"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.
Bagging idea:

- If the predictors have independent errors, a majority vote of their outputs should be good.
Suppose $t$ classifiers $\left(f_j\right)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := \mathbb{1}[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!).
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). We can model the distribution of errors with $\text{Binom}(t, 0.4)$. 
Behavior of independent errors

Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!).

We can model the distribution of errors with $\text{Binom}(t, 0.4)$. 
Behavior of independent errors

Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). We can model the distribution of errors with $\text{Binom}(t, 0.4)$. 

![Distribution of errors](image-url)
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!).

We can model the distribution of errors with $\text{Binom}(t, 0.4)$. 

![Histogram of errors for $t = 30$ classifiers. The green region indicates the error of majority vote.](image)
Suppose \( t \) classifiers \((f_j)_{j=1}^t\), fixed example \((x, y)\), \textbf{independent} error probabilities \( Z_j := 1[\text{sgn}(f_j(x)) \neq y] \), and \( \Pr[Z_j] = 0.4 =: p \) (pretty bad!).

We can model the distribution of errors with Binom\( (t, 0.4) \).

\[ 0.075 \ll 0.4 \]
Suppose \( t \) classifiers \( (f_j)_{j=1}^t \), fixed example \( (x, y) \), independent error probabilities \( Z_j := 1[\text{sgn}(f_j(x)) \neq y] \), and \( \Pr[Z_j] = 0.4 =: p \) (pretty bad!).

We can model the distribution of errors with Binom\((t, 0.4)\).
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!).

We can model the distribution of errors with $\text{Binom}(t, 0.4)$. 

\[
\begin{align*}
\text{Red: all classifiers wrong.} \\
\text{Green: at least half classifiers wrong.}
\end{align*}
\]
Behavior of independent errors

Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!).

We can model the distribution of errors with $\text{Binom}(t, 0.4)$.

**Red:** all classifiers wrong.

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

We see that $p = 0.4$ is a pretty bad choice for the error probability.
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). We can model the distribution of errors with $\text{Binom}(t, 0.4)$. 

**Red:** all classifiers wrong.

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

$0.075 \ll 0.4$ !!!
Behavior of independent errors

Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). We can model the distribution of errors with Binom($t, 0.4$).

**Red:** all classifiers wrong.

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

\[ \frac{0.0}{0.00} \quad \frac{0.0}{0.05} \quad \frac{0.0}{0.10} \quad \frac{0.0}{0.15} \quad \frac{0.0}{0.20} \quad \frac{0.0}{0.25} \quad \frac{0.0}{0.30} \quad \frac{0.0}{0.35} \quad \frac{0.0}{0.40} \quad \frac{0.0}{0.45} \quad \frac{0.0}{0.50} \quad \frac{0.0}{0.55} \quad \frac{0.0}{0.60} \quad \frac{0.0}{0.65} \quad \frac{0.0}{0.70} \quad \frac{0.0}{0.75} \quad \frac{0.0}{0.80} \quad \frac{0.0}{0.85} \quad \frac{0.0}{0.90} \quad \frac{0.0}{0.95} \quad \frac{0.0}{1.00} \]

# classifiers = $t = 4$, fraction red = 0.0256

\[
\text{Green region is error of majority vote!}
\]

\[ 0.075 \ll 0.4 \]
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, \textbf{independent} error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). We can model the distribution of errors with $\text{Binom}(t, 0.4)$. \textbf{Red:} all classifiers wrong.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure.png}
\caption{#classifiers = $t = 5$, fraction red = 0.01024}
\end{figure}
Behavior of independent errors

Suppose \( t \) classifiers \( (f_j)_{j=1}^t \), fixed example \((x, y)\), independent error probabilities \( Z_j := 1[\text{sgn}(f_j(x)) \neq y] \), and \( \Pr[Z_j] = 0.4 =: p \) (pretty bad!).

We can model the distribution of errors with Binom(\( t, 0.4 \)).

**Red:** all classifiers wrong.

**Green** region is error of majority vote!

\[
\begin{array}{c|c|c|c|c|c|c}
\#\text{classifiers} = t & 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1.0 \\
\hline
\text{fraction red} & 0.00 & 0.05 & 0.10 & 0.15 & 0.20 & 0.25 \end{array}
\]

\[
\text{#classifiers = 6, fraction red = 0.004096}
\]
Suppose $t$ classifiers $(f_j)_{j=1}^{t}$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z] = 0.4 =: p$ (pretty bad!).

We can model the distribution of errors with $\text{Binom}(t, 0.4)$.

