Kernel SVM and Ensemble methods

Sriram Sankararaman

The instructor gratefully acknowledges Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

Outline

1. Review of last lecture
   - SVM Dual Derivation and Support Vectors
2. Ensemble methods
3. Boosting

Support Vector Machine

- A linear classifier (hyperplane) that maximizes the margin.
- A linear classifier that minimizes the (l_2-regularized) hinge loss.
- Both lead to the same constrained optimization problem.

Constrained optimization

- "Primal" and "Dual" problems
- The dual solution is a lower bound to the primal (weak duality).
- For some problems (including the SVM problem), the dual solution is equal to the primal solution (strong duality).
- KKT conditions (relates the primal and dual variables at the optimal solution).

SVM: Primal and Dual

We can solve either the primal or the dual.

Primal (soft-margin SVM)

$$
\begin{aligned}
\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\
\text{s.t.} \quad & 1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \le \xi_n, \quad n \in \{1, \ldots, N\} \\
& 0 \le \xi_n, \quad n \in \{1, \ldots, N\}
\end{aligned}
$$

Dual

We can solve either the primal or the dual.

$$
\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\
\text{s.t.} \quad & 0 \leq \alpha_n \leq C, \quad n \in 1, \ldots, N \\
& \sum_n \alpha_n y_n = 0
\end{aligned}
$$

Dual tells us how to kernelize. Replace the inner products $\boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)$ with a kernel function.

Derivation of the dual

We will derive the dual formulation as the process will reveal some interesting and important properties of SVM. Particularly, what are "support vectors"?

Recipe

- Formulate the Lagrangian function that incorporates the constraints and introduces dual variables
- Minimize the Lagrangian function over the primal variables
- Substitute the primal variables for dual variables in the Lagrangian
- Maximize the Lagrangian with respect to dual variables
- Recover the solution (for the primal variables) from the dual variables

Deriving the dual for SVM

Primal SVM

$$\begin{aligned}
\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\
\text{s.t.} \quad & 1 - \xi_n - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \leq 0 \quad n \in 1, \ldots, N \\
& -\xi_n \leq 0, \quad n \in 1, \ldots, N
\end{aligned}$$

Deriving the dual for SVM

Primal SVM

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n$$

$$\text{s.t.} \quad 1 - \xi_n - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \le 0 \quad n \in 1, \ldots, N$$

$$-\xi_n \le 0, \quad n \in 1, \ldots, N$$

Lagrangian

$$L(\boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) = C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_n \alpha_n\{1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\} - \sum_n \lambda_n \xi_n$$

under the constraint that $\alpha_n \ge 0$ and $\lambda_n \ge 0$ (because these are lagrange multipliers corresponding to inequality constraints).

The dual problem

$$\max_{\alpha_n \geq 0, \lambda \geq 0} g(\{\alpha_n\}, \{\lambda_n\})$$

$$g(\{\alpha_n\}, \{\lambda_n\}) = \min_{\boldsymbol{w}, b, \{\xi_n\}} L(\boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\})$$

To compute g, we need to minimize the Lagrangian with respect to the primal variables.

Minimizing the Lagrangian with respect to the primal variables

Taking derivatives with respect to the primal variables

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{0}$$

$$\frac{\partial L}{\partial b} = \sum_n \alpha_n y_n = 0$$

$$\frac{\partial L}{\partial \xi_n} = C - \lambda_n - \alpha_n = 0$$

Minimizing the Lagrangian with respect to the primal variables

Taking derivatives with respect to the primal variables

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{0}$$

$$\frac{\partial L}{\partial b} = \sum_n \alpha_n y_n = 0$$

$$\frac{\partial L}{\partial \xi_n} = C - \lambda_n - \alpha_n = 0$$

These equations link the primal variables and the dual variables and provide new constraints on the dual variables:

$$\boldsymbol{w} = \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

$$\sum_n \alpha_n y_n = 0$$

$$C - \lambda_n - \alpha_n = 0$$

Substitute the solution back into the Lagrangian

$$g(\{\alpha_n\},\{\lambda_n\}) = L(\boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\})$$

