Outline

1. Administration
2. Review of last lecture
3. Geometric Understanding of SVM
4. Boosting
Quiz 1

Tuesday Oct 21 6-7:20pm, THH 301

Please arrive on time
Lecture schedule

Oct 15 or Oct 16

TA will lead on *Pragmatics*

Oct 20 or Oct 21

No lecture in the scheduled time — prepare for the quiz
## Homework #1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>136</td>
</tr>
<tr>
<td>Min</td>
<td>23.00</td>
</tr>
<tr>
<td>Max</td>
<td>100.00</td>
</tr>
<tr>
<td>Average</td>
<td>88.24</td>
</tr>
<tr>
<td>Median</td>
<td>92.00</td>
</tr>
<tr>
<td>STD</td>
<td>13.28</td>
</tr>
</tbody>
</table>
Outline

1. Administration

2. Review of last lecture
   - Support vector machines
   - Basic Lagrange duality theory

3. Geometric Understanding of SVM

4. Boosting
Support vector machines

**Hinge loss** Assuming the label $y \in \{-1, 1\}$ and the decision rule is $h(x) = \text{SIGN}(f(x))$ with $f(x) = w^T \phi(x) + b$,

$$
\ell_{\text{HINGE}}(f(x), y) = \begin{cases} 
0 & \text{if } yf(x) \geq 1 \\
1 - yf(x) & \text{otherwise}
\end{cases}
$$

or $\ell_{\text{HINGE}}(f(x), y) = \max(0, 1 - yf(x))$

**Intuition:** penalize more if incorrectly classified (the left branch to the kink point)
Primal formulation of support vector machines (SVM)

Minimizing the total hinge loss on all the training data

\[
\min_{w,b} \sum_n \max(0, 1 - y_n [w^T \phi(x_n) + b]) + \frac{\lambda}{2} \|w\|^2_2
\]

equivalently,

\[
\min_{w,b,\{\xi_n\}} C \sum_n \xi_n + \frac{1}{2} \|w\|^2_2
\]

s.t. \[1 - y_n [w^T \phi(x_n) + b] \leq \xi_n, \quad \forall \ n\]
\[
\xi_n \geq 0, \quad \forall \ n
\]
Basic Lagrange duality theory

**Key concepts you should know**

- What do “primal” and “dual” mean?
- How SVM exploits dual formulation, thus results in using kernel functions for nonlinear classification
- What do support vectors mean?

**Our roadmap**

- We will tell you what dual looks like
- We will show you how it is derived
Derivation of the dual

We will derive the dual formulation as the process will reveal some interesting and important properties of SVM. Particularly, why is it called “support vector”?

**Recipe**

- Formulate a Lagrangian function that incorporates the constraints, thru introducing dual variables
- Minimize the Lagrangian function to solve the primal variables
- Put the primal variables into the Lagrangian and express in terms of dual variables
- Maximize the Lagrangian with respect to dual variables
- Recover the solution (for the primal variables) from the dual variables
Deriving the dual for SVM

**Lagrangian**

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

under the constraint that \( \alpha_n \geq 0 \) and \( \lambda_n \geq 0 \).
Minimizing the Lagrangian

Taking derivatives with respect to the primal variables

$$\frac{\partial L}{\partial w} = w - \sum_n y_n \alpha_n \phi(x_n) = 0$$
Minimizing the Lagrangian

**Taking derivatives with respect to the primal variables**

\[
\frac{\partial L}{\partial w} = w - \sum_n y_n \alpha_n \phi(x_n) = 0
\]

\[
\frac{\partial L}{\partial b} = \sum_n \alpha_n y_n = 0
\]
Minimizing the Lagrangian

Taking derivatives with respect to the primal variables

\[
\frac{\partial L}{\partial w} = w - \sum_{n} y_n \alpha_n \phi(x_n) = 0
\]

\[
\frac{\partial L}{\partial b} = \sum_{n} \alpha_n y_n = 0
\]

\[
\frac{\partial L}{\partial \xi_n} = C - \lambda_n - \alpha_n = 0
\]
Minimizing the Lagrangian

