Ensemble methods

CS 446 / ECE 449

2021-03-02 17:33:49 -0600 (80e094f)
“pytorch meta-algorithm”.

6. Tweak 1-5, possibly reducing model complexity, until testing error is small.

Rather than trying many models and only selecting one, can we combine them into an “ensemble” and do better?
Plan for today

▶ Bagging.
▶ Boosting.
This first approach is based upon a simple idea:

- If the predictors have independent errors, a majority vote of their outputs should be good.

Let’s first check this majority vote claim.
Suppose we have $n$ classifiers.
Suppose each is wrong independently with probability 0.4.
Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$).
Combining classifiers

Suppose we have $n$ classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $E(Z_i) = 0.4$). We can model the distribution of errors with Binom($n$, 0.4).
Combining classifiers

Suppose we have $n$ classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$). We can model the distribution of errors with $\text{Binom}(n, 0.4)$. 

![Histogram showing distribution of errors for $n=10$. The green region represents at least half classifiers wrong, corresponding to the error of majority vote. The red region corresponds to all classifiers wrong. The probability of at least half classifiers wrong is significantly less than 0.4, indicating the benefit of combining classifiers.]
Suppose we have \( n \) classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables \((Z_i)_{i=1}^n\) (thus \(E(Z_i) = 0.4\)). We can model the distribution of errors with \(\text{Binom}(n, 0.4)\).
Combining classifiers

Suppose we have $n$ classifiers.
Suppose each is wrong independently with probability 0.4.
Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$).
We can model the distribution of errors with Binom$(n, 0.4)$.

![Histogram showing distribution of errors for #classifiers = n = 30]
Suppose we have $n$ classifiers. Suppose each is wrong independently with probability 0.4.
Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$).
We can model the distribution of errors with $\text{Binom}(n, 0.4)$.
Suppose we have \( n \) classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables \((Z_i)_{i=1}^n\) (thus \( \mathbb{E}(Z_i) = 0.4 \)). We can model the distribution of errors with Binom\((n, 0.4)\).
Suppose we have \( n \) classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables \((Z_i)_{i=1}^n\) (thus \(\mathbb{E}(Z_i) = 0.4\)). We can model the distribution of errors with \(\text{Binom}(n, 0.4)\).
Suppose we have $n$ classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$). We can model the distribution of errors with $\text{Binom}(n, 0.4)$. **Red:** all classifiers wrong.
Combining classifiers

Suppose we have \( n \) classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables \((Z_i)_{i=1}^{n}\) (thus \(\mathbb{E}(Z_i) = 0.4\)). We can model the distribution of errors with Binom\((n, 0.4)\).

**Red:** all classifiers wrong.
Suppose we have \( n \) classifiers.
Suppose each is wrong independently with probability 0.4.  
Model classifier errors as random variables \( (Z_i)_{i=1}^n \) (thus \( \mathbb{E}(Z_i) = 0.4 \)).
We can model the distribution of errors with Binom\( (n, 0.4) \).
**Red:** all classifiers wrong.
Combining classifiers

Suppose we have \( n \) classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables \((Z_i)_{i=1}^{n}\) (thus \(\mathbb{E}(Z_i) = 0.4\)). We can model the distribution of errors with Binom(\(n, 0.4\)).

**Red:** all classifiers wrong.

![Graph showing the distribution of errors with red indicating all classifiers wrong.](image-url)
Suppose we have $n$ classifiers.
Suppose each is wrong independently with probability $0.4$.
Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $E(Z_i) = 0.4$).
We can model the distribution of errors with $\text{Binom}(n, 0.4)$.
**Red:** all classifiers wrong.
Combining classifiers

Suppose we have $n$ classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $E(Z_i) = 0.4$). We can model the distribution of errors with Binom($n$, 0.4).

**Red:** all classifiers wrong.

**Green:** at least half classifiers wrong.
Combining classifiers

Suppose we have $n$ classifiers.
Suppose each is wrong independently with probability 0.4.
Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$).
We can model the distribution of errors with $\text{Binom}(n, 0.4)$.