Substitute the solution back into the Lagrangian

$$
\begin{aligned}
g(\{\alpha_n\},\{\lambda_n\}) &= L(\boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \\
&= \sum_n (C - \alpha_n - \lambda_n)\xi_n + \frac{1}{2}\|\sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)\|_2^2 + \sum_n \alpha_n \\
&+ \left(\sum_n \alpha_n y_n\right) b - \sum_n \alpha_n y_n \left(\sum_m y_m \alpha_m \boldsymbol{\phi}(\boldsymbol{x}_m)\right)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)
\end{aligned}
$$

Substitute the solution back into the Lagrangian

$$
\begin{aligned}
g(\{\alpha_n\},\{\lambda_n\}) &= L(\boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \\
&= \sum_n (C - \alpha_n - \lambda_n)\xi_n + \frac{1}{2}\|\sum_n y_n\alpha_n\boldsymbol{\phi}(\boldsymbol{x}_n)\|_2^2 + \sum_n \alpha_n \\
&+ \left(\sum_n \alpha_n y_n\right) b - \sum_n \alpha_n y_n \left(\sum_m y_m\alpha_m\boldsymbol{\phi}(\boldsymbol{x}_m)\right)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= \sum_n \alpha_n + \frac{1}{2}\|\sum_n y_n\alpha_n\boldsymbol{\phi}(\boldsymbol{x}_n)\|_2^2 - \sum_{m,n} \alpha_n\alpha_m y_m y_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n)
\end{aligned}
$$

Substitute the solution back into the Lagrangian

$$
\begin{aligned}
g(\{\alpha_n\},\{\lambda_n\}) &= L(\boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \\
&= \sum_n (C - \alpha_n - \lambda_n)\xi_n + \frac{1}{2}\|\sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)\|_2^2 + \sum_n \alpha_n \\
&+ \left(\sum_n \alpha_n y_n\right) b - \sum_n \alpha_n y_n \left(\sum_m y_m \alpha_m \boldsymbol{\phi}(\boldsymbol{x}_m)\right)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= \sum_n \alpha_n + \frac{1}{2}\|\sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n)\|_2^2 - \sum_{m,n} \alpha_n \alpha_m y_m y_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\
&= \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} \alpha_n \alpha_m y_m y_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)
\end{aligned}
$$

Several terms vanish because of the constraints $\sum_n \alpha_n y_n = 0$ and $C - \lambda_n - \alpha_n = 0$.

The dual problem

Maximizing the dual under the constraints

$$
\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & g(\{\alpha_n\},\{\lambda_n\}) = \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n k(\boldsymbol{x}_m, \boldsymbol{x}_n) \\
\text{s.t.} \quad & \alpha_n \geq 0, \quad n \in 1,\ldots,N \\
& \sum_n \alpha_n y_n = 0 \\
& C - \lambda_n - \alpha_n = 0, \quad n \in 1,\ldots,N \\
& \lambda_n \geq 0, \quad n \in 1,\ldots,N
\end{aligned}
$$

The dual problem

Maximizing the dual under the constraints

$$\max_{\boldsymbol{\alpha}} \quad g(\{\alpha_n\}, \{\lambda_n\}) = \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(\boldsymbol{x}_m, \boldsymbol{x}_n)$$

$$\text{s.t.} \quad \alpha_n \geq 0, \quad n \in 1, \ldots, N$$

$$\sum_n \alpha_n y_n = 0$$

$$C - \lambda_n - \alpha_n = 0, \quad n \in 1, \ldots, N$$

$$\lambda_n \geq 0, \quad n \in 1, \ldots, N$$

We can simplify as the objective function does not depend on λ_n. Specifically, we can combine the constraints involving λ_n resulting in the following inequality constraint: $\alpha_n \leq C$:

$$\begin{aligned} C - \lambda_n - \alpha_n = 0, \; \lambda_n \geq 0 &\iff \lambda_n = C - \alpha_n \geq 0 \\ &\iff \alpha_n \leq C \end{aligned}$$

Simplified Dual

$$
\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) \\
\text{s.t.} \quad & 0 \leq \alpha_n \leq C, \quad n \in 1, \ldots, N \\
& \sum_n \alpha_n y_n = 0
\end{aligned}
$$

Recovering solution to the primal formulation

We already identified the primal variable $\boldsymbol{w}$ as

$$\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$$

- From the dual, we know that $0 \leq \alpha_n \leq C$.
- If $\alpha_n = 0$, then the example n does not affect the hyperplane/decision boundary computed by SVM.
- Only those examples with $\alpha_n > 0$ affect the hyerplane/decision boundary. These examples are termed support vectors.
- When will $\alpha_n > 0$?