Taking derivatives with respect to the primal variables

\[
\frac{\partial L}{\partial w} = w - \sum_n y_n \alpha_n \phi(x_n) = 0
\]

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

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

This gives rise to equations linking the primal variables and the dual variables as well as new constraints on the dual variables:

\[
w = \sum_n y_n \alpha_n \phi(x_n)
\]

\[
\sum_n \alpha_n y_n = 0
\]

\[
C - \lambda_n - \alpha_n = 0
\]
Rewrite the Lagrange in terms of dual variables

Substitute the solution to the primal back into the Lagrangian

\[ g(\{\alpha_n\}, \{\lambda_n\}) = L(\mathbf{w}, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) \]
Rewrite the Lagrange in terms of dual variables

Substitute the solution to the primal back into the Lagrangian

\[
g(\{\alpha_n\}, \{\lambda_n\}) = L(w, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\})
\]

\[
= \sum_n (C - \alpha_n - \lambda_n) \xi_n + \frac{1}{2} \left\| \sum_n y_n \alpha_n \phi(x_n) \right\|_2^2 + \sum_n \alpha_n
\]

\[
+ \left( \sum_n \alpha_n y_n \right) b - \sum_n \alpha_n y_n \left( \sum_m y_m \alpha_m \phi(x_m) \right)^T \phi(x_n)
\]
Rewrite the Lagrange in terms of dual variables

Substitute the solution to the primal back into the Lagrangian

\[
g(\{\alpha_n\}, \{\lambda_n\}) = L(w, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\})
\]

\[
= \sum_n (C - \alpha_n - \lambda_n) \xi_n + \frac{1}{2} \| \sum_n y_n \alpha_n \phi(x_n) \|_2^2 + \sum_n \alpha_n
\]

\[
+ \left( \sum_n \alpha_n y_n \right) b - \sum_n \alpha_n y_n \left( \sum_m y_m \alpha_m \phi(x_m) \right)^T \phi(x_n)
\]

\[
= \sum_n \alpha_n + \frac{1}{2} \| \sum_n y_n \alpha_n \phi(x_n) \|_2^2 - \sum_{m,n} \alpha_n \alpha_m y_m y_n \phi(x_m)^T \phi(x_n)
\]
Rewrite the Lagrange in terms of dual variables

Substitute the solution to the primal back into the Lagrangian

\[
g(\{\alpha_n\}, \{\lambda_n\}) = L(w, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\})
\]

\[
= \sum_n (C - \alpha_n - \lambda_n)\xi_n + \frac{1}{2} \| \sum_n y_n \alpha_n \phi(x_n) \|^2_2 + \sum_n \alpha_n
\]

\[
+ \left( \sum_n \alpha_n y_n \right) b - \sum_n \alpha_n y_n \left( \sum_m y_m \alpha_m \phi(x_m) \right)^T \phi(x_n)
\]

\[
= \sum_n \alpha_n + \frac{1}{2} \| \sum_n y_n \alpha_n \phi(x_n) \|^2_2 - \sum_{m,n} \alpha_n \alpha_m y_m y_n \phi(x_m)^T \phi(x_n)
\]

\[
= \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} \alpha_n \alpha_m y_m y_n \phi(x_m)^T \phi(x_n)
\]

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

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

s.t.  \( \alpha_n \geq 0, \quad \forall \ n \)

\( \sum_n \alpha_n y_n = 0 \)

\( C - \lambda_n - \alpha_n = 0, \quad \forall \ n \)

