  - decompositon into simpler subproblems
  - modular estimation
  - multiple "weak" methods can be combined into a "strong" method
- We have to
  - estimate the component methods in a modular way
  - find an appropriate combination rule
  - worry about generalization

Combination of regression methods

- We want to combine multiple "weak" regression methods into a single "strong" method
- Basic (forward fitting) idea: given a regression training set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

  Step 1: $\hat{f}_1(\mathbf{x}) \leftarrow \text{argmin}_f \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2/2$

  Step 2: $\hat{f}_2(\mathbf{x}) \leftarrow \text{argmin}_f \sum_{i=1}^n (y_i - \hat{f}_1(\mathbf{x}_i) - f(\mathbf{x}_i))^2/2$

  Step 3: ...

  The resulting combined regression method

$$\hat{f}_{comb}(\mathbf{x}) = \hat{f}_1(\mathbf{x}) + \ldots + \hat{f}_m(\mathbf{x}) \qquad (1)$$

  has much lower (training) error.

Combination of classifiers

- Suppose we have already generated a number of binary ± 1 classifiers $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$.

  We assume for simplicity that each classifier is equally likely to be "correct", i.e., we have no reason to prefer one over another.
- Under these circumstances, we can collect them together into a single classification method by *voting*: we decide the label according to the *majority* of the component methods

$$h(\mathbf{x}) = \text{sign}\left(\frac{1}{m} \sum_{k=1}^{m} h_k(\mathbf{x}) \right) \tag{2}$$

- If all classifiers agree on a specific example $\mathbf{x}$, then

$$\left|\frac{1}{m} \sum_{k=1}^{m} h_k(\mathbf{x})\right| = 1 \tag{3}$$

  Larger *majority* means a larger *classification margin*.

Bagging

- How do we generate the component classifiers?
- Suppose we have a training set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
  1. We *resample* (with replacement) a smaller training set of $n' < n$ examples
  2. We train a new component classifier $h_k(\mathbf{x})$ based on the smaller training set
- The combined classifier is then obtained by voting

$$h(\mathbf{x}) = \text{sign}\left(\frac{1}{m} \sum_{k=1}^{m} h_k(\mathbf{x}) \right) \tag{4}$$

+ This averaging of component classifiers can reduce *variance* arising from the random choice of the original training set

– Does not reduce *bias* (how far off we are from the optimal classifier)

Reweighting training examples

- To reduce *bias* the component classifiers should concentrate more on training examples that are difficult to classify correctly
- We can tune the classifiers towards harder examples by reweighting the training examples (small majority $\Rightarrow$ large weight)

  Example: suppose we already have $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$. We train $h_{m+1}(\mathbf{x})$ on a reweighted training set

$$\text{Weight } p(i) \text{ on } (\mathbf{x}_i, y_i):\quad p(i) \propto \exp\left(-y_i \sum_{k=1}^{m} h_k(\mathbf{x}_i) \right) \tag{5}$$

  so that examples with small or negative classification margin will have a large weight

- Why should we weight each classifier equally?

Boosting

- A Boosting algorithm sequentially estimates and combines classifiers by reweighting training sets (concentrating on the harder examples)
- AdaBoost preliminaries:

  a) Training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ with binary ± 1 labels y_i.

  b) A set of possible "weak" binary (± 1) classifiers $h(\mathbf{x}; \mathbf{w})$, $\mathbf{w} \in \mathcal{W}$ such as *decision stumps*.

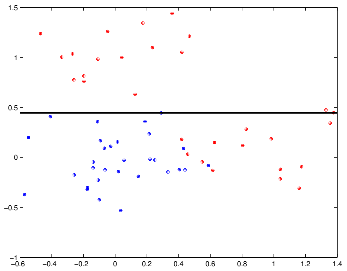
  Each decision stump pays attention to only a single component of the input vector

$$h(\mathbf{x}; \mathbf{w}) = \text{sign}(\, w_1 \, x_k - w_0 \,) \tag{6}$$

  where $\mathbf{w} = \{w_1, w_0, k\}$.

  c) Initially all examples have equal weights: $p(i) = 1/n$.