Complementary slackness and support vectors

At the optimal solution to both primal and dual, the following condition must hold due to the KKT (Karsh-Kuhn-Tucker) conditions:

$$\lambda_n \xi_n = 0$$
$$\alpha_n \{1 - \xi_n - y_n[\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) + b]\} = 0$$

- At the optimum, product of lagrange multipler for a constraint and the value of the constraint equals zero

Complementary slackness and support vectors

$$\lambda_n \xi_n = 0$$
$$\alpha_n \{1 - \xi_n - y_n[\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) + b]\} = 0$$

From the second condition, if $\alpha_n > 0$, then

$$1 - \xi_n - y_n[\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) + b] = 0$$

What are support vectors?

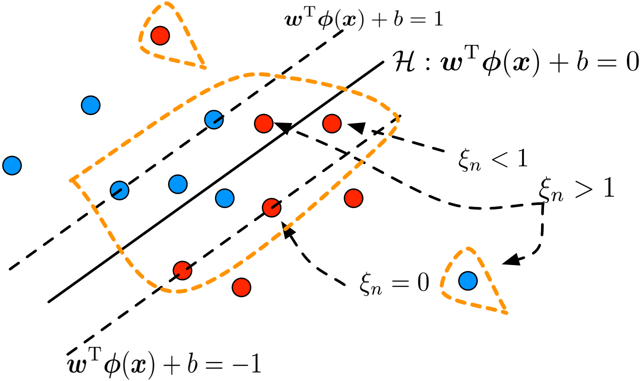
Case analysis Since, we have

$$1 - \xi_n - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b]\} = 0$$

We have

- $\xi_n = 0$. This implies $y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] = 1$. They are on points that are $1/\|\boldsymbol{w}\|_2$ away from the decision boundary.
- $\xi_n < 1$. These are points that can be classified correctly but do not satisfy the large margin constraint – they have smaller distances to the decision boundary.
- $\xi_n > 1$. These are points that are misclassified.

Visualization of how training data points are categorized



Support vectors are highlighted by the dotted orange lines

Mini-Summary

- By converting the SVM problem from primal to dual, we can also kernelize the SVM.
- Led us to study optimization problems with constraints.
  - Lagrangian and lagrange multipliers (how to deal with constraints)
  - Primal vs dual
  - Complementary slackness
- The dual also allows us to understand what the "support vectors" are.

Outline

1. Review of last lecture
2. Ensemble methods
   - Multiple classifiers on the same data
   - Same classifier on multiple data
3. Boosting

Idea

- Consider a set of predictors (base learners): $h_1, \ldots, h_L$.
- Combine their predictions to get a more accurate predictor H: (ensemble).

When might this work ?

Idea

- Consider a set of predictors (base learners): $h_1, \ldots, h_L$.
- Combine their predictions to get a more accurate predictor H: (ensemble).

When might this work ?

The predictors make different types of mistakes.

Practical application: the Netflix prize

Goal: predict how a user will rate a movie

- Based on other user's ratings of the movie.
- Based on this user's ratings of other movies.
- Application called collaborative filtering.
- Netflix prize: $1M$ prize for the first team to do 10% better than Netflix system.
- Winner: an ensemble of more than 800 rating systems.

Train multiple classifiers on the same dataset

- Train $h_1, \ldots, h_L$ on a given training data (*e.g.*, decision trees, logistic regression).
- How to combine their predictions ?

Train multiple classifiers on the same dataset

- Train $h_1, \ldots, h_L$ on a given training data (*e.g.*, decision trees, logistic regression).
- How to combine their predictions ?
- Majority vote : $h = \textsc{sign}(h_1 + .. + h_L)$, where $h_l \in \{+1, -1\}$.

Train multiple classifiers on the same dataset

- Train $h_1, \ldots, h_L$ on a given training data (*e.g.*, decision trees, logistic regression).
- How to combine their predictions ?
- Majority vote : $h = \text{SIGN}(h_1 + .. + h_L)$, where $h_l \in \{+1, -1\}$.
- Weighted majority vote: $h = \text{SIGN}(w_1 h_1 + .. + w_L h_L)$, where $h_l \in \{+1, -1\}$. How do we set the weights ?

