Boosting

Professor Ameet Talwalkar

Outline

Grade Policy and Final Exam

Final Grades

- HWs (30%), midterm (30%), and final exam (40%) of final grade
- The final grades will be curved so that the median grade is either a B or B+ (I have not yet decided)
- I may increase weight of final for students who do much better on final than midterm (I don't have a strict policy in place though)

Final Exam

- Cumulative but with more emphasis on new material
- On last day of class (Wednesday, 3/15)

Outline

Support vector machines (SVM)

Primal Formulation

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

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

Two equivalent interpretations

Support vector machines (SVM)

Primal Formulation

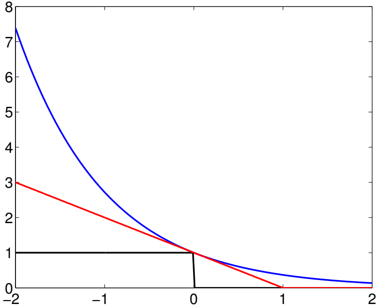
$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n$$

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

Two equivalent interpretations

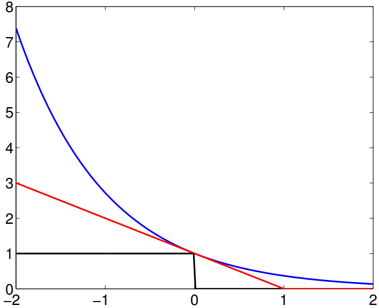
- Geometric: Maximizing (soft) margin
- Optimization: Minimize hinge loss with L2 regularization

Hinge Loss



- Upper-bound for $0/1$ loss function (black line)
- Convex surrogate to $0/1$ loss

Hinge Loss



- Upper-bound for $0/1$ loss function (black line)
- Convex surrogate to $0/1$ loss, though others exist as well
  - Hinge loss less sensitive to outliers than exponential (or logistic) loss
  - Logistic loss has a natural probabilistic interpretation
  - We can optimize exponential loss efficiently in a greedy manner (Adaboost)

Constrained Optimization

Primal Formulation

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

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

- When working with constrained optimization problems with inequality constraints, we can write down primal and dual problems

Constrained Optimization

Primal Formulation

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

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

- When working with constrained optimization problems with inequality constraints, we can write down primal and dual problems
- The dual solution is always a lower bound on the primal solution (weak duality)
- The duality gap equals 0 under certain conditions (strong duality), and in such cases we can either solve the primal or dual problem
- Strong duality holds for the SVM problem, and in particular the KKT conditions are necessary and sufficient for the optimal solution

Dual formulation of SVM and Kernel SVMs

$$\max_{\boldsymbol{\alpha}} \quad \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)$$

$$\text{s.t.} \quad 0 \leq \alpha_n \leq C, \quad \forall\ n$$

$$\sum_n \alpha_n y_n = 0$$

- Dual problem is also a convex quadratic programming

Dual formulation of SVM and Kernel SVMs

$$\max_{\boldsymbol{\alpha}} \quad \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n)$$

$$\text{s.t.} \quad 0 \leq \alpha_n \leq C, \quad \forall\ n$$

$$\sum_n \alpha_n y_n = 0$$

- Dual problem is also a convex quadratic programming involving N dual variables α_n
- Kernel SVM:

Dual formulation of SVM and Kernel SVMs

$$\max_{\boldsymbol{\alpha}} \quad \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)$$

$$\text{s.t.} \quad 0 \leq \alpha_n \leq C, \quad \forall\ n$$

$$\sum_n \alpha_n y_n = 0$$

- Dual problem is also a convex quadratic programming involving N dual variables α_n
- Kernel SVM:
  - Replace inner products $\boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n)$ with a kernel function, $k(\boldsymbol{x}_m, \boldsymbol{x}_n)$ when solving dual problem
  - Show that we can recover primal predictions at test time without relying explicitly on $\phi(\cdot)$.

Recovering primal solution using dual variables

Why do we care?

- Using solely a kernel function, we can solve the dual optimization problem and make predictions at test time!
- Prediction only depends on support vectors, i.e., points with $\alpha_n > 0$!

Recovering primal solution using dual variables

Why do we care?

- Using solely a kernel function, we can solve the dual optimization problem and make predictions at test time!
- Prediction only depends on support vectors, i.e., points with $\alpha_n > 0$!

Weights

$$\boldsymbol{w} = \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n) \leftarrow \text{ Linear combination of the input features}$$

Offset

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

Recovering primal solution using dual variables

Why do we care?