Decision stump



Each decision stump pays attention to only a single component of the input vector

$$h(\mathbf{x};\mathbf{w}) = \text{sign}(\, w_1\, x_k - w_0\,) \qquad (7)$$

where $\mathbf{w} = \{w_1, w_0, k\}$.

AdaBoost

1: Find (any) classifier $h(\mathbf{x};\hat{\mathbf{w}}_k)$ such that the weighted training error

$$\epsilon_k = \sum_{i=1}^{n} p(i)\,[y_i \neq h(\mathbf{x}_i;\hat{\mathbf{w}}_k)] \qquad (8)$$

is better than chance. (Here $[y \neq y'] = 1$ if $y \neq y'$ and zero otherwise).

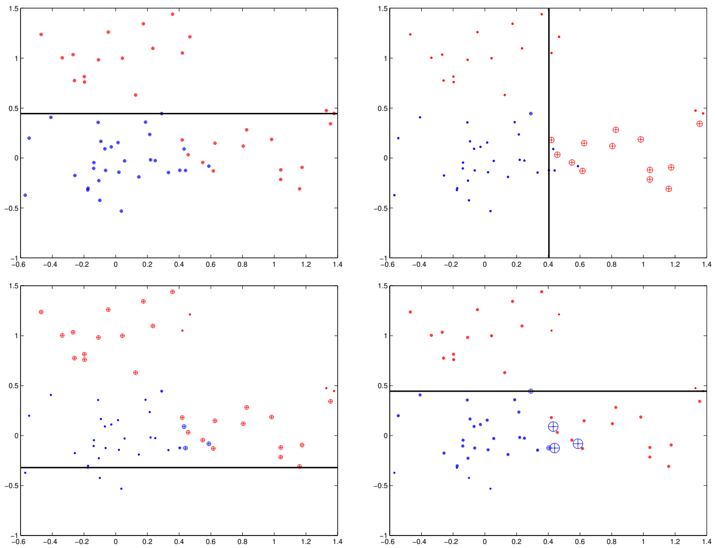
2: Set $\hat{\alpha}_k = \log(\,(1-\epsilon_k)/\epsilon_k\,)$

3: Update example weights: $p(i) \leftarrow p(i)\,\cdot \exp(\,-\hat{\alpha}_k\, y_i\, h(\mathbf{x}_i;\hat{\mathbf{w}}_k)\,)$ and renormalize.

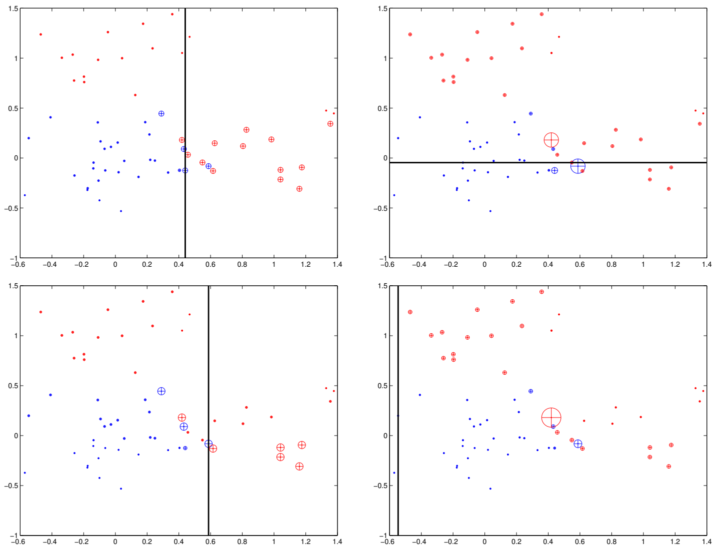
- The final hypothesis after m boosting iterations is given by

$$\hat{h}(\mathbf{x}) = \text{sign}(\,\hat{\alpha}_1 h(\mathbf{x};\hat{\mathbf{w}}_1) + \ldots + \hat{\alpha}_m h(\mathbf{x};\hat{\mathbf{w}}_m)\,) \qquad (9)$$