Train multiple classifiers on the same dataset

- Train $h_1, \ldots, h_L$ on a given training data (*e.g.*, decision trees, logistic regression).
- How to combine their predictions ?
- Majority vote : $h = \text{SIGN}(h_1 + .. + h_L)$, where $h_l \in \{+1, -1\}$.
- Weighted majority vote: $h = \text{SIGN}(w_1 h_1 + .. + w_L h_L)$, where $h_l \in \{+1, -1\}$. How do we set the weights ?
  - Compute weights on validation data, *i.e.*, data not used for learning $h_1, \ldots, h_L$.

Train multiple classifiers on the same dataset

- Train $h_1, \ldots, h_L$ on a given training data (*e.g.*, decision trees, logistic regression).
- How to combine their predictions ?
- Majority vote : $h = \textsc{sign}(h_1 + .. + h_L)$, where $h_l \in \{+1, -1\}$.
- Weighted majority vote: $h = \textsc{sign}(w_1 h_1 + .. + w_L h_L)$, where $h_l \in \{+1, -1\}$. How do we set the weights ?
    - Compute weights on validation data, *i.e.*, data not used for learning $h_1, \ldots, h_L$.
    - What if we are trying to learn a regression model ?

Train multiple classifiers on the same dataset

- Train $h_1, \ldots, h_L$ on a given training data (*e.g.*, decision trees, logistic regression).
- How to combine their predictions ?
- Majority vote : $h = \text{SIGN}(h_1 + .. + h_L)$, where $h_l \in \{+1, -1\}$.
- Weighted majority vote: $h = \text{SIGN}(w_1 h_1 + .. + w_L h_L)$, where $h_l \in \{+1, -1\}$. How do we set the weights ?
  - Compute weights on validation data, *i.e.*, data not used for learning $h_1, \ldots, h_L$.
  - What if we are trying to learn a regression model ?
- Can use mean, weighted mean, or median.

Bagging (Bootstrap Aggregation)

Training same classifier on multiple datasets

- Where do we get multiple datasets ?
- One idea: split original training dataset into multiple datasets and train a classifier on each.
  - Each classifier is trained on a small part of the data and so might not generalize well.

Bagging (Bootstrap Aggregation)

Training same classifier on multiple datasets

- Where do we get multiple datasets ?
- Bootstrap resampling
  - Training data with N instances: $\mathcal{D}$
  - Create B bootstrap training datasets: $\tilde{\mathcal{D}}_1, \ldots, \tilde{\mathcal{D}}_B$.
  - Each $\tilde{\mathcal{D}}_b$ contains N training examples drawn randomly from $\mathcal{D}$ with replacement.
  - Train a classifier h_b on each of $\tilde{\mathcal{D}}_b$.
  - Use a majority vote of $h_1, \ldots, h_B$ to classify test data.

Outline

1. Review of last lecture
2. Ensemble methods
3. Boosting
   - AdaBoost
   - Derivation of AdaBoost

Boosting

High-level idea: combine a lot of classifiers

- Construct / identify these classifiers one at a time
- Use *weak or base* classifiers to arrive at complex decision boundaries (*strong* classifiers)

Our plan

- Describe AdaBoost algorithm
- Derive the algorithm

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

  2. Compute contribution for this classifier

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

$$w_{t+1}(n) \propto w_t(n)e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

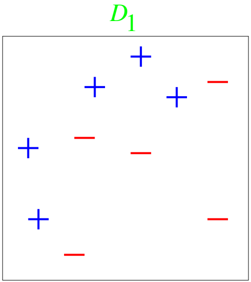
  and normalize them such that $\sum_n w_{t+1}(n) = 1$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error
  $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$
  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points
  $$w_{t+1}(n) \propto w_t(n)e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$
  and normalize them such that $\sum_n w_{t+1}(n) = 1$
- Output the final classifier
$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T}\beta_t h_t(\boldsymbol{x})\right]$$

Example

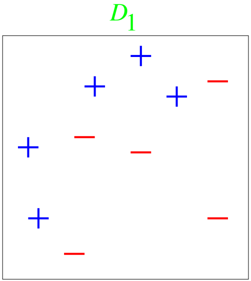
10 data points and 2 features



- The data points are clearly not linear separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")