Green: at least half classifiers wrong.
Combining classifiers

Suppose we have $n$ classifiers.
Suppose each is wrong independently with probability 0.4.
Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$).
We can model the distribution of errors with $\text{Binom}(n, 0.4)$.

**Green:** at least half classifiers wrong.

![Histogram showing the distribution of classifier errors with green region indicating at least half classifiers wrong.](image)
Combining classifiers

Suppose we have $n$ classifiers. Suppose each is wrong independently with probability 0.4. 
Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$). We can model the distribution of errors with Binom($n$, 0.4).  

**Green:** at least half classifiers wrong.
Combining classifiers

Suppose we have $n$ classifiers. Suppose each is wrong independently with probability 0.4. Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $\mathbb{E}(Z_i) = 0.4$). We can model the distribution of errors with $\text{Binom}(n, 0.4)$. 

**Green**: at least half classifiers wrong.
Combining classifiers

Suppose we have $n$ classifiers.
Suppose each is wrong independently with probability 0.4.
Model classifier errors as random variables $(Z_i)_{i=1}^n$ (thus $E(Z_i) = 0.4$).
We can model the distribution of errors with Binom($n, 0.4$).

**Green:** at least half classifiers wrong.
Combining classifiers

Suppose we have \( n \) classifiers.
Suppose each is wrong independently with probability 0.4.
Model classifier errors as random variables \((Z_i)_{i=1}^n\) (thus \(\mathbb{E}(Z_i) = 0.4\)).
We can model the distribution of errors with Binom\((n, 0.4)\).

**Green:** at least half classifiers wrong.
Combining classifiers

Suppose we have \( n \) classifiers.
Suppose each is wrong independently with probability 0.4.
Model classifier errors as random variables \((Z_i)^n_{i=1}\) (thus \( E(Z_i) = 0.4 \)).
We can model the distribution of errors with Binom \((n, 0.4)\).  
**Green**: at least half classifiers wrong.

Green region is error of majority vote! 0.075 \( \ll \) 0.4 !!!
**Majority vote**

**Green region** is error of majority vote!
Suppose $y_i \in \{-1, +1\}$.

$$\text{MAJ}(y_1, \ldots, y_n) := \begin{cases}  +1 & \text{when } \sum_i y_i \geq 0, \\  -1 & \text{when } \sum_i y_i < 0. \end{cases}$$

Error rate of majority classifier (with individual error probability $p$):

$$\Pr[\text{Binom}(n, p) \geq n/2] = \sum_{i=n/2}^{n} \binom{n}{i} p^i (1-p)^{n-i} \leq \exp \left( -2n(1/2 - p)^2 \right).$$
Green region is error of majority vote!
Suppose $y_i \in \{-1, +1\}$.

$$\text{MAJ}(y_1, \ldots, y_n) := \begin{cases} +1 & \text{when } \sum_i y_i \geq 0, \\ -1 & \text{when } \sum_i y_i < 0. \end{cases}$$

Error rate of majority classifier (with individual error probability $p$):

$$\Pr[\text{Binom}(n, p) \geq n/2] = \sum_{i=n/2}^{n} \binom{n}{i} p^i (1-p)^{n-i} \leq \exp\left(-2n\left(\frac{1}{2} - p\right)^2\right).$$
Green region is error of majority vote!
Suppose $y_i \in \{-1, +1\}$.

$$\text{MAJ}(y_1, \ldots, y_n) := \begin{cases} +1 & \text{when } \sum_i y_i \geq 0, \\ -1 & \text{when } \sum_i y_i < 0. \end{cases}$$

Error rate of majority classifier (with individual error probability $p$):

$$\Pr[\text{Binom}(n, p) \geq n/2] = \sum_{i=n/2}^{n} \binom{n}{i} p^i (1-p)^{n-i} \leq \exp \left(-2n(1/2 - p)^2\right).$$
Green region is error of majority vote!
Suppose \( y_i \in \{-1, +1\} \).

