Boosting
Simple (weak) classifiers are good!

- Logistic regression with simple features
- Shallow decision trees
- Decision stumps

Low variance. Learning is fast!

But high bias...
Finding a classifier that’s just right

Classification error

Model complexity

true error

train error

Weak learner

Need stronger learner

Option 1: add more features or depth
Option 2: ?????
Boosting question

“Can a set of weak learners be combined to create a stronger learner?” *Kearns and Valiant (1988)*

Yes! *Schapire (1990)*

Boosting

Amazing impact: • simple approach • widely used in industry • wins most Kaggle competitions • great systems (e.g., XGBoost)
Ensemble classifier
A single classifier

Input: $x$

Income > $100K? [Decision Node]

- Yes
  - Safe [Outcome]

- No
  - Risky [Outcome]

Output: $\hat{y} = f(x)$

- Either +1 or -1
Ensemble methods: Each classifier "votes" on prediction

\[ x_i = (\text{Income}=$120K, \text{Credit}=\text{Bad}, \text{Savings}=$50K, \text{Market}=\text{Good}) \]

\[ f_1(x_i) = +1 \]
\[ f_2(x_i) = -1 \]
\[ f_3(x_i) = -1 \]
\[ f_4(x_i) = +1 \]

\[ F(x_i) = \text{sign}(w_1 f_1(x_i) + w_2 f_2(x_i) + w_3 f_3(x_i) + w_4 f_4(x_i)) \]
Ensemble classifier in general

• Goal:
  – Predict output $y$
    • Either +1 or -1
  – From input $x$

• Learn ensemble model:
  – Classifiers: $f_1(x), f_2(x), \ldots, f_T(x)$
  – Coefficients: $\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_T$

• Prediction:

$$\hat{y} = sign \left( \sum_{t=1}^{T} \hat{w}_t f_t(x) \right)$$
Boosting
Training a classifier

Training data -> Learn classifier: \( f(x) \) -> Predict: \( \hat{y} = \text{sign}(f(x)) \)
Learning decision stump

<table>
<thead>
<tr>
<th>Credit</th>
<th>Income</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$130K</td>
<td>Safe</td>
</tr>
<tr>
<td>B</td>
<td>$80K</td>
<td>Risky</td>
</tr>
<tr>
<td>C</td>
<td>$110K</td>
<td>Risky</td>
</tr>
<tr>
<td>A</td>
<td>$110K</td>
<td>Safe</td>
</tr>
<tr>
<td>A</td>
<td>$90K</td>
<td>Safe</td>
</tr>
<tr>
<td>A</td>
<td>$98K</td>
<td>Safe</td>
</tr>
<tr>
<td>B</td>
<td>$120K</td>
<td>Safe</td>
</tr>
<tr>
<td>C</td>
<td>$30K</td>
<td>Risky</td>
</tr>
<tr>
<td>C</td>
<td>$60K</td>
<td>Risky</td>
</tr>
<tr>
<td>B</td>
<td>$95K</td>
<td>Safe</td>
</tr>
<tr>
<td>A</td>
<td>$60K</td>
<td>Safe</td>
</tr>
</tbody>
</table>

Income? > $100K

3 1

\( \hat{y} = \text{Safe} \)

Income? ≤ $100K

4 3

\( \hat{y} = \text{Safe} \)
Boosting = Focus learning on “hard” points

Training data → Learn classifier → Predict $\hat{y} = \text{sign}(f(x))$ → Evaluate → Learn where $f(x)$ makes mistakes

Boosting: focus next classifier on places where $f(x)$ does less well
Learning on weighted data:

*More weight on “hard” or more important points*

- Weighted dataset:
  - Each \( x_i, y_i \) weighted by \( \alpha_i \)
    - More important point = higher weight \( \alpha_i \)

- Learning:
  - Data point \( i \) counts as \( \alpha_i \) data points
    - E.g., \( \alpha_i = 2 \) ➔ count point twice
Learning a decision stump on weighted data

<table>
<thead>
<tr>
<th>Credit</th>
<th>Income</th>
<th>y</th>
<th>Weight α</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$130K</td>
<td>Safe</td>
<td>0.5</td>
</tr>
<tr>
<td>B</td>
<td>$80K</td>
<td>Risky</td>
<td>1.5</td>
</tr>
<tr>
<td>C</td>
<td>$110K</td>
<td>Risky</td>
<td>1.2</td>
</tr>
<tr>
<td>A</td>
<td>$110K</td>
<td>Safe</td>
<td>0.8</td>
</tr>
<tr>
<td>A</td>
<td>$90K</td>
<td>Safe</td>
<td>0.6</td>
</tr>
<tr>
<td>B</td>
<td>$120K</td>
<td>Safe</td>
<td>0.7</td>
</tr>
<tr>
<td>C</td>
<td>$30K</td>
<td>Risky</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>$60K</td>
<td>Risky</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>$95K</td>
<td>Safe</td>
<td>0.8</td>
</tr>
<tr>
<td>A</td>
<td>$60K</td>
<td>Safe</td>
<td>0.7</td>
</tr>
<tr>
<td>A</td>
<td>$98K</td>
<td>Safe</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Increase weight $\alpha$ of harder/misclassified points

Increase weight $\alpha$ of harder/misclassified points

Income?