- Using solely a kernel function, we can solve the dual optimization problem and make predictions at test time!
- Prediction only depends on support vectors, i.e., points with $\alpha_n > 0$!

Weights

$$\boldsymbol{w} = \sum_n y_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n) \leftarrow \text{ Linear combination of the input features}$$

Offset

$$b = [y_n - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)] = [y_n - \sum_m y_m \alpha_m k(\boldsymbol{x}_m, \boldsymbol{x}_n)], \quad \text{for any } C > \alpha_n > 0$$

Prediction on a test point x

$$h(\boldsymbol{x}) = \text{SIGN}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) + b) = \text{SIGN}(\sum_n y_n \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x}) + b)$$

Deriving the dual for SVM

Primal SVM

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

$$\text{s.t.} \quad y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad \forall \;\; n$$

$$\xi_n \geq 0, \quad \forall \; n$$

Deriving the dual for SVM

Primal SVM

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

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

$$\xi_n \geq 0, \quad \forall \ n$$

Lagrangian

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

under the constraints that $\alpha_n \geq 0$ and $\lambda_n \geq 0$.

Minimizing the Lagrangian

Taking derivatives with respect to the primal variables

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

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

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

Minimizing the Lagrangian

Taking derivatives with respect to the primal variables

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

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

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

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

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

$$\sum_n \alpha_n y_n = 0$$

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

Rearrange the Lagrangian and incorporate these constraints

Recall:

- $L(\cdot) = C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n \xi_n + \sum_n \alpha_n\{1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\}$ where $\alpha_n \geq 0$ and $\lambda_n \geq 0$
- Constraints from partial derivatives: $\sum_n \alpha_n y_n = 0$ and $C - \lambda_n - \alpha_n = 0$

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

Rearrange the Lagrangian and incorporate these constraints

Recall:

- $L(\cdot) = C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n\xi_n + \sum_n \alpha_n\{1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\}$ where $\alpha_n \geq 0$ and $\lambda_n \geq 0$
- Constraints from partial derivatives: $\sum_n \alpha_n y_n = 0$ and $C - \lambda_n - \alpha_n = 0$

$$
\begin{aligned}
g(\{\alpha_n\},\{\lambda_n\}) &= L(\boldsymbol{w}, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \\
&= \sum_n (C - \alpha_n - \lambda_n)\xi_n
\end{aligned}
$$

Rearrange the Lagrangian and incorporate these constraints

Recall:

- $L(\cdot) = C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n \xi_n + \sum_n \alpha_n\{1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\}$ where $\alpha_n \geq 0$ and $\lambda_n \geq 0$
- Constraints from partial derivatives: $\sum_n \alpha_n y_n = 0$ and $C - \lambda_n - \alpha_n = 0$

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

Rearrange the Lagrangian and incorporate these constraints

Recall:

- $L(\cdot) = C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n\xi_n + \sum_n \alpha_n\{1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\}$ where $\alpha_n \geq 0$ and $\lambda_n \geq 0$
- Constraints from partial derivatives: $\sum_n \alpha_n y_n = 0$ and $C - \lambda_n - \alpha_n = 0$

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

Rearrange the Lagrangian and incorporate these constraints

Recall:

- $L(\cdot) = C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n\xi_n + \sum_n \alpha_n\{1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\}$ where $\alpha_n \geq 0$ and $\lambda_n \geq 0$
- Constraints from partial derivatives: $\sum_n \alpha_n y_n = 0$ and $C - \lambda_n - \alpha_n = 0$

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

Rearrange the Lagrangian and incorporate these constraints

Recall:

- $L(\cdot) = C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n\xi_n + \sum_n \alpha_n\{1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\}$ where $\alpha_n \geq 0$ and $\lambda_n \geq 0$
- Constraints from partial derivatives: $\sum_n \alpha_n y_n = 0$ and $C - \lambda_n - \alpha_n = 0$

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

Rearrange the Lagrangian and incorporate these constraints

Recall:

- $L(\cdot) = C\sum_n \xi_n + \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_n \lambda_n\xi_n + \sum_n \alpha_n\{1 - y_n[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) + b] - \xi_n\}$ where $\alpha_n \geq 0$ and $\lambda_n \geq 0$
- Constraints from partial derivatives: $\sum_n \alpha_n y_n = 0$ and $C - \lambda_n - \alpha_n = 0$