\[
\text{MAJ}(y_1, \ldots, y_n) := \begin{cases} 
+1 & \text{when } \sum_i y_i \geq 0, \\
-1 & \text{when } \sum_i y_i < 0. 
\end{cases}
\]

Error rate of majority classifier (with individual error probability \( p \)):

\[
\Pr[\text{Binom}(n, p) \geq n/2] = \sum_{i=n/2}^{n} \binom{n}{i} p^i (1-p)^{n-i} \leq \exp \left(-2n(1/2 - p)^2\right).
\]
Majority vote

\textbf{Green region} is error of majority vote!
Suppose \( y_i \in \{-1, +1\} \).

\[
\text{MAJ}(y_1, \ldots, y_n) := \begin{cases} 
+1 & \text{when } \sum_i y_i \geq 0, \\
-1 & \text{when } \sum_i y_i < 0.
\end{cases}
\]

Error rate of majority classifier (with individual error probability \( p \)):

\[
\Pr[\text{Binom}(n, p) \geq n/2] = \sum_{i=n/2}^{n} \binom{n}{i} p^i (1 - p)^{n-i} \leq \exp \left(-2n(1/2 - p)^2\right).
\]
Green region is error of majority vote!
Suppose \( y_i \in \{-1, +1\} \).

\[
\text{MAJ}(y_1, \ldots, y_n) := \begin{cases} 
+1 & \text{when } \sum_i y_i \geq 0, \\
-1 & \text{when } \sum_i y_i < 0.
\end{cases}
\]

Error rate of majority classifier (with individual error probability \( p \)):

\[
\Pr[\text{Binom}(n, p) \geq n/2] = \sum_{i=n/2}^{n} \binom{n}{i} p^i (1-p)^{n-i} \leq \exp \left(-2n(1/2 - p)^2\right).
\]
Green region is error of majority vote!

Error of majority vote classifier goes down exponentially in \( n \).

\[
\Pr[\text{Binom}(n, p) \geq n/2] = \sum_{i=n/2}^{n} \binom{n}{i} p^i (1 - p)^{n-i} \leq \exp \left(-2n(1/2 - p)^2\right).
\]
How to use independent errors in an algorithm?

1. For $t = 1, 2, \ldots, T$:
   1.1 Obtain IID data $S_t := ((x_i^{(t)}, y_i^{(t)}))_{i=1}^n$,
   1.2 Train classifier $f_t$ on $S_t$.

2. Output $\mathbf{x} \mapsto \text{MAJ}(f_1(\mathbf{x}), \ldots, f_T(\mathbf{x}))$. 

Good news: errors are independent! (Our exponential error estimate from before is valid.)

Bad news: classifiers trained on $1/T$ fraction of data (why not just train ResNet on all of it . . . ).
How to use independent errors in an algorithm?

1. For $t = 1, 2, \ldots, T$:
   1.1 Obtain IID data $S_t := ((x_i^{(t)}, y_i^{(t)}))_{i=1}^n$,
   1.2 Train classifier $f_t$ on $S_t$.

2. Output $x \mapsto \text{MAJ}(f_1(x), \ldots, f_T(x))$.

▶ **Good news:** errors are independent!
   (Our exponential error estimate from before is valid.)

▶ **Bad news:** classifiers trained on $\frac{1}{T}$ fraction of data
   (why not just train ResNet on all of it . . . ).
Bagging = **Bootstrap aggregating** (Leo Breiman, 1994).

1. Obtain IID data $S := ((x_i, y_i))_{i=1}^n$.

2. For $t = 1, 2, \ldots, T$:
   
   2.1 Resample $n$ points uniformly at random with replacement from $S$, obtaining "Bootstrap sample" $S_t$.
   
   2.2 Train classifier $f_t$ on $S_t$.

3. Output $x \mapsto \text{MAJ}(f_1(x), \ldots, f_T(x))$. 

▶ Good news: using most of the data for each $f_t$!