\( \lambda_n \geq 0, \quad \forall \ n \)
The dual problem

Maximizing the dual under the constraints

\[
\max_{\alpha} \quad g(\{\alpha_n\}, \{\lambda_n\}) = \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(x_m, x_n)
\]

s.t. \( \alpha_n \geq 0, \quad \forall \ n \)
\( \sum_n \alpha_n y_n = 0 \)
\( C - \lambda_n - \alpha_n = 0, \quad \forall \ n \)
\( \lambda_n \geq 0, \quad \forall \ n \)

We can simplify as the objective function does not depend on \( \lambda_n \), thus we can convert the equality constraint involving \( \lambda_n \) with an inequality constraint on \( \alpha_n \leq C \):

\[
\alpha_n \leq C \leftrightarrow \lambda_n = C - \alpha_n \geq 0 \leftrightarrow C - \lambda_n - \alpha_n = 0, \lambda_n \geq 0
\]
Final form

\[
\max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^T \phi(x_n)
\]

s.t. \[0 \leq \alpha_n \leq C, \quad \forall \ n\]
\[
\sum_n \alpha_n y_n = 0
\]
Recover the solution

The primal variable $w$ is identified as

$$w = \sum_n \alpha_n y_n \phi(x_n)$$
Recover the solution

The primal variable $w$ is identified as

$$w = \sum_n \alpha_n y_n \phi(x_n)$$

To identify $b$, we need something else.
Complementary slackness and support vectors

At the optimal solution to both primal and dual, the following must be satisfied for every inequality constraint (these are called KKT conditions)

\[
\lambda_n \xi_n = 0 \\
\alpha_n \{1 - \xi_n - y_n [w^T \phi(x_n) + b]\} = 0
\]
Complementary slackness and support vectors

At the optimal solution to both primal and dual, the following must be satisfied for every inequality constraint (these are called KKT conditions)

\[ \lambda_n \xi_n = 0 \]
\[ \alpha_n \{1 - \xi_n - y_n[w^T \phi(x_n) + b]\} = 0 \]

From the first condition, if \( \alpha_n < C \), then
\[ \lambda_n = C - \alpha_n > 0 \rightarrow \xi_n = 0 \]

Thus, in conjunction with the second condition, we know that, if \( C > \alpha_n > 0 \), then
\[ 1 - y_n[w^T \phi(x_n) + b] = 0 \rightarrow b = y_n - w^T \phi(x_n) \]
as \( y_n \in \{-1, 1\} \).

For those \( n \) whose \( \alpha_n > 0 \), we call such training samples as “support vectors”. (We will discuss their geometric interpretation later).
Dual formulation and kernelized SVM

**Dual is also a convex quadratic programming**

$$\max_\alpha \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^T \phi(x_n)$$

s.t. $0 \leq \alpha_n \leq C$, $\forall n$

$$\sum_n \alpha_n y_n = 0$$

**We replace the inner products $\phi(x_m)^T \phi(x_n)$ with a kernel function**

$$\max_\alpha \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(x_m, x_n)$$

s.t. $0 \leq \alpha_n \leq C$, $\forall n$

$$\sum_n \alpha_n y_n = 0$$
Review of last lecture

Basic Lagrange duality theory

Recovering solution to the primal formulation

Weights

\[ w = \sum_{n} y_n \alpha_n \phi(x_n) \quad \leftarrow \quad \text{Linear combination of the input features!} \]

\[ b \]

\[ b = [y_n - w^T \phi(x_n)] = [y_n - \sum_{m} y_m \alpha_m k(x_m, x_n)], \quad \text{for any } C > \alpha_n > 0 \]

Making prediction on a test point \( x \)

\[ h(x) = \text{SIGN}(w^T \phi(x) + b) = \text{SIGN}(\sum_{n} y_n \alpha_n k(x_n, x) + b) \]

Again, to make prediction, it suffices to know the kernel function.
Things you need to know about deriving the dual

Make sure you can follow the recipe

- Formulate a Lagrangian function that incorporates the constraints, thru introducing dual variables
- Minimize the Lagrangian function to solve the primal variables
- Put the primal variables into the Lagrangian and express in terms of dual variables
- Maximize the Lagrangian with respect to dual variables
- Recover the solution (for the primal variables) from the dual variables
Outline

1. Administration
2. Review of last lecture
3. Geometric Understanding of SVM
4. Boosting
Intuition: where to put the decision boundary?