> $100K

- $\hat{y} = Safe$

- $\alpha = 2$
- $\alpha = 1.2$

≤ $100K

- $\hat{y} = Risky$

- $\alpha = 3$
- $\alpha = 6.5$

©2021 Carlos Guestrin
Boosting = Greedy learning ensembles from data

- **Training data**
- **Learn classifier**
  - Predict $\hat{y} = \text{sign}(f_1(x))$
- **Weighted data**
- **Learn classifier & coefficient**
  - Predict $\hat{y} = \text{sign}(\hat{w}_1 f_1(x) + \hat{w}_2 f_2(x))$

Higher weight for points where $f_1(x)$ is wrong
AdaBoost algorithm
AdaBoost: learning ensemble

[Freund & Schapire 1999]

- Start with same weight for all points: $\alpha_i = 1/N$

- For $t = 1, \ldots, T$
  - Learn $f_t(x)$ with data weights $\alpha_i$
  - Compute coefficient $\hat{w}_t$
  - Recompute weights $\alpha_i$

- Final model predicts by:

$$\hat{y} = \text{sign} \left( \sum_{t=1}^{T} \hat{w}_t f_t(x) \right)$$
Computing coefficient $\hat{w}_t$
AdaBoost: Computing coefficient $\hat{w}_t$ of classifier $f_t(x)$

- $f_t(x)$ is good $\Rightarrow f_t$ has low training error

- Measuring error in weighted data?
  - Just weighted # of misclassified points
AdaBoost:
Formula for computing coefficient $\hat{w}_t$ of classifier $f_t(x)$

$$\hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - \text{weighted_error}(f_t)}{\text{weighted_error}(f_t)} \right)$$

<table>
<thead>
<tr>
<th>Is $f_t(x)$ good?</th>
<th>weighted_error($f_t$) on training data</th>
<th>$\frac{1 - \text{weighted_error}(f_t)}{\text{weighted_error}(f_t)}$</th>
<th>$\hat{w}_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AdaBoost: learning ensemble

• Start with same weight for all points: \( \alpha_i = 1/N \)

• For \( t = 1, \ldots, T \)
  – Learn \( f_t(x) \) with data weights \( \alpha_i \)
  – Compute coefficient \( \hat{w}_t \)
  – Recompute weights \( \alpha_i \)

• Final model predicts by:
  \[
  \hat{y} = sign \left( \sum_{t=1}^{T} \hat{w}_t f_t(x) \right)
  \]
Recompute weights $\alpha_i$
AdaBoost: Updating weights $\alpha_i$ based on where classifier $f_t(x)$ makes mistakes

- Did $f_t$ get $x_i$ right?
  - Yes: Decrease $\alpha_i$
  - No: Increase $\alpha_i$
AdaBoost: Formula for updating weights $\alpha_i$

\[
\alpha_i \left\{ \begin{array}{l}
\alpha_i e^{-\hat{w}_t}, \text{ if } f_t(x_i) = y_i \\
\alpha_i e^{\hat{w}_t}, \text{ if } f_t(x_i) \neq y_i
\end{array} \right.
\]

<table>
<thead>
<tr>
<th>$f_t(x_i)=y_i$?</th>
<th>$\hat{w}_t$</th>
<th>Multiply $\alpha_i$ by</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AdaBoost: learning ensemble

- Start with same weight for all points: $\alpha_i = 1/N$

- For $t = 1, \ldots, T$
  - Learn $f_t(x)$ with data weights $\alpha_i$
  - Compute coefficient $\hat{w}_t$
  - Recompute weights $\alpha_i$

- Final model predicts by:
  $$\hat{y} = \text{sign} \left( \sum_{t=1}^{T} \hat{w}_t f_t(x) \right)$$

\[ \hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - \text{weighted_error}(f_t)}{\text{weighted_error}(f_t)} \right) \]

\[ \alpha_i \left\{ \begin{array}{ll}
\alpha_i e^{-\hat{w}_t}, & \text{if } f_t(x_i) = y_i \\
\alpha_i e^{\hat{w}_t}, & \text{if } f_t(x_i) \neq y_i 
\end{array} \right. \]
AdaBoost: Normalizing weights $\alpha_i$