▶ Bad news: errors no longer independent...?
**Bagging**

Bagging = **Bootstrap aggregating** (Leo Breiman, 1994).

1. Obtain IID data \( S := ((x_i, y_i))_{i=1}^{n} \).
2. For \( t = 1, 2, \ldots, T \):
   2.1 Resample \( n \) points uniformly at random with replacement from \( S \), obtaining “Bootstrap sample” \( S_t \).
   2.2 Train classifier \( f_t \) on \( S_t \).
3. Output \( x \mapsto \text{MAJ}(f_1(x), \ldots, f_T(x)) \).

- **Good news:** using most of the data for each \( f_t \)!
- **Bad news:** errors no longer independent...?
Question:

Take $n$ samples uniformly at random with replacement from a population of size $n$. What is the probability that a given individual is not picked?

Answer:

\[
\left(1 - \frac{1}{n}\right)^n; \text{ for large } n: \lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = 1 \approx 0.3679.
\]

Implications for bagging:

▶ Each bootstrap sample contains about 63% of the data set.
▶ Remaining 37% can be used to estimate error rate of classifier trained on the bootstrap sample.
▶ If we have three classifiers, some of their error estimates must share examples! Independence is violated!
Question:

Take $n$ samples uniformly at random with replacement from a population of size $n$. What is the probability that a given individual is not picked?

Answer: $\left(1 - \frac{1}{n}\right)^n$; for large $n$: $\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.3679$. 

Implications for bagging:

- Each bootstrap sample contains about 63% of the data set.
- Remaining 37% can be used to estimate error rate of classifier trained on the bootstrap sample.
- If we have three classifiers, some of their error estimates must share examples! Independence is violated!
Question:

Take \( n \) samples uniformly at random with replacement from a population of size \( n \). What is the probability that a given individual is not picked?

Answer: \( \left( 1 - \frac{1}{n} \right)^n \); for large \( n \):

\[
\lim_{n \to \infty} \left( 1 - \frac{1}{n} \right)^n = \frac{1}{e} \approx 0.3679.
\]

Implications for bagging:

- Each bootstrap sample contains about 63% of the data set.
- Remaining 37% can be used to estimate error rate of classifier trained on the bootstrap sample.
- If we have three classifiers, some of their error estimates must share examples! Independence is violated!
Random Forests (Leo Breiman, 2001).

1. Obtain IID data $S := ((x_i, y_i))_{i=1}^n$.

2. For $t = 1, 2, \ldots, T$:
   2.1 Resample $n$ points uniformly at random with replacement from $S$, obtaining “Bootstrap sample” $S_t$.
   2.2 Train a decision tree $f_T$ on $S_t$ as follows: when greedily splitting tree nodes, consider only $\sqrt{d}$ and not $d$ possible features.

3. Output $x \mapsto \text{MAJ}(f_1(x), \ldots, f_T(x))$. 

▶ Heuristic news: maybe errors are more independent now?
Random Forests (Leo Breiman, 2001).

1. Obtain IID data $S := ((x_i, y_i))_{i=1}^n$.

2. For $t = 1, 2, \ldots, T$:
   
   2.1 Resample $n$ points uniformly at random with replacement from $S$, obtaining “Bootstrap sample” $S_t$.
   
   2.2 Train a decision tree $f_T$ on $S_t$ as follows:
       when greedily splitting tree nodes,
       consider only $\sqrt{d}$ and not $d$ possible features.

3. Output $x \mapsto \text{MAJ}(f_1(x), \ldots, f_T(x))$.

▶ Heuristic news: maybe errors are more independent now?
Boosting overview

We no longer assume classifiers have independent errors.

We no longer output a simple majority: we reweight the classifiers via optimization.