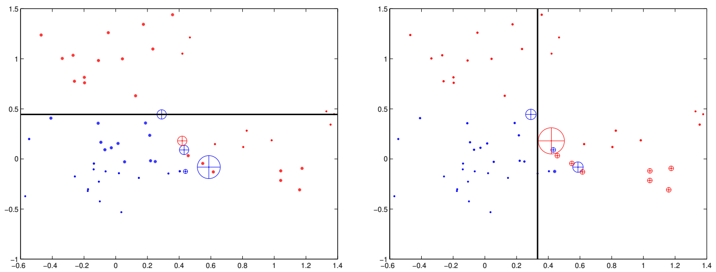
Boosting: example



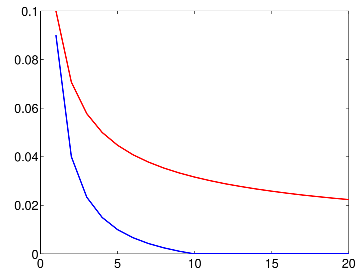
Boosting: example cont'd



Boosting: example cont'd

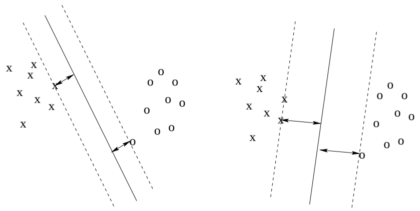


Boosting performance



- After training error reduces to zero, we can still be improving the test (generalization) error
- We can explain this in terms of the *margin*

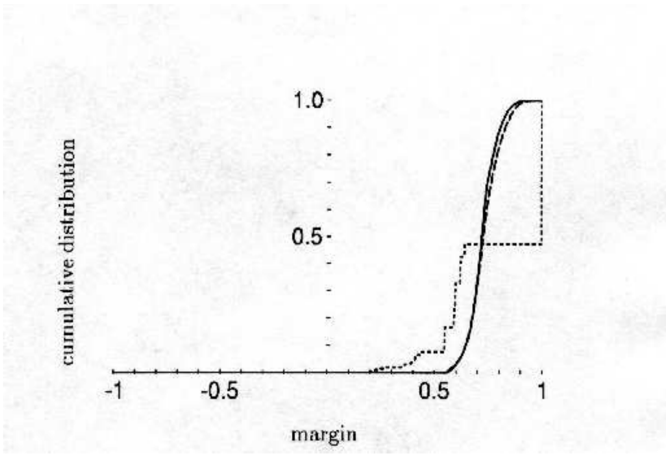
Classification margin



small margin large margin

- In both cases, the training error is zero
- Why would we prefer larger margin?

Boosting and margin



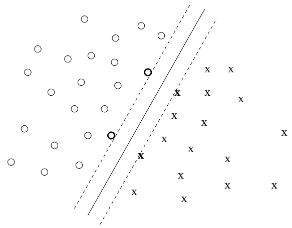
$$\text{Margin}(\mathbf{x}_i) = y_i \frac{\hat{\alpha}_1 h(\mathbf{x}_i; \hat{\mathbf{w}}_1) + \ldots + \hat{\alpha}_m h(\mathbf{x}_i; \hat{\mathbf{w}}_m)}{\hat{\alpha}_1 + \ldots + \hat{\alpha}_m} \tag{10}$$

Where we are

- Combining multiple methods
  - Sequential (forward) fitting
  - Bagging
  - Boosting
- *Support vector machines*
  - optimal hyperplane
  - kernel function

"Optimal" hyperplane

- Let's assume for simplicity that the classification problem is *linearly separable*



- Maximum margin hyperplane is maximally removed from all the training examples
- This hyperplane can be defined on the basis of only a few training examples called *support vectors*
- Leave-one-out cross-validation error estimate is bounded by the number of support vectors

"Optimal" hyperplane cont'd

Training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where the labels are binary ± 1