Consider the binary classification in the following figure. We have assumed, for convenience, that the *training* dataset is separable — there is a decision boundary that separates the two classes perfectly.

There are *infinite* many ways of putting the decision boundary $\mathcal{H} : \mathbf{w}^T \phi(x) + b = 0$! Our intuition is, however, to put the decision boundary to be in the middle of the two classes as much as possible. In other words, we want the decision boundary is to be far to every point as much as possible as long as the decision boundary classifies every point correctly.
Distances

The distance from a point $\phi(x)$ to the decision boundary is

$$d_H(\phi(x)) = \frac{|w^T \phi(x) + b|}{\|w\|_2}$$

(We have derived the above in the recitation/quiz0. Please re-verify it as a take-home exercise). We can remove the absolute $| \cdot |$ by exploiting the fact that the decision boundary classifies every point in the training dataset correctly. Namely, $(w^T \phi(x) + b)$ and $x$’s label $y$ are of the same sign. The distance is now,

$$d_H(\phi(x)) = \frac{y[w^T \phi(x) + b]}{\|w\|_2}$$
Maximizing margin

**Margin** The margin is defined as the smallest distance from all the training points

\[
\text{MARGIN} = \min_n \frac{y_n [\mathbf{w}^T \phi(\mathbf{x}_n) + b]}{\|\mathbf{w}\|_2}
\]
Maximizing margin

**Margin** The margin is defined as the smallest distance from all the training points.

\[
\text{MARGIN} = \min_n \frac{y_n[w^T \phi(x_n) + b]}{\|w\|_2}
\]

Since we are interested in finding a \( w \) to put all points *as distant as possible* from the decision boundary, we maximize the margin:

\[
\max_w \min_n \frac{y_n[w^T \phi(x_n) + b]}{\|w\|} = \max_w \frac{1}{\|w\|_2} \min_n y_n[w^T \phi(x_n) + b]
\]
Rescaled margin

Since the margin does not change if we scale \((w, b)\) by a constant factor \(c\) (as \(w^T \phi(x) + b = 0\) and \((cw)^T \phi(x) + (cb) = 0\) are the same decision boundary), we fix the scale by forcing

\[
\min_n y_n [w^T \phi(x_n) + b] = 1
\]
Rescaled margin

Since the MARGIN does not change if we scale \((w, b)\) by a constant factor \(c\) (as \(w^T \phi(x) + b = 0\) and \((cw)^T \phi(x) + (cb) = 0\) are the same decision boundary), we fix the scale by forcing

\[
\min_n y_n [w^T \phi(x_n) + b] = 1
\]

In this case, our margin becomes

\[
\text{MARGIN} = \frac{1}{\|w\|_2}
\]

precisely, the closest point to the decision boundary has a distance of that.
Primal formulation

Combining everything we have, for a separable training dataset, we aim to

$$\max_w \frac{1}{\|w\|_2^2} \quad \text{such that} \quad y_n [w^T \phi(x_n) + b] \geq 1, \quad \forall \ n$$

This is equivalent to

$$\min_w \frac{1}{2} \|w\|_2^2 \quad \text{s.t.} \quad y_n [w^T \phi(x_n) + b] \geq 1, \quad \forall \ n$$

This starts to look like our first formulation for SVMs. For this geometric intuition, SVM is called *max margin* (or large margin) classifier. The constraints are called *large margin constraints.*
SVM for non-separable data