There is a rich theory with many interpretations.
Boosting overview

- We no longer assume classifiers have independent errors.
- We no longer output a simple majority: we reweight the classifiers via optimization.
- There is a rich theory with many interpretations.
Simplified boosting scheme

1. Start with data \(((x_i, y_i))_{i=1}^n\) and classifiers \((h_1, \ldots, h_T)\).
2. Find weights \(w \in \mathbb{R}^T\) which approximately minimize

\[
\frac{1}{n} \sum_{i=1}^{n} \ell \left( y_i \sum_{j=1}^{T} w_j h_j(x_i) \right) = \frac{1}{n} \sum_{i=1}^{n} \ell \left( y_i w^T z_i \right),
\]

where \(z_i = (h_1(x_i), \ldots, h_T(x_i)) \in \mathbb{R}^T\).

(We use classifiers to give us features.)

3. Predict with \(x \mapsto \sum_{j=1}^{T} w_j h_j(x)\).
Simplified boosting scheme

1. Start with data \( ((\mathbf{x}_i, y_i)^n_{i=1} \) and classifiers \((h_1, \ldots, h_T)\).

2. Find weights \( \mathbf{w} \in \mathbb{R}^T \) which approximately minimize

\[
\frac{1}{n} \sum_{i=1}^n \ell \left( y_i \sum_{j=1}^T w_j h_j(\mathbf{x}_i) \right) = \frac{1}{n} \sum_{i=1}^n \ell \left( y_i \mathbf{w}^T \mathbf{z}_i \right),
\]

where \( \mathbf{z}_i = (h_1(\mathbf{x}_i), \ldots, h_T(\mathbf{x}_i)) \in \mathbb{R}^T \).
(We use classifiers to give us features.)

3. Predict with \( \mathbf{x} \mapsto \sum_{j=1}^T w_j h_j(\mathbf{x}) \).

Remarks.

- If \( \ell \) is convex, this is standard linear prediction: convex in \( \mathbf{w} \).
- In the classical setting:
  \( \ell(r) = \exp(-r) \), optimizer = coordinate descent, \( T = \infty \).
- Most commonly, \((h_1, \ldots, h_T)\) are decision stumps.
- Popular software implementation: xgboost.
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width
Decision stumps?

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

$\hat{y} = 2$
Decision stumps?

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

Decision stumps?
Decision stumps?

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 = \text{ratio of sepal length to width}$
- $x_2 = \text{ratio of petal length to width}$

\[ x_1 > 1.7 \]
\[ \hat{y} = 1 \]
\[ \hat{y} = 3 \]

... and stop there!
Minimizing \( \frac{1}{n} \sum_{i=1}^{n} \ell \left( y_i \sum_{j=1}^{T} w_j h_j(x_i) \right) \) over \( w \in \mathbb{R}^T \), where \((h_1, \ldots, h_T)\) are decision stumps.
Minimizing \( \frac{1}{n} \sum_{i=1}^{n} \ell \left( y_i \sum_{j=1}^{T} w_j h_j(x_i) \right) \) over \( w \in \mathbb{R}^T \), where \((h_1, \ldots, h_T)\) are decision stumps.
Boosting decision stumps

Minimizing \( \frac{1}{n} \sum_{i=1}^{n} \ell \left( y_i \sum_{j=1}^{T} w_j h_j(x_i) \right) \) over \( w \in \mathbb{R}^T \), where \( (h_1, \ldots, h_T) \) are decision stumps.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Boosted stumps. (( \mathcal{O}(n) ) param.)</td>
<td>2-layer ReLU. (( \mathcal{O}(n) ) param.)</td>
<td>3-layer ReLU. (( \mathcal{O}(n) ) param.)</td>
</tr>
</tbody>
</table>
Boosting decision stumps

Minimizing \( \frac{1}{n} \sum_{i=1}^{n} \ell \left( y_i \sum_{j=1}^{T} w_j h_j(x_i) \right) \) over \( w \in \mathbb{R}^T \), where \((h_1, \ldots, h_T)\) are decision stumps.