- Linear separator:

$$\begin{aligned} f(\mathbf{x}; \mathbf{w}, w_0) &= w_0 + x_1 w_1 + \ldots x_d w_d \\ &= w_0 + \mathbf{w}^T \mathbf{x} \end{aligned} \tag{11}$$

- We can try to enforce large margin classification by requiring that the sign of the decision boundary $w_0 + \mathbf{w}^T\mathbf{x}$ (clearly) agrees with the training labels

$$y_i \left[w_0 + \mathbf{w}^T \mathbf{x}_i \right] - 1 \geq 0, \quad i = 1, \ldots, n \tag{12}$$

BUT: larger values of $\mathbf{w}, w_0$ would yield larger "margins".

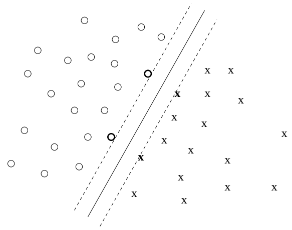
"Optimal" hyperplane cont'd

- We enforce large margin classification by minimizing

$$\|\mathbf{w}\|^2/2 = \sum_{j=1}^{d} w_i^2/2 \tag{13}$$

subject to the classification constraints

$$y_i \left[w_0 + \mathbf{w}^T \mathbf{x}_i \right] - 1 \geq 0, \quad i = 1, \ldots, n \tag{14}$$



- Only a few of the classification constraints are relevant
  $\Rightarrow$ support vectors

"Optimal" hyperplane cont'd

- We find the optimal setting of $\{w_0, \mathbf{w}\}$ by introducing Lagrange multipliers $\alpha_i \geq 0$ for the inequality constraints
- We *minimize*

$$J(\mathbf{w}, w_0, \alpha) = \|\mathbf{w}\|^2/2 - \sum_{i=1}^{n} \alpha_i \left(y_i \left[w_0 + \mathbf{w}^T \mathbf{x}_i \right] - 1 \right) \tag{15}$$

  with respect to $\mathbf{w}, w_0$. $\{\alpha_i\}$ are subsequently adjusted so that the classification constraints

$$y_i \left[w_0 + \mathbf{w}^T \mathbf{x}_i \right] - 1 \geq 0, \quad i = 1, \ldots, n \tag{16}$$

  are indeed satisfied.

- The setting of Lagrange multipliers corresponds to *maximizing* $J(\mathbf{w}, w_0, \alpha)$

Dual formulation

- Performing the optimization with respect to $\{w_0, \mathbf{w}\}$ leads to the dual form of support vector machines (we omit the details here):

  Maximize

$$J(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j) \tag{17}$$

  subject to the constraints

$$\alpha_i \geq 0, \quad i = 1, \ldots, n, \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{18}$$

  (For non-separable problems we have to limit $\alpha_i \leq C$)

Solution to support vector machines

- The resulting solution for $\mathbf{w}$ is a weighted combination of the training examples:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \mathbf{x}_i \tag{19}$$

  where the "weight" $\alpha_i = 0$ for all but the *support vectors*

- The decision boundary has an interpretable form

$$\hat{\mathbf{w}}^T \mathbf{x} + w_0 = \sum_{i=1}^{n} \hat{\alpha}_i \, y_i \, (\mathbf{x}_i^T \mathbf{x}) + w_0 \tag{20}$$

Interpretation of support vector machines

- To use support vector machines we have specify
  1. Weight (importance) of training examples: $\{\alpha_i\}$
  2. Pairwise similarity between the examples: $(\mathbf{x}_i^T \mathbf{x})$

  The weights $\{\alpha_i\}$ are adjusted to satisfy the classification constraints

- We make decisions by comparing the new example $\mathbf{x}$ to each (relevant) training example $\mathbf{x}_i$:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{n} \hat{\alpha}_i \, y_i \, (\mathbf{x}_i^T \mathbf{x}) + w_0 \right) \tag{21}$$

- The similarity or inner product $(\mathbf{x}_i^T \mathbf{x})$ is known as the *kernel function*