Suppose there are training data points that cannot be classified correctly no matter how we choose $w$. For those data points,

$$y_n[w^T \phi(x_n) + b] \leq 0$$

for any $w$. Thus, the previous constraint

$$y_n[w^T \phi(x_n) + b] \geq 1, \quad \forall \ n$$

is no longer feasible.
Suppose there are training data points that cannot be classified correctly no matter how we choose $\mathbf{w}$. For those data points,

$$y_n[\mathbf{w}^T \phi(\mathbf{x}_n) + b] \leq 0$$

for any $\mathbf{w}$. Thus, the previous constraint

$$y_n[\mathbf{w}^T \phi(\mathbf{x}_n) + b] \geq 1, \quad \forall \ n$$

is no longer **feasible**. To deal with this issue, we introduce *slack variables* $\xi_n$ to help

$$y_n[\mathbf{w}^T \phi(\mathbf{x}_n) + b] \geq 1 - \xi_n, \quad \forall \ n$$

where we also require $\xi_n \geq 0$. Note that, even for “hard” points that cannot be classified correctly, the slack variable will be able to make them satisfy the above constraint (we can keep increasing $\xi_n$ until the above inequality is met.)
SVM Primal formulation with slack variables

We obviously do not want $\xi_n$ goes to infinity, so we balance their sizes by penalizing them toward zero as much as possible

$$\min_w \frac{1}{2} \|w\|^2_2 + C \sum_n \xi_n$$

s.t. $y_n[w^T \phi(x_n) + b] \geq 1 - \xi_n$, $\forall \ n$

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

where $C$ is our tradeoff (hyper)parameter. This is precisely the primal formulation we first got for SVM.
Meaning of “support vectors” in SVMs

**Complementary slackness** At optimum, we have to have

\[ \alpha_n \{ 1 - \xi_n - y_n \left[ w^T \phi(x_n) + b \right] \} = 0, \quad \forall \quad n \]

That means, for some \( n \), \( \alpha_n = 0 \). Additionally, our optimal solution is given by

\[ w = \sum_n \alpha_n y_n \phi(x_n) = \sum_{n: \alpha_n > 0} \alpha_n y_n \phi(x_n) \]

In words, our solution is only determined by those training samples whose corresponding \( \alpha_n \) is strictly positive. Those samples are called support vectors.

Non-support vectors whose \( \alpha_n = 0 \) can be removed by the training dataset — this removal will not affect the optimal solution (i.e., after the removal, if we construct another SVM classifier on the reduced dataset, the optimal solution is the same as the one on the original dataset.)
Who are support vectors?

**Case analysis** Since, we have

\[ 1 - \xi_n - y_n [w^T \phi(x_n) + b] \} = 0 \]

We have

- \( \xi_n = 0 \). This implies \( y_n [w^T \phi(x_n) + b] = 1 \). They are on points that are \( 1/\|w\|_2 \) 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 those being circled with the orange line.
Outline

1. Administration

2. Review of last lecture

3. Geometric Understanding of SVM

4. Boosting
   - AdaBoost
   - Derivation of AdaBoost
   - Boosting as learning nonlinear basis
Boosting

**High-level idea**: combine a lot of classifiers
- Sequentially construct those classifiers one at a time
- Use *weak* classifiers to arrive at complex decision boundaries

**Our plan**
- Describe AdaBoost algorithm
- Derive the algorithm
How Boosting algorithm works?

- Given: \( N \) samples \( \{x_n, y_n\} \), where \( y_n \in \{+1, -1\} \), and some ways 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 \( h_t(x) \) based on the current weight \( w_t(n) \), by minimizing the weighted classification error
     \[
     \epsilon_t = \sum_n w_t(n) I[y_n \neq h_t(x_n)]
     \]
  2. Calculate weights for combining classifiers \( \beta_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \)
  3. Update weights
     \[
     w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(x_n)}
     \]
     and normalize them such that \( \sum_n w_{t+1}(n) = 1 \).
- Output the final classifier
  \[
  h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right]
  \]
Example

10 data points

- Base classifier $h(\cdot)$: either horizontal or vertical lines (these are called decision stumps, classifying data based on a single attribute)
- The data points are clearly not linear separable.
- In the beginning, all data points have equal weights (the size of the data markers “+” or “-”)

$$D_1$$
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: $t = 2$

- 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 $1/10$.
- Weights recomputed; the 3 misclassified data points receive larger weights. Note that the data points classified correctly on round $t = 1$ receive much smaller weights as they have been consistently classified correctly.
3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.