Boosted stumps.  
\( (O(n) \text{ param.}) \)

2-layer ReLU.  
\( (O(n) \text{ param.}) \)

3-layer ReLU.  
\( (O(n) \text{ param.}) \)
Boosting decision stumps

Minimizing $\frac{1}{n} \sum_{i=1}^{n} \ell \left( y_i \sum_{j=1}^{T} w_j h_j(x_i) \right)$ over $w \in \mathbb{R}^T$, where $(h_1, \ldots, h_T)$ are decision stumps.

Boosted stumps.  
($O(n)$ param.)

2-layer ReLU.  
($O(n)$ param.)

3-layer ReLU.  
($O(n)$ param.)
The classical methods used coordinate descent:

- Find the maximum magnitude coordinate of the gradient:

\[
\arg \max_j \frac{d}{dw_j} \sum_{i=1}^n \ell \left( \sum_j w_j h_j(x_i) y_i \right)
\]

\[
= \arg \max_j \sum_{i=1}^n \ell' \left( \sum_j w_j h_j(x_i) y_i \right) h_j(x_i) y_i
\]

\[
= \arg \max_j \sum_{i=1}^n q_i h_j(x_i) y_i,
\]

where we’ve defined \(q_i := \ell' \left( \sum_j h_j(x_i) y_i \right)\).

- Iterate: \(w' := w - \eta s e_j\), where \(j\) is the maximum coordinate, \(s \in \{-1, +1\}\) is its sign, and \(\eta\) is a step size.
Interpreting coordinate descent

Suppose $h_j : \mathbb{R}^d \to \{-1, +1\}$; then $h_j(x)y = 2 \cdot 1[h_j(x) = y] - 1$, and each step solves

$$
\arg \max_j \left| \sum_{i=1}^n q_i h_j(x_i) y_i \right| = \arg \max_j \left| \sum_{i=1}^n q_i \left( 1[h_j(x_i) = y] - 1/2 \right) \right|.
$$

We are solving a weighted zero-one loss minimization problem.
Suppose $h_j : \mathbb{R}^d \to \{-1, +1\}$; then $h_j(x)y = 2 \cdot 1[h_j(x) = y] - 1$, and each step solves

$$\arg \max_j \left| \sum_{i=1}^{n} q_i h_j(x_i) y_i \right| = \arg \max_j \left| \sum_{i=1}^{n} q_i \left(1[h_j(x_i) = y] - 1/2\right) \right|.$$  

We are solving a weighted zero-one loss minimization problem.

**Remarks:**

- The classical choice of coordinate descent is equivalent to solving a problem akin to weighted zero-one loss minimization.
- We can abstract away a finite set $(h_1, \ldots, h_T)$, and have an arbitrary set of predictors (e.g., all linear classifiers).
There is a **Weak Learning Oracle**, and a corresponding $\gamma$-weak-learnable assumption:

A set of points is $\gamma$-weak-learnable a weak learning oracle if for any weighting $q$, it returns predictor $h$ so that $\mathbb{E}_q \left( h(X)Y \right) \geq \gamma$.

**Interpretation:** for any reweighting $q$, we get a predictor $h$ which is at least $\gamma$-correlated with the target.
Classical boosting setup

There is a Weak Learning Oracle, and a corresponding $\gamma$-weak-learnable assumption:

\[
\text{A set of points is } \gamma\text{-weak-learnable a weak learning oracle if for any weighting } q, \text{ it returns predictor } h \text{ so that } E_q(h(X)Y) \geq \gamma.
\]

**Interpretation:** For any reweighting $q$, we get a predictor $h$ which is at least $\gamma$-correlated with the target.