Example

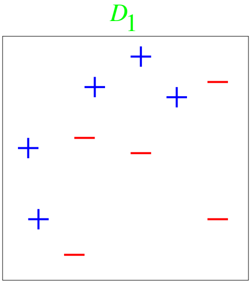
10 data points and 2 features



- The data points are clearly not linear separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: either horizontal or vertical lines

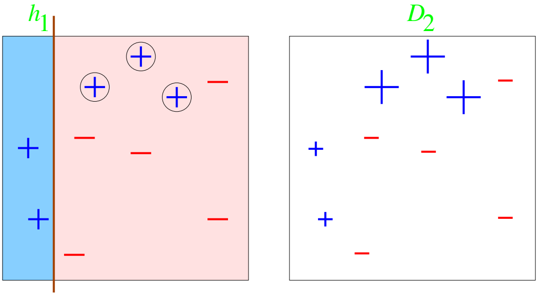
Example

10 data points and 2 features

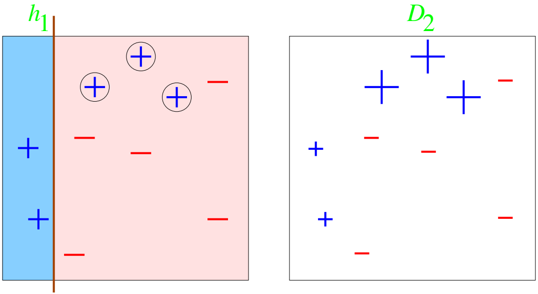


- The data points are clearly not linear separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: either horizontal or vertical lines
  - These 'decision stumps' are just trees with a single internal node, i.e., they classifying data based on a single attribute

Round 1: $t = 1$

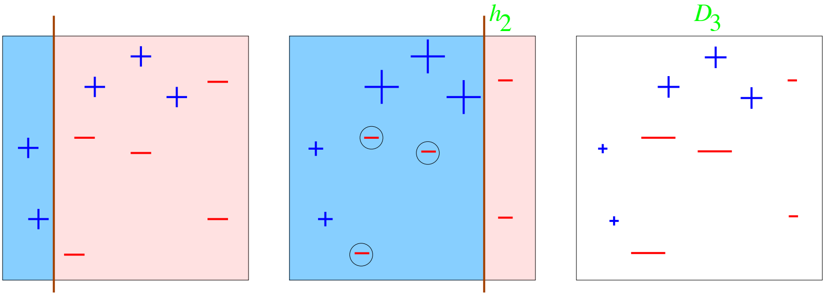


Round 1: $t = 1$

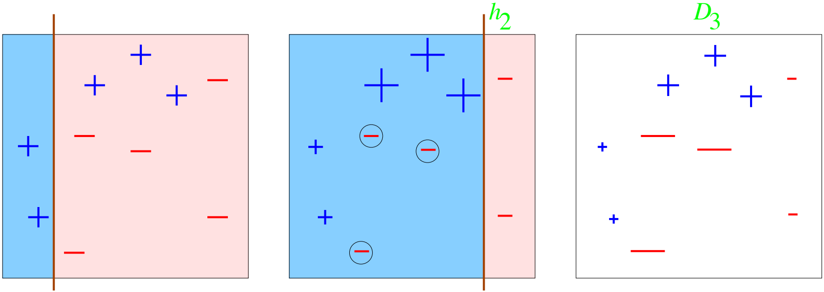


- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$.
- Weights recomputed; the 3 misclassified data points receive larger weights