Note that those previously correctly classified data points are now misclassified — however, we might be lucky on this as if they have been consistently classified correctly, then this round’s mistake is probably not a big deal.
Final classifier: combining 3 classifiers

\[ H_{\text{final}} = \text{sign} \left( 0.42 + 0.65 + 0.92 \right) \]

- all data points are now classified correctly!
Why AdaBoost works?

We will show next that it minimizes a loss function related to classification error.

**Classification loss**

- Suppose we want to have a classifier

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

- our loss function is thus

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

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

The previous loss function $\ell(h(x), y)$ is difficult to optimize. Instead, we will use the following loss function

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

This loss function will function as a surrogate to the true loss function $\ell(h(x, y))$. However, $\ell^{\text{EXP}}(h(x), y)$ is easier to handle numerically as it is differentiable, see below the contrast between the red and black curves.
Choosing the $t$-th classifier

Suppose we have built a classifier $f_{t-1}(x)$, and we want to improve it by adding a new classifier $h_t(x)$ to construct a new classifier

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

how can we choose optimally the new classifier $h_t(x)$ and the combination coefficient $\beta_t$? The strategy we will use is to greedily minimize the exponential loss function.

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

$$= \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_n [f_{t-1}(x_n) + \beta_t h_t(x_n)]}$$

Choosing the $t$-th classifier

Suppose we have built a classifier $f_{t-1}(x)$, and we want to improve it by adding a new classifier $h_t(x)$ to construct a new classifier

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

how can we choose optimally the new classifier $h_t(x)$ and the combination coefficient $\beta_t$? The strategy we will use is to greedily minimize the exponential loss function.

$$(h^*_t(x), \beta^*_t) = \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_n f(x_n)}$$

$$= \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_n [f_{t-1}(x_n) + \beta_t h_t(x_n)]}$$

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

where we have used $w_t(n)$ as a shorthand for $e^{-y_n f_{t-1}(x_n)}$
The new classifier

We decompose the *weighted* loss function (by $w_t(n)$) into two parts

$$\sum_n w_t(n)e^{-y_n\beta_t h_t(x_n)}$$

$$= \sum_n w_t(n)e^{\beta_t I[y_n \neq h_t(x_n)]} + \sum_n w_t(n)e^{-\beta_t I[y_n = h_t(x_n)]}$$
The new classifier

We decompose the \textit{weighted} loss function (by $w_t(n)$) into two parts

$$
\sum_{n} w_t(n) e^{-y_n \beta_t h_t(x_n)}
= \sum_{n} w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(x_n)] + \sum_{n} w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(x_n)]
= \sum_{n} w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(x_n)] + \sum_{n} w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(x_n)])
$$
The new classifier

We decompose the \textit{weighted} loss function (by $w_t(n)$) into two parts

$$
\sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}
$$

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

$$
= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(x_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(x_n)])
$$

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

We have used the following properties to derive the above

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

Minimizing the weighted classification error

Thus, we would want to choose $h_t(x_n)$ such that

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

Namely, the weighted classification error is minimized — precisely \textit{train a weak classifier based on the current weight $w_t(n)$ on the slide} \textbf{How Boosting algorithm works?}.
Minimizing the weighted classification error

Thus, we would want to choose \( h_t(x_n) \) such that

\[
h^*_t(x) = \arg \min_{h_t(x)} \epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(x_n)]
\]

Namely, the weighted classification error is minimized — precisely \textit{train a weak classifier based on the current weight \( w_t(n) \) on the slide How Boosting algorithm works?}.