**Remarks:**

- The classical methods iteratively invoke the oracle with different weightings and then output a final aggregated predictor.
- The best-known method, AdaBoost, performs coordinate-descent updates (invoking the oracle) with a specific step size, and needs $O\left(\frac{1}{\gamma^2} \ln\left(\frac{1}{\epsilon}\right)\right)$ iterations for accuracy $\epsilon > 0$.
- The original description of AdaBoost is in terms of the sequence of weightings $q_1, q_2, \ldots$, and says nothing about coordinate descent.
- **Adaptive Boosting:** Method doesn’t need to know $\gamma$, and adapts to varying $\gamma_t := E_{q_t}(h_t(X)Y)$. 
Example: AdaBoost with decision stumps

Weak learning oracle (WLO): pick the best decision stump, meaning
\[ F := \{ x \mapsto \text{sign}(x_i - b) : i \in \{1, \ldots, d\}, b \in \mathbb{R} \}. \]
(Straightforward to handle weights in ERM.)
Example: AdaBoost with decision stumps

Weak learning oracle (WLO): pick the best decision stump, meaning
\[ \mathcal{F} := \{ \mathbf{x} \mapsto \text{sign}(x_i - b) : i \in \{1, \ldots, d\}, b \in \mathbb{R} \}. \]

(Straightforward to handle weights in ERM.)

Remark:

▶ Only need to consider \( O(n) \) stumps (Why?).
Example: execution of AdaBoost
Example: execution of AdaBoost
Example: execution of AdaBoost
Example: execution of AdaBoost
Example: execution of AdaBoost

\[ D_1 \]

\[ D_2 \]

\[ D_3 \]

\[ f_1 \]

\[ f_2 \]
Example: execution of AdaBoost
Example: final classifier from AdaBoost

\[
\hat{f}(x) = \text{sign}(0.42f_1(x) + 0.65f_2(x) + 0.92f_3(x))
\]

(Zero training error rate!)
Example: final classifier from AdaBoost

Final classifier
\[ \hat{f}(\boldsymbol{x}) = \text{sign}(0.42 f_1(\boldsymbol{x}) + 0.65 f_2(\boldsymbol{x}) + 0.92 f_3(\boldsymbol{x})) \]

(Zero training error rate!)
A typical run of boosting.

AdaBoost+C4.5 on “letters” dataset.

![Graph showing training and test error rates over rounds](image)

- **Training error rate** is zero after just five rounds, but **test error rate continues to decrease, even up to 1000 rounds!**

(Number of nodes across all decision trees in \( \hat{f} \) is \( >2 \times 10^6 \))

(Figure 1.7 from Schapire & Freund text)
Final classifier from AdaBoost:

\[
\hat{f}(x) = \text{sign} \left( \frac{\sum_{t=1}^{T} \alpha_t f_t(x)}{\sum_{t=1}^{T} |\alpha_t|} \right). \\
g(x) \in [-1, +1]
\]

Call \( y \cdot g(x) \in [-1, +1] \) the **margin** achieved on example \((x, y)\).
(Note: \( \ell_1 \) not \( \ell_2 \) normalized.)
Boosting the margin.

Final classifier from AdaBoost:

\[
\hat{f}(x) = \text{sign} \left( \frac{\sum_{t=1}^{T} \alpha_t f_t(x)}{\sum_{t=1}^{T} |\alpha_t|} \right).
\]

\(g(x) \in [-1, +1]\)

Call \(y \cdot g(x) \in [-1, +1]\) the \textbf{margin} achieved on example \((x, y)\).

(Note: \(\ell_1\) not \(\ell_2\) normalized.)

\textbf{Margin theory} [Schapire, Freund, Bartlett, and Lee, 1998]:

- Larger margins \(\Rightarrow\) better \textbf{generalization}, independent of \(T\).
- AdaBoost tends to increase margins on training examples.