Round 2: $t = 2$

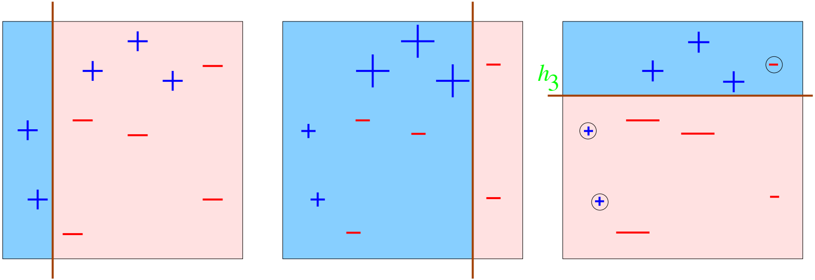


Round 2: $t = 2$

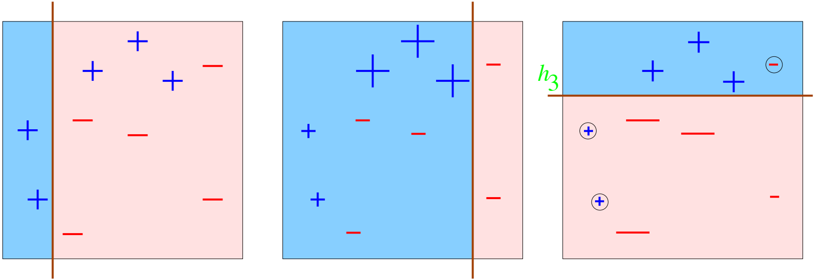


- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
  Note that $\epsilon_2 \neq 0.3$ as those 3 data points have weights less than $1/10$
- 3 misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights

Round 3: $t = 3$

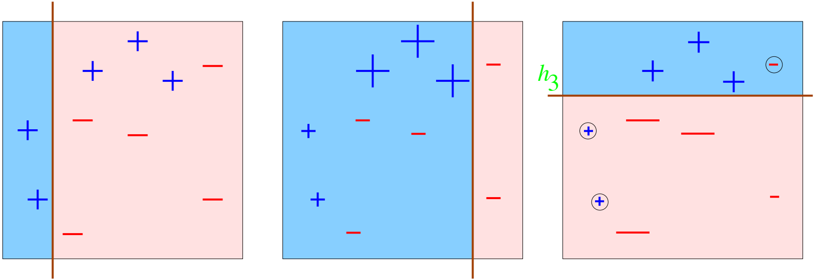


Round 3: $t = 3$



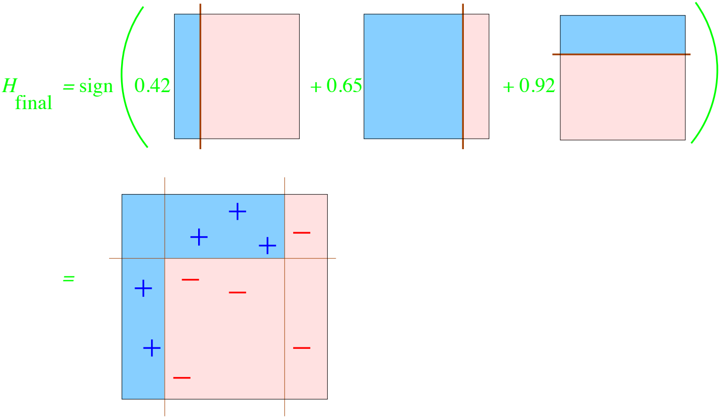
- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?

Round 3: $t = 3$



- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?
  - Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction

Final classifier: combining 3 classifiers



- All data points are now classified correctly!

Why AdaBoost works?

It minimizes a loss function related to the classification error.

0/1 loss

- Suppose we want to have a classifier

$$h(\boldsymbol{x}) = \text{sign}[a(\boldsymbol{x})] = \begin{cases} 1 & \text{if } a(\boldsymbol{x}) > 0 \\ -1 & \text{if } a(\boldsymbol{x}) < 0 \end{cases}$$

- Our loss function is thus

$$\ell(y, h(\boldsymbol{x})) = \begin{cases} 0 & \text{if } ya(\boldsymbol{x}) > 0 \\ 1 & \text{if } ya(\boldsymbol{x}) < 0 \end{cases}$$

Namely, the function $a(\boldsymbol{x})$ and the target label y should have the same sign to avoid a loss of 1.

Surrogate loss

$0 - 1$ loss function $\ell(h(\boldsymbol{x}), y)$ is non-convex and difficult to optimize.

We can instead use a surrogate loss – what are examples?

Surrogate loss

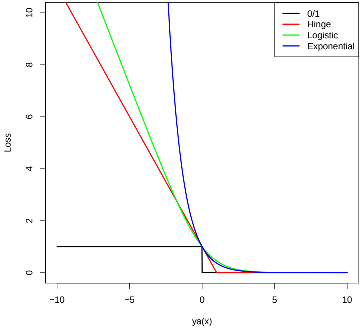
$0-1$ loss function $\ell(h(\boldsymbol{x}), y)$ is non-convex and difficult to optimize.

We can instead use a surrogate loss – what are examples?