**Red:** all classifiers wrong.

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

0.075 $\ll$ 0.4 !!!

\[\text{#classifiers = } t = 7, \text{ fraction red } = 0.0016384\]
Suppose $t$ classifiers $(f_j)^t_{j=1}$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). We can model the distribution of errors with $\text{Binom}(t, 0.4)$. 

Green: at least half classifiers wrong.
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, \textbf{independent} error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!).

We can model the distribution of errors with Binom$(t, 0.4)$.

\textbf{Green:} at least half classifiers wrong.
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, \textbf{independent} error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). We can model the distribution of errors with $\text{Binom}(t, 0.4)$.

\textbf{Green:} at least half classifiers wrong.
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!).

We can model the distribution of errors with $\text{Binom}(t, 0.4)$. 

**Green:** at least half classifiers wrong.
Behavior of independent errors

Suppose $t$ classifiers $(f_j)_{j=1}^{t}$, fixed example $(x, y)$, independent error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). We can model the distribution of errors with $\text{Binom}(t, 0.4)$.

**Green:** at least half classifiers wrong.
Suppose $t$ classifiers $(f_j)_{j=1}^t$, fixed example $(x, y)$, 
**independent** error probabilities $Z_j := 1[\text{sgn}(f_j(x)) \neq y]$, 
and $\Pr[Z_j] = 0.4 =: p$ (pretty bad!). 
We can model the distribution of errors with Binom($t, 0.4$). 
\underline{Green}: at least half classifiers wrong.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{Green region is error of majority vote!}
\end{figure}
Suppose \( t \) classifiers \((f_j)^t_{j=1}\), fixed example \((x, y)\), independent error probabilities \( Z_j := 1[\text{sgn}(f_j(x)) \neq y] \), and \( \Pr[Z_j] = 0.4 =: p \) (pretty bad!).

We can model the distribution of errors with Binom\((t, 0.4)\).

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

Green region is error of majority vote! 0.075 \( \ll \) 0.4 !!!
Green region is error of majority vote! Defining $\hat{y}_j := \text{sgn}(f_j(x))$, $\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) := \begin{cases} +1 & \text{when } \sum_j \hat{y}_j \geq 0, \\ -1 & \text{when } \sum_j \hat{y}_j < 0. \end{cases}$

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

$$\Pr[\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) \neq y] = \Pr[\text{Binom}(t, p) \geq t/2] = \sum_{i=t/2}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq \exp\left(-2t(1/2 - p)^2\right).$$
Green region is error of majority vote! Defining \( \hat{y}_j := \text{sgn}(f_j(x)) \),

\[
\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) := \begin{cases} 
+1 & \text{when } \sum_j \hat{y}_j \geq 0, \\
-1 & \text{when } \sum_j \hat{y}_j < 0.
\end{cases}
\]

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

\[
\Pr[\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) \neq y] = \Pr[\text{Binom}(t, p) \geq t/2]
\]

\[
= \sum_{i=t/2}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq \exp \left(-2t(1/2 - p)^2\right).
\]
Green region is error of majority vote! Defining $\hat{y}_j := \text{sgn}(f_j(x))$,

$$\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) := \begin{cases} +1 & \text{when } \sum_j \hat{y}_j \geq 0, \\ -1 & \text{when } \sum_j \hat{y}_j < 0. \end{cases}$$

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

$$\Pr[\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) \neq y] = \Pr[\text{Binom}(t, p) \geq t/2]$$

$$= \sum_{i=t/2}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq \exp\left(-\frac{2t}{t/2} \left(\frac{1}{2} - p\right)^2\right).$$
Green region is error of majority vote! Defining $\hat{y}_j := \text{sgn}(f_j(x))$, 

$$\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) := \begin{cases} +1 \quad \text{when} \quad \sum_j \hat{y}_j \geq 0, \\ -1 \quad \text{when} \quad \sum_j \hat{y}_j < 0. \end{cases}$$

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

$$\Pr[\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) \neq y] = \Pr[\text{Binom}(t, p) \geq t/2]$$

$$= \sum_{i=t/2}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq \exp \left( -2t(1/2 - p)^2 \right).$$
Green region is error of majority vote! Defining $\hat{y}_j := \text{sgn}(f_j(x))$,

$$\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) := \begin{cases} +1 & \text{when } \sum_j \hat{y}_j \geq 0, \\ -1 & \text{when } \sum_j \hat{y}_j < 0. \end{cases}$$

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

$$\Pr[\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) \neq y] = \Pr[\text{Binom}(t, p) \geq t/2]$$

$$= \sum_{i=t/2}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq \exp\left(-2t(1/2 - p)^2\right).$$
Green region is error of majority vote! Defining $\hat{y}_j := \text{sgn}(f_j(x))$,

$$\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) := \begin{cases} +1 \quad \text{when } \sum_j \hat{y}_j \geq 0, \\ -1 \quad \text{when } \sum_j \hat{y}_j < 0. \end{cases}$$

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

$$\Pr[\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_t) \neq y] = \Pr[\text{Binom}(t, p) \geq t/2]$$

$$= \sum_{i=t/2}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq \exp \left(-2t(1/2 - p)^2\right).$$
Green region is error of majority vote!