<table>
<thead>
<tr>
<th>“letters” dataset:</th>
<th>(T = 5)</th>
<th>(T = 100)</th>
<th>(T = 1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>training error rate</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>test error rate</td>
<td>8.4%</td>
<td>3.3%</td>
<td>3.1%</td>
</tr>
<tr>
<td>% margins (\leq 0.5)</td>
<td>7.7%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>min. margin</td>
<td>0.14</td>
<td>0.52</td>
<td>0.55</td>
</tr>
</tbody>
</table>

- Similar phenomenon in deep networks and gradient descent.
Given \(((x_i, y_i))_{i=1}^{n}\) and \(f\), plot **unnormalized margin distribution**

\[ f(x_i) y_i - \max_{y \neq y_i} f(x_i) y. \]
Margin plots

Given \(((x_i, y_i))_{i=1}^n\) and \(f\), plot unnormalized margin distribution

\[ f(x_i)y_i - \max_{y \neq y_i} f(x_i)y. \]

Boosted stumps.
(\(\mathcal{O}(n)\) param.)
Given \(((x_i, y_i))_{i=1}^n\) and \(f\), plot unnormalized margin distribution

\[ f(x_i)y_i - \max_{y \neq y_i} f(x_i)y. \]

Boosted stumps.
\((\mathcal{O}(n) \text{ param.})\)
Margin plots

Given \((x_i, y_i)_{i=1}^n\) and \(f\), plot **unnormalized margin distribution**

\[ f(x_i)y_i - \max_{y \neq y_i} f(x_i)y. \]

**Boosted stumps.**

\((\mathcal{O}(n)\text{ param.})\)
Margin plots

Given \( ((x_i, y_i))_{i=1}^n \) and \( f \), plot **unnormalized margin distribution**

\[
f(x_i)y_i - \max_{y \neq y_i} f(x_i)y.
\]

Boosted stumps.
\((O(n) \text{ param.})\)
Margin plots

Given \( ((x_i, y_i))_{i=1}^n \) and \( f \), plot **unnormalized margin distribution**

\[
f(x_i) y_i - \max_{y \neq y_i} f(x_i) y.
\]

Boosted stumps.

\( \mathcal{O}(n) \) param.
Margin plots

Given \(((x_i, y_i))_{i=1}^n\) and \(f\), plot \textbf{unnormalized margin distribution} \(f(x_i) y_i - \max_{y \neq y_i} f(x_i) y\).
Margin plots

Given \( ((x_i, y_i))_{i=1}^n \) and \( f \), plot the unnormalized margin distribution \( f(x_i) y_i - \max_{y \neq y_i} f(x_i) y. \)

Boosted stumps.  
\( (\mathcal{O}(n) \text{ param.}) \)

2-layer ReLU.  
\( (\mathcal{O}(n) \text{ param.}) \)

3-layer ReLU.  
\( (\mathcal{O}(n) \text{ param.}) \)
Margin plots

Given \(((x_i, y_i))_{i=1}^n\) and \(f\), plot unnormalized margin distribution

\[ f(x_i) y_i - \max_{y \neq y_i} f(x_i) y. \]
Margin plots

Given \(((x_i, y_i))_{i=1}^n\) and \(f\), plot \textbf{unnormalized margin distribution}

\[ f(x_i)y_i - \max_{y \neq y_i} f(x_i)y. \]

\[ \begin{array}{c}
\text{Boosted stumps.} \\
(\mathcal{O}(n) \text{ param.})
\end{array} \quad \begin{array}{c}
2\text{-layer ReLU.} \\
(\mathcal{O}(n) \text{ param.})
\end{array} \quad \begin{array}{c}
3\text{-layer ReLU.} \\
(\mathcal{O}(n) \text{ param.})
\end{array} \]
Margin plots

Given \( ((x_i, y_i))_{i=1}^{n} \) and \( f \), plot \textbf{unnormalized margin distribution}

\[
f(x_i) y_i - \max_{y \neq y_i} f(x_i) y.
\]

Boosted stumps.  
(\( \mathcal{O}(n) \) param.)

2-layer ReLU.  
(\( \mathcal{O}(n) \) param.)

3-layer ReLU.  
(\( \mathcal{O}(n) \) param.)
Summary for today

- Bagging.
- Boosting.
(Appendix.)
Ensemble methods and deep learning; dropout?