Exponential Loss

$$\ell^{\mathrm{EXP}}(y, h(\boldsymbol{x})) = e^{-ya(\boldsymbol{x})}$$

$\ell^{\mathrm{EXP}}(y, h(\boldsymbol{x}))$ is easier to optimze as it is differentiable



Choosing the t-th classifier

Suppose we have built a classifier $a_{t-1}(\boldsymbol{x})$, and we want to improve it by adding a weak learner $h_t(\boldsymbol{x})$

$$a(\boldsymbol{x}) = a_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

How can we choose optimally the new classifier $h_t(\boldsymbol{x})$ and the combination coefficient β_t?

Adaboost greedily *minimizes the empirical risk of the exponential loss function*.

$$(h_t^*(\boldsymbol{x}), \beta_t^*) = \arg\min_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n a(\boldsymbol{x}_n)}$$

Choosing the t-th classifier

Suppose we have built a classifier $a_{t-1}(\boldsymbol{x})$, and we want to improve it by adding a weak learner $h_t(\boldsymbol{x})$

$$a(\boldsymbol{x}) = a_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

How can we choose optimally the new classifier $h_t(\boldsymbol{x})$ and the combination coefficient β_t?

Adaboost greedily *minimizes the empirical risk of the exponential loss function*.

$$\begin{aligned}(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n a(\boldsymbol{x}_n)} \\ &= \arg\min_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n [a_{t-1}(\boldsymbol{x}_n) + \beta_t h_t(\boldsymbol{x}_n)]}\end{aligned}$$

Choosing the t-th classifier

Suppose we have built a classifier $a_{t-1}(\boldsymbol{x})$, and we want to improve it by adding a weak learner $h_t(\boldsymbol{x})$

$$a(\boldsymbol{x}) = a_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

How can we choose optimally the new classifier $h_t(\boldsymbol{x})$ and the combination coefficient β_t?

Adaboost greedily *minimizes the empirical risk of the exponential loss function*.

$$\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n a(\boldsymbol{x}_n)} \\
&= \arg\min_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n [a_{t-1}(\boldsymbol{x}_n) + \beta_t h_t(\boldsymbol{x}_n)]} \\
&= \arg\min_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}
\end{aligned}$$

where we have used $w_t(n)$ as a shorthand for $e^{-y_n a_{t-1}(\boldsymbol{x}_n)}$

The new classifier

We can decompose the *weighted* loss function into two parts

$$\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}$$
$$= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)]$$

The new classifier

We can decompose the *weighted* loss function into two parts

$$
\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)] \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)])
\end{aligned}
$$

The new classifier

We can decompose the *weighted* loss function into two parts

$$
\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)] \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]) \\
&= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

We have used the following properties to derive the above

- $y_n h_t(\boldsymbol{x}_n)$ is either 1 or -1 as $h_t(\boldsymbol{x}_n)$ is the output of a binary classifier
- The indicator function $\mathbb{I}[y_n = h_t(\boldsymbol{x}_n)]$ is either 0 or 1, so it equals $1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$

Finding the optimal weak learner

Summary

$$
\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\
&\qquad + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

What term(s) must we optimize to choose $h_t(x_n)$?

Finding the optimal weak learner

Summary

$$(h_t^*(\boldsymbol{x}), \beta_t^*) = \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}$$

$$= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

$$+ e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize to choose $h_t(x_n)$?

$$h_t^*(\boldsymbol{x}) = \arg\min_{h_t(\boldsymbol{x})} \epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

Finding the optimal weak learner

Summary

$$
\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\
&\qquad + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

What term(s) must we optimize to choose $h_t(x_n)$?

$$
h_t^*(\boldsymbol{x}) = \arg\min_{h_t(\boldsymbol{x})} \epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]
$$

Minimize weighted classification error as noted in step 1 of Adaboost!

How to choose β_t?

Summary

$$
\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\
&\qquad + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

What term(s) must we optimize to choose β_t?

How to choose β_t?

Summary

$$
\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\
&\qquad + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

What term(s) must we optimize to choose β_t?

We need to minimize the entire objective function with respect to β_t!

How to choose β_t?

Summary

$$(h_t^*(\boldsymbol{x}), \beta_t^*) = \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}$$
$$= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize to choose β_t?

We need to minimize the entire objective function with respect to β_t!