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

The dual problem

Maximizing the dual under the constraints

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

$$\text{s.t.} \quad \alpha_n \geq 0, \quad \forall\, n$$

$$\sum_n \alpha_n y_n = 0$$

$$C - \lambda_n - \alpha_n = 0, \quad \forall\, n$$

$$\lambda_n \geq 0, \quad \forall\, n$$

The dual problem

Maximizing the dual under the constraints

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

$$\begin{aligned}
\text{s.t.} \quad & \alpha_n \geq 0, \quad \forall\ n \\
& \sum_n \alpha_n y_n = 0 \\
& C - \lambda_n - \alpha_n = 0, \quad \forall\ n \\
& \lambda_n \geq 0, \quad \forall\ n
\end{aligned}$$

We can simplify as the objective function does not depend on λ_n.

$$C - \lambda_n - \alpha_n = 0,\ \lambda_n \geq 0 \iff \lambda_n = C - \alpha_n \geq 0$$

The dual problem

Maximizing the dual under the constraints

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

$$\text{s.t.} \quad \alpha_n \geq 0, \quad \forall\ n$$

$$\sum_n \alpha_n y_n = 0$$

$$C - \lambda_n - \alpha_n = 0, \quad \forall\ n$$

$$\lambda_n \geq 0, \quad \forall\ n$$

We can simplify as the objective function does not depend on λ_n.

$$C - \lambda_n - \alpha_n = 0,\ \lambda_n \geq 0 \iff \lambda_n = C - \alpha_n \geq 0$$

$$\iff 0 \leq \alpha_n \leq C$$

Simplified Dual

$$\max_{\boldsymbol{\alpha}} \quad \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n)$$

$$\text{s.t.} \quad 0 \leq \alpha_n \leq C, \quad \forall\ n$$

$$\sum_n \alpha_n y_n = 0$$

Outline

Boosting

High-level idea: combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where β_t is the contribution of each weak classifier

Boosting

High-level idea: combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where β_t is the contribution of each weak classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

Boosting

High-level idea: combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where β_t is the contribution of each weak classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

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

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

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

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

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

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  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

     $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  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
  $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$
  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

     $$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

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

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

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

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  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

     $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

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

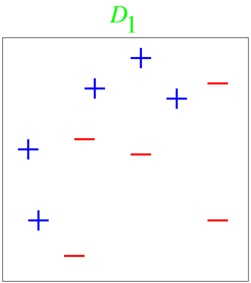
     $$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

     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

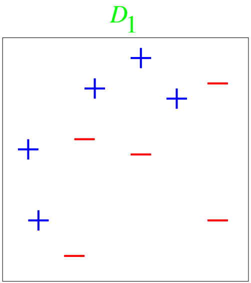
10 data points and 2 features



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

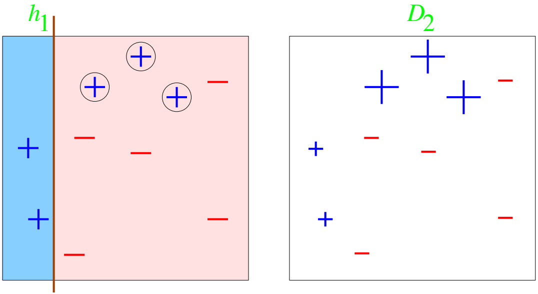
Example

10 data points and 2 features

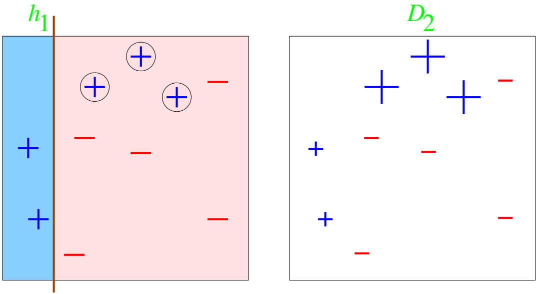


- The data points are clearly not linear separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: horizontal or vertical lines ('decision stumps')
  - Depth-1 decision trees, i.e., classify data based on a single attribute