If $x_i$ often mistake, weight $\alpha_i$ gets very large

If $x_i$ often correct, weight $\alpha_i$ gets very small

Can cause numerical instability after many iterations

Normalize weights to add up to 1 after every iteration

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$
AdaBoost: learning ensemble

• Start with same weight for all points: \( \alpha_i = 1/N \)

• For \( t = 1, \ldots, T \)
  - Learn \( f_t(x) \) with data weights \( \alpha_i \)
  - Compute coefficient \( \hat{w}_t \)
  - Recompute weights \( \alpha_i \)
  - Normalize weights \( \alpha_i \)

• Final model predicts by:

\[
\hat{y} = \text{sign}\left( \sum_{t=1}^{T} \hat{w}_t f_t(x) \right)
\]
AdaBoost example:
A visualization
\( t=1: \) Just learn a classifier on original data
Updating weights $\alpha_i$

Increase weight $\alpha_i$ of misclassified points

Learned decision stump $f_1(x)$

New data weights $\alpha_i$
t=2: Learn classifier on weighted data

Learned decision stump $f_2(x)$ on weighted data
Ensemble becomes weighted sum of learned classifiers

\[
\hat{w}_1 \begin{bmatrix} f_1(x) \end{bmatrix} + 0.61 \hat{w}_2 \begin{bmatrix} f_2(x) \end{bmatrix} = 0.53 \begin{bmatrix} f_2(x) \end{bmatrix}
\]
Decision boundary of ensemble classifier after 30 iterations

training_error = 0
Boosting convergence & overfitting
Boosting question revisited

“Can a set of weak learners be combined to create a stronger learner?” *Kearns and Valiant (1988)*

Yes! *Schapire (1990)*

Boosting
After some iterations, training error of boosting goes to zero!!!
AdaBoost Theorem

Under some technical conditions...

Training error of boosted classifier $\rightarrow 0$ as $T \rightarrow \infty$

May oscillate a bit

But will generally decrease, & eventually become 0!
Condition of AdaBoost Theorem

Under some technical conditions...

Training error of boosted classifier → 0 as $T \to \infty$

Condition = At every $t$, can find a weak learner with $\text{weighted_error}(f_t) < 0.5$

Not always possible

Nonetheless, boosting often yields great training error

Extreme example: No classifier can separate a +1 on top of -1
Decision trees on loan data

![Graph showing decision trees with 39% test error and 8% training error, with an indication of overfitting.]

Boosted decision stumps on loan data

![Graph showing boosted decision stumps with 32% test error and 28.5% training error, indicating better fit and lower test error.]

©2021 Carlos Guestrin
Boosting tends to be robust to overfitting

Test set performance about constant for many iterations ➔ Less sensitive to choice of T
But boosting will eventually overfit, so must choose max number of components $T$. Test error eventually increases to 33% (overfits). Best test error around 31%.
Summary of boosting
Variants of boosting and related algorithms

There are hundreds of variants of boosting, most important:

**Gradient boosting**
- Like AdaBoost, but useful beyond basic classification
- Great implementations available (e.g., XGBoost)

Many other approaches to learn ensembles, most important:

**Random forests**
- **Bagging:** Pick random subsets of the data
  - Learn a tree in each subset
  - Average predictions
- Simpler than boosting & easier to parallelize
- Typically higher error than boosting for same # of trees (# iterations T)
Impact of boosting (*spoiler alert... HUGE IMPACT*)

<table>
<thead>
<tr>
<th>Amongst most useful ML methods ever created</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extremely useful in computer vision</td>
</tr>
<tr>
<td>• Standard approach for face detection, for example</td>
</tr>
<tr>
<td>Used by most winners of ML competitions</td>
</tr>
<tr>
<td>(Kaggle, KDD Cup, ...)</td>
</tr>
<tr>
<td>• Malware classification, credit fraud detection, ads click through rate estimation, sales forecasting, ranking webpages for search, Higgs boson detection,...</td>
</tr>
<tr>
<td>Most deployed ML systems use model ensembles</td>
</tr>
<tr>
<td>• Coefficients chosen manually, with boosting, with bagging, or others</td>
</tr>
</tbody>
</table>

©2021 Carlos Guestrin
What you can do now...

- Identify notion ensemble classifiers
- Formalize ensembles as weighted combination of simpler classifiers
- Outline the boosting framework – sequentially learn classifiers on weighted data
- Describe the AdaBoost algorithm
  - Learn each classifier on weighted data
  - Compute coefficient of classifier
  - Recompute data weights
  - Normalize weights
- Implement AdaBoost to create an ensemble of decision stumps