Error of majority vote classifier goes down **exponentially** in $t$.

$$\Pr[\text{MAJ}(\hat{y}_1, \ldots, \hat{y}_m) \neq y] = \Pr[\text{Binom}(t, p) \geq t/2]$$

$$= \sum_{i=t/2}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq \exp \left(-2m(1/2 - p)^2 \right).$$
Technical aside: **Hoeffding’s inequality**

How did we estimate $\Pr[\text{Binom}(t, p) \geq t/2] \leq \exp \left( -2t(1/2 - p)^2 \right)$?

**Theorem (Hoeffding’s inequality).** Given IID $Z_i \in [a, b]$,

$$\Pr \left[ \frac{1}{t} \sum_{i=1}^{t} Z_i - \mathbb{E}Z_1 \geq \epsilon \right] \leq \exp \left( \frac{-2t\epsilon^2}{(b-a)^2} \right).$$
Technical aside: **Hoeffding’s inequality**

How did we estimate $\Pr[\text{Binom}(t, p) \geq t/2] \leq \exp \left( -2t(1/2 - p)^2 \right)$?

**Theorem (Hoeffding’s inequality).** Given IID $Z_i \in [a, b]$,

$$
\Pr \left[ \frac{1}{t} \sum_{i=1}^{t} Z_i - \mathbb{E}Z_1 \geq \epsilon \right] \leq \exp \left( \frac{-2t\epsilon^2}{(b - a)^2} \right).
$$

Our application: set $p := \mathbb{E}Z_1$, we want

$$
\Pr \left[ \text{MAJ}(\hat{y}_1, \ldots, \hat{y}_m) \neq y \right] \leq \Pr \left[ \sum_{j=1}^{t} Z_j \leq \frac{t}{2} \right]
$$

$$
= \Pr \left[ \frac{1}{t} \sum_{j=1}^{t} Z_j - \mathbb{E}Z_1 \leq \frac{1}{2} - p \right]
$$

$$
\leq \exp \left( -2t(1/2 - p)^2 \right).
$$
Algorithmically using independent errors

**Training 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 \( x \mapsto \text{MAJ}(f_1(x), \ldots, f_T(x)) \).
Algorithmically using independent errors

Training 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 \( 1/T \) 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 $f$!
▶ Bad news: errors no longer independent. . . ?
**Bagging** = Bootstrap agggregating (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:

$$1 - \frac{1}{n}$$; for large $n$:

$$\lim_{n \to \infty} 1 - \frac{1}{n} = 1 - e^{-1} \approx 0.3679$$.

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!
Sampling with replacement?

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

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 \( S_t \), considering only \( \sqrt{d} \) random features at each split.
3. Output \( x \mapsto \text{MAJ}(f_1(x), \ldots, f_T(x)) \).
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 $S_t$, considering only $\sqrt{d}$ random features at each split.

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

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

- Classifier errors no longer independent.
- Predict with a reweighted majority.
- There is a rich theory with many interpretations.
Classifiers errors no longer independent.

Predict with a **reweighted** majority.

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 empirical risk

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

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

Remarks.

▶ If \(\ell\) is convex, this is standard linear prediction: convex in \(w\).

▶ 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 implementations: xgboost and catboost.
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 empirical risk

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

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

Remarks.