Round 1: $t = 1$

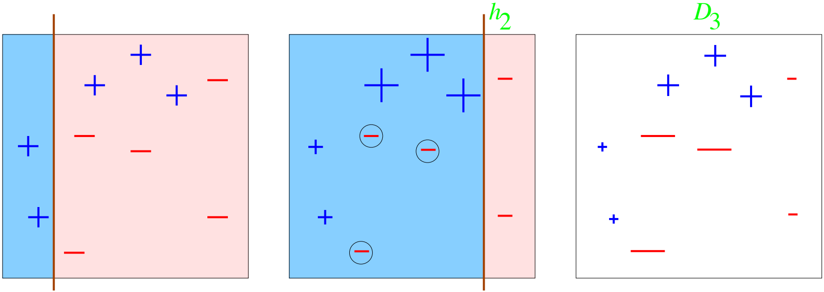


Round 1: $t = 1$

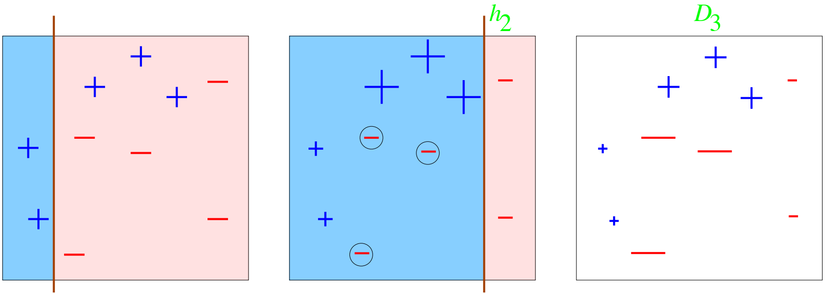


- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$.
- Weights recomputed; the 3 misclassified data points receive larger weights

Round 2: $t = 2$

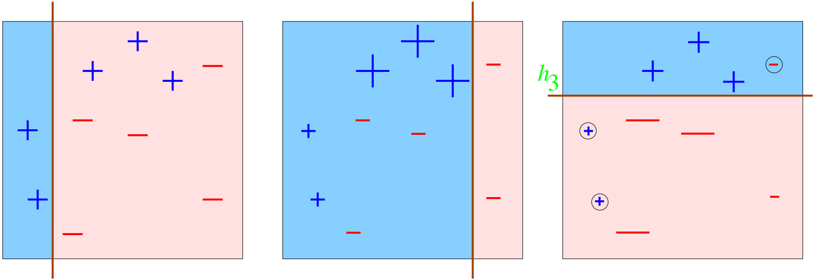


Round 2: $t = 2$

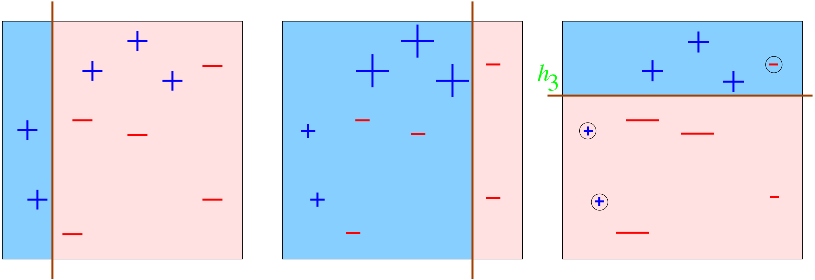


- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
  Note that $\epsilon_2 \neq 0.3$ as those 3 data points have weights less than $1/10$
- 3 misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights

Round 3: $t = 3$

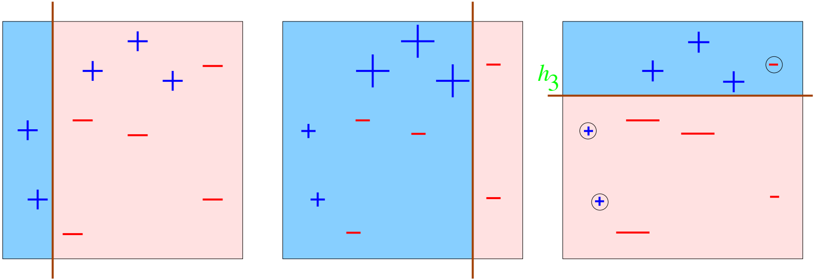


Round 3: $t = 3$



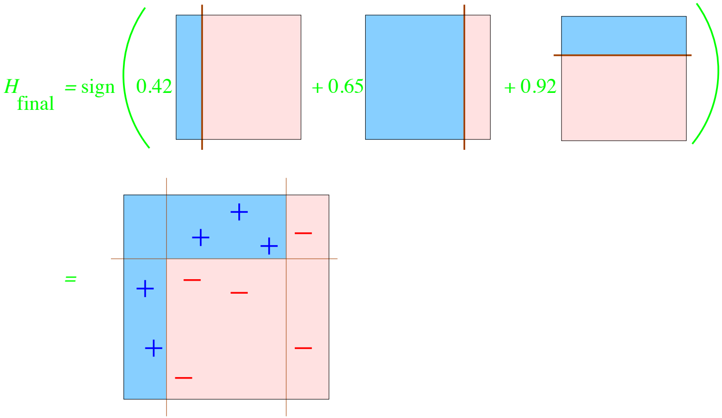
- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?