We can do this by taking derivative with respect to β_t, setting to zero, and solving for β_t. After some calculation and using $\sum_n w_t(n) = 1$, we find:

$$\beta_t^* = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

which is precisely step 2 of Adaboost! (*Exercise – verify the solution*)

Updating the weights

Once we find the optimal weak learner we can update our classifier:

$$a(\boldsymbol{x}) = a_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

Updating the weights

Once we find the optimal weak learner we can update our classifier:

$$a(\boldsymbol{x}) = a_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

We then need to compute the weights for the above classifier as:

$$w_{t+1}(n) = e^{-y_n a(\boldsymbol{x}_n)} = e^{-y_n [a_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x}_n)]}$$

Updating the weights

Once we find the optimal weak learner we can update our classifier:

$$a(\boldsymbol{x}) = a_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

We then need to compute the weights for the above classifier as:

$$\begin{aligned} w_{t+1}(n) &= e^{-y_n a(\boldsymbol{x}_n)} = e^{-y_n[a_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x}_n)]} \\ &= w_t(n) e^{-y_n \beta_t^* h_t^*(\boldsymbol{x}_n)} \end{aligned}$$

Updating the weights

Once we find the optimal weak learner we can update our classifier:

$$a(\boldsymbol{x}) = a_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

We then need to compute the weights for the above classifier as:

$$\begin{aligned} w_{t+1}(n) &= e^{-y_n a(\boldsymbol{x}_n)} = e^{-y_n[a_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x}_n)]} \\ &= w_t(n) e^{-y_n \beta_t^* h_t^*(\boldsymbol{x}_n)} = \begin{cases} w_t(n) e^{\beta_t^*} & \text{if } y_n \neq h_t^*(\boldsymbol{x}_n) \\ w_t(n) e^{-\beta_t^*} & \text{if } y_n = h_t^*(\boldsymbol{x}_n) \end{cases} \end{aligned}$$

Intuition Misclassified data points will get their weights increased, while correctly classified data points will get their weight decreased

Meta-Algorithm

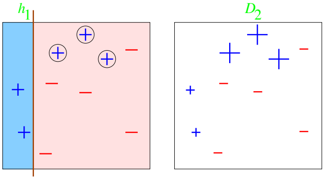
Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\boldsymbol{x})$ as long as it minimizes the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t^*(\boldsymbol{x}_n)]$$

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any type of classifier

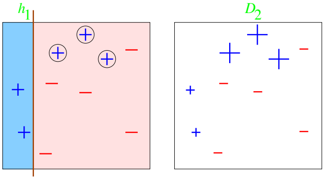
Decision Stumps

How do we choose the decision stump classifier given the weights at the second round of the following distribution?



Decision Stumps

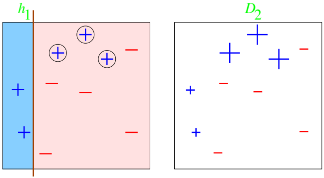
How do we choose the decision stump classifier given the weights at the second round of the following distribution?



We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error! Runtime?

Decision Stumps

How do we choose the decision stump classifier given the weights at the second round of the following distribution?



We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error! Runtime?

- Presort data by each feature in $O(dN \log N)$ time
- Evaluate $N + 1$ thresholds for each feature at each round in $O(dN)$ time
- In total $O(dN \log N + dNT)$ time – this efficiency is an attractive quality of boosting!

Interpreting boosting as learning nonlinear basis

Two-stage process

- Get $\textsc{sign}[a_1(\boldsymbol{x})]$, $\textsc{sign}[a_2(\boldsymbol{x})]$,$\cdots$,
- Combine into a linear classification model

$$y = \textsc{sign}\left\{\sum_t \beta_t \textsc{sign}[a_t(\boldsymbol{x})]\right\}$$

Interpreting boosting as learning nonlinear basis

Two-stage process

- Get $\text{SIGN}[a_1(\boldsymbol{x})]$, $\text{SIGN}[a_2(\boldsymbol{x})]$,$\cdots$,
- Combine into a linear classification model

$$y = \text{SIGN}\left\{\sum_t \beta_t \text{SIGN}[a_t(\boldsymbol{x})]\right\}$$

In other words, each stage learns a nonlinear basis $\phi_t(\boldsymbol{x}) = \text{SIGN}[a_t(\boldsymbol{x})]$

- This is an alternative way to introduce non-linearity aside from kernel methods
- We could also try to learn the basis functions and the classifier at the same time, as we'll talk about with neural networks later.