- If \(\ell\) is convex, this is standard linear prediction: convex in \(w\).
- 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 implementations: xgboost and catboost.
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

$\hat{y} = 2$
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

$x_1 > 1.7$
Classifying irises by sepal and petal measurements

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

\[
\hat{y} = 1 \quad \text{if} \quad x_1 > 1.7
\]

\[
\hat{y} = 3 \quad \text{if} \quad x_1 \leq 1.7
\]
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

\[ x_1 > 1.7 \]

\[ \hat{y} = 1 \quad \hat{y} = 3 \]

... 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.
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. ($\mathcal{O}(n)$ param.) 2-layer ReLU. ($\mathcal{O}(n)$ param.) 3-layer ReLU. ($\mathcal{O}(n)$ 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.
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.)
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 $\mathbf{w} \in \mathbb{R}^{T}$, 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.)
More realistic boosting algorithm outline

1. Initial predictor \( f_0(x) := 0 \).

2. For \( t \in \{1, 2, \ldots \} \):
   
   2.1 Define example weights \( q_i := \exp(-y_i f_t(x_i)) \).
   
   2.2 Selecting maximally correlated predictor:
   
   \[
   (s_t, h_t) := \arg \max_{s \in \pm 1} \sum_{i=1}^{n} q_i y_i h(x_i)
   \]

   2.3 Update predictor: \( f_t := f_{t-1} - \eta_t s_t h_t \).
1. Initial predictor $f_0(x) := 0$.

2. For $t \in \{1, 2, \ldots\}$:
   
   2.1 Define **example weights** $q_i := \exp(-y_if_t(x_i))$.

   2.2 Selecting **maximally correlated predictor**:

   $$(s_t, h_t) := \arg\max_{s \in \pm 1, h \in H} \sum_{i=1}^{n} q_i y_i h(x_i)$$

   2.3 Update predictor: $f_t := f_{t-1} - \eta_t s_t h_t$.

**Remarks.** (Further details in appendix.)

1. $H$ typically infinite; the inner maximization is its own ERM problem.

2. This is equivalent to **coordinate descent** with $\ell(y^\hat{}) = \exp(-y^\hat{})$: reweighting is loss derivative term.

3. Step size $\eta_t$ usually carefully chosen.

4. These remarks follow “AdaBoost” (which has the most theory); xgboost has many differences.
Boosting and classifier complexity

![Graph showing training and test error rates over iterations]

**Resilient to overfitting.**

- # classifiers (and thus "complexity") increase, but test error stays low.
- Many consider this similar to deep learning, where test error plateaus or improves with increasing network size.
Margins and generalization?

- Write final classifier as $f_t(x) := \sum_{i=1}^{m} w_i h_i(x)$.

- Define $\ell_1$ margins $\gamma_i := \frac{y_i f_t(x_i)}{\|w\|_1}$.
  (Recall margin with SVM: $y_i x_i^T w / \|w\|_2$.)

- Margins and test error seem to improve on “letters” dataset:

<table>
<thead>
<tr>
<th></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>fraction $\gamma_i \leq 0.5$</td>
<td>7.7%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>min$_i \gamma_i$</td>
<td>0.14</td>
<td>0.52</td>
<td>0.55</td>
</tr>
</tbody>
</table>

- There is a lot of theory about margins, generalization properties of margins, and margins in deep learning and boosting.
Summary for today

- Bagging.
- Boosting.
(Appendix.)
The original presentation of dropout called it an ensemble method (you train random subnetworks to do well, and at test time predict with their combination). There’s no evidence or math behind this assertion, but still it’s an interesting perspective.
Classical perspective on boosting: coordinate descent?

The classical methods used coordinate descent:

- Find the maximum magnitude coordinate of the gradient:

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

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

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

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

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

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

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

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

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 $\mathbb{E}_q (h(X)Y) \geq \gamma$.

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:

A set of points is $\gamma$-weak-learnable a weak learning oracle if for any weighting $q$, it returns predictor $h$ 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
\( \mathcal{F} := \{ x \mapsto \text{sign}(x_i - b) : i \in \{1, \ldots, d\}, b \in \mathbb{R} \} \).

(Straightforward to handle weights in ERM.)
Example: AdaBoost with decision stumps

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

Remark:
- Only need to consider \( O(n) \) stumps (Why?).

(This example from Schapire&Freund’s book.)
Example: execution of AdaBoost
Example: execution of AdaBoost

$D_1$

$f_1$
Example: execution of AdaBoost
Example: execution of AdaBoost
Example: execution of AdaBoost
Example: execution of AdaBoost
Example: final classifier from AdaBoost

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

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

AdaBoost+C4.5 on “letters” dataset.

![Error rate graph](image)

- **C4.5 test error**
- **AdaBoost test error**
- **AdaBoost training error**

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

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

(Figure 1.7 from Schapire & Freund text)
Boosting the margin.

Final classifier from AdaBoost:

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

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

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

(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 margin achieved on example \((x, y)\).
(Note: \(\ell_1\) not \(\ell_2\) normalized.)

**Margin theory** [Schapire, Freund, Bartlett, and Lee, 1998]:

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

**“letters” dataset:**

<table>
<thead>
<tr>
<th></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**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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