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 classification error.

Classification loss

- Suppose we want to have a classifier

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

- One seemingly natural loss function is 0-1 loss:

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

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

Surrogate loss

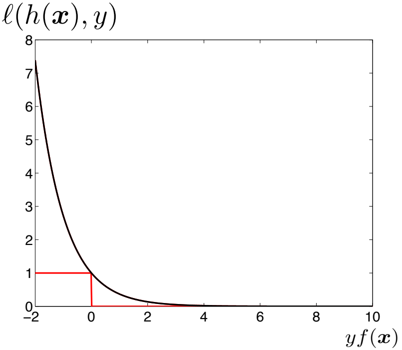
$0 - 1$ loss function $\ell(h(\boldsymbol{x}), y)$ is non-convex and difficult to optimize.
We can instead use a surrogate loss – what are examples?

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

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



Choosing the t-th classifier

Suppose a classifier $f_{t-1}(\boldsymbol{x})$, and want to add a weak learner $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

note: $h_t(\cdot)$ outputs -1 or 1, as does $\text{sign}\left[f_{t-1}(\cdot)\right]$

Choosing the t-th classifier

Suppose a classifier $f_{t-1}(\boldsymbol{x})$, and want to add a weak learner $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

note: $h_t(\cdot)$ outputs -1 or 1, as does $\text{sign}\left[f_{t-1}(\cdot)\right]$

How can we 'optimally' choose $h_t(\boldsymbol{x})$ and combination coefficient β_t?
Adaboost greedily *minimizes the exponential loss function*!

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

Choosing the t-th classifier

Suppose a classifier $f_{t-1}(\boldsymbol{x})$, and want to add a weak learner $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

note: $h_t(\cdot)$ outputs -1 or 1, as does sign $[f_{t-1}(\cdot)]$

How can we 'optimally' choose $h_t(\boldsymbol{x})$ and combination coefficient β_t? Adaboost greedily *minimizes the exponential loss function*!

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

Choosing the t-th classifier

Suppose a classifier $f_{t-1}(\boldsymbol{x})$, and want to add a weak learner $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

note: $h_t(\cdot)$ outputs -1 or 1, as does $\text{sign}\left[f_{t-1}(\cdot)\right]$

How can we 'optimally' choose $h_t(\boldsymbol{x})$ and combination coefficient β_t? Adaboost greedily *minimizes the exponential loss function*!

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

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

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

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

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

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

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

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?

How to choose β_t?

Summary

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

What term(s) must we optimize?

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?

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:

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

Updating the weights

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

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

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

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

Updating the weights

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

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

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

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

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

Updating the weights

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

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

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

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

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

Meta-Algorithm

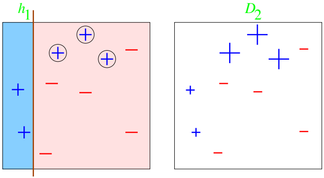
Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\boldsymbol{x})$ as long as it minimizes the weighted classification error

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

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

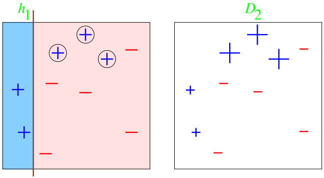
E.g., Decision Stumps

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



E.g., Decision Stumps

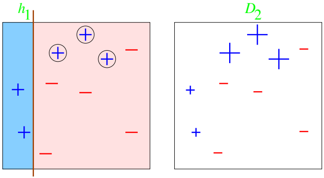
How do we choose the decision stump classifier given the weights at the second round of the following distribution?



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?

E.g., 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 $\textsc{sign}[h_1(\boldsymbol{x})]$, $\textsc{sign}[h_2(\boldsymbol{x})]$,$\cdots$,
- Combine into a linear classification model

$$y = \textsc{sign}\left\{\sum_t \beta_t \textsc{sign}[h_t(\boldsymbol{x})]\right\} = \textsc{sign}\left\{\boldsymbol{\beta}^\top \boldsymbol{\phi}(\boldsymbol{x})\right\}$$

Interpreting boosting as learning nonlinear basis

Two-stage process

- Get $\text{SIGN}[h_1(\boldsymbol{x})]$, $\text{SIGN}[h_2(\boldsymbol{x})]$, $\cdots$,
- Combine into a linear classification model

$$y = \text{SIGN}\left\{\sum_t \beta_t \text{SIGN}[h_t(\boldsymbol{x})]\right\} = \text{SIGN}\left\{\boldsymbol{\beta}^\top \boldsymbol{\phi}(\boldsymbol{x})\right\}$$

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

- 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 next class