**Remarks** We can safely assume that \( w_t(x_n) \) is normalized so that \( \sum_n w_t(x_n) = 1 \). This normalization requirement can be easily maintained by changing the weights to

\[
w_t(x_n) \leftarrow \frac{w_t(x_n)}{\sum_{n'} w_t(x_{n'})}
\]

This change \textit{does not} affect how to choose \( h^*_t(x) \), as the term \( \sum_{n'} w_t(x_{n'}) \) is a constant with respect to \( n \).
How to choose $\beta_t$?

We will select $\beta_t$ to minimize

$$\sum_n w_t(n) I[y_n \neq h_t(x_n)] + e^{-\beta_t} \sum_n w_t(n)$$

We assume $\sum_n w_t(n)$ is now 1 (cf. the previous slide’s Remarks). We take derivative with respect to $\beta_t$ and set to zero, and derive the optimal $\beta_t$ as

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

which is precisely what is on the slide How Boosting algorithm works?

Take-home exercise. Verify the solution
Updating the weights

Now that we have improved our classifier into

\[ f(x) = f_{t-1}(x) + \beta^* h^*_t(x) \]

At the \( t \)-th iteration, we will need to compute the weights for the above classifier, which is,

\[ w_{t+1}(n) = e^{-y_n f(x_n)} = e^{-y_n [f_{t-1}(x) + \beta^* h^*_t(x_n)]} \]
Updating the weights

Now that we have improved our classifier into

\[ f(\mathbf{x}) = f_{t-1}(\mathbf{x}) + \beta_t^* h_t^*(\mathbf{x}) \]

At the \( t \)-th iteration, we will need to compute the weights for the above classifier, which is,

\[
w_{t+1}(n) = e^{-y_nf(x_n)} = e^{-y_n[f_{t-1}(x) + \beta_t^* h_t^*(x_n)]} = w_t(n)e^{-y_n\beta_t^* h_t^*(x_n)}
\]
Updating the weights

Now that we have improved our classifier into

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

At the \( t \)-th iteration, we will need to compute the weights for the above classifier, which is,

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

\[ = w_t(n) e^{-y_n \beta_t h_t^*(x_n)} = \begin{cases} w_t(n) e^{\beta_t} & \text{if } y_n \neq h_t^*(x_n) \\ w_t(n) e^{-\beta_t} & \text{if } y_n = h_t^*(x_n) \end{cases} \]
Updating the weights

Now that we have improved our classifier into

$$f(x) = f_{t-1}(x) + \beta^*_t h^*_t(x)$$

At the $t$-th iteration, we will need to compute the weights for the above classifier, which is,

$$w_{t+1}(n) = e^{-y_n f(x_n)} = e^{-y_n [f_{t-1}(x) + \beta^*_t h^*_t(x_n)]}$$

$$= w_t(n) e^{-y_n \beta^*_t h^*_t(x_n)} = \begin{cases} w_t(n) e^{\beta^*_t} & \text{if } y_n \neq h^*_t(x_n) \\ w_t(n) e^{-\beta^*_t} & \text{if } y_n = h^*_t(x_n) \end{cases}$$

**Remarks** The key point is that the misclassified data point will get its weight increased, while the correctly data point will get its weight decreased.
Remarks

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

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

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any classifier where we can do the above.
Remarks

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

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

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any classifier where we can do the above.

**Ex.** 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!
Nonlinear basis learned by boosting

Two-stage process

- Get \( \text{SIGN}[f_1(x)], \text{SIGN}[f_2(x)], \cdots, \)
- Combine into a linear classification model

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

Equivalently, each stage learns a nonlinear basis \( \phi_t(x) = \text{SIGN}[f_t(x)] \).

This relates to neural networks, which we might discuss next week.