Decision Trees: Overfitting

CSE 446: Machine Learning
Emily Fox
University of Washington
January 30, 2017

Decision tree recap

For each leaf node, set $\hat{y} =$ majority value
Greedy decision tree learning

- **Step 1:** Start with an empty tree
- **Step 2:** Select a feature to split data
- For each split of the tree:
  - **Step 3:** If nothing more to, make predictions
  - **Step 4:** Otherwise, go to Step 2 & continue (recurse) on this split

Pick feature split leading to lowest classification error
Stopping conditions 1 & 2
Recursion

Scoring a loan application

\[ x_i = (\text{Credit} = \text{poor}, \text{Income} = \text{high}, \text{Term} = 5 \text{ years}) \]

\[ y_i = \text{Safe} \]
Decision trees vs logistic regression: Example

Logistic regression

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
<th>Weight Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_0(x)$</td>
<td>1</td>
<td>0.22</td>
</tr>
<tr>
<td>$h_1(x)$</td>
<td>$x[1]$</td>
<td>1.12</td>
</tr>
<tr>
<td>$h_2(x)$</td>
<td>$x[2]$</td>
<td>-1.07</td>
</tr>
</tbody>
</table>
Depth 1: Split on $x[1]$

```
Depth 2
```

```
Depth 1: Split on $x[1]$
```

```
Depth 2
```
Threshold split caveat

For threshold splits, same feature can be used multiple times

Decision boundaries

Depth 1

Depth 2

Depth 10
Comparing decision boundaries

Logistic Regression
Degree 1 features
Degree 2 features
Degree 6 features

Decision Tree
Depth 1
Depth 3
Depth 10

Degree 2 features

Predicting probabilities with decision trees

Loan status: Safe Risky

Root
18 12

Credit?

excellent
9 2

fair
6 9

poor
3 1

P(y = Safe | x) = \frac{3}{3 + 1} = 0.75
Depth 1 probabilities

Depth 2 probabilities
Comparison with logistic regression

Overfitting in decision trees
What happens when we increase depth?

Training error reduces with depth

<table>
<thead>
<tr>
<th>Tree depth</th>
<th>depth = 1</th>
<th>depth = 2</th>
<th>depth = 3</th>
<th>depth = 5</th>
<th>depth = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training error</td>
<td>0.22</td>
<td>0.13</td>
<td>0.10</td>
<td>0.03</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Decision boundary

Two approaches to picking simpler trees

1. Early Stopping:
   Stop the learning algorithm before tree becomes too complex

2. Pruning:
   Simplify the tree after the learning algorithm terminates
Technique 1: Early stopping

- **Stopping conditions (recap):**
  1. All examples have the same target value
  2. No more features to split on

- **Early stopping conditions:**
  1. Limit tree depth (choose max_depth using validation set)
  2. Do not consider splits that do not cause a sufficient decrease in classification error
  3. Do not split an intermediate node which contains too few data points

Challenge with early stopping condition 1

- Hard to know *exactly* when to stop
- Also, might want some branches of tree to go deeper while others remain shallow
Early stopping condition 2: Pros and Cons

• **Pros:**
  - A reasonable heuristic for early stopping to avoid useless splits

• **Cons:**
  - **Too short sighted:** We may miss out on “good” splits may occur right after “useless” splits
  - Saw this with “xor” example

Two approaches to picking simpler trees

1. **Early Stopping:**
   Stop the learning algorithm before tree becomes too complex

2. **Pruning:**
   Simplify the tree after the learning algorithm terminates

Complements early stopping
Pruning: Intuition
Train a complex tree, simplify later

Complex Tree

Simpler Tree

Pruning motivation

Classification Error

True Error

Simple tree

Complex tree

Simplify after tree is built

Don’t stop too early

Training Error

Tree depth
Scoring trees: Desired total quality format

Want to balance:

i. How well tree fits data
ii. Complexity of tree

Total cost = \text{measure of fit} + \text{measure of complexity}

want to balance

(classification error)
Large # = bad fit to training data

Large # = likely to overfit

Simple measure of complexity of tree

\[ L(T) = \# \text{ of leaf nodes} \]
Balance simplicity & predictive power

Too complex, risk of overfitting

Start

Credit?

excellent
Safe

fair

poor

Income?

Term?

high

Risky

low

Safe

5 years

3 years

Risky

Too simple, high classification error

Start

Income?

Term?

high

Risky

low

Safe

5 years

3 years

Risky

L(T) = 6

L(T) = 1

Balancing fit and complexity

Total cost \( C(T) = Error(T) + \lambda L(T) \)

\( \lambda \) tuning parameter

If \( \lambda = 0 \):
standard decision tree learning

If \( \lambda = \infty \):

no penalty \( \Rightarrow \frac{\text{Total}}{0} \Rightarrow \hat{y} = \text{majority class} \)

If \( \lambda \) in between:

balance fit & complexity
Tree pruning algorithm

Step 1: Consider a split

Tree $T$

Candidate for pruning
Step 2: Compute total cost $C(T)$ of split

$$C(T) = \text{Error}(T) + \lambda \text{L}(T)$$

Step 2: “Undo” the splits on $T_{\text{smaller}}$

$$C(T) = \text{Error}(T) + \lambda \text{L}(T)$$
Prune if total cost is lower: \( C(T_{\text{smaller}}) \leq C(T) \)

\[ C(T) = \text{Error}(T) + \lambda \cdot L(T) \]

\( \lambda = 0.3 \)

<table>
<thead>
<tr>
<th>Tree</th>
<th>Error</th>
<th>#Leaves</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0.25</td>
<td>6</td>
<td>0.43</td>
</tr>
<tr>
<td>( T_{\text{smaller}} )</td>
<td>0.26</td>
<td>5</td>
<td>0.41</td>
</tr>
</tbody>
</table>

Replace split by leaf node? **YES!**

Step 5: Repeat Steps 1-4 for every split

Decide if each split can be “pruned”
Summary of overfitting in decision trees

What you can do now...

- Identify when overfitting in decision trees
- Prevent overfitting with early stopping
  - Limit tree depth
  - Do not consider splits that do not reduce classification error
  - Do not split intermediate nodes with only few points
- Prevent overfitting by pruning complex trees
  - Use a total cost formula that balances classification error and tree complexity
  - Use total cost to merge potentially complex trees into simpler ones
Simple (weak) classifiers are good!

- Logistic regression w. simple features
- Shallow decision trees
- Decision stumps

Low variance. Learning is fast!

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

Classification error vs. Model complexity

- 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
A single classifier

Input: \( x \)

Income \( \geq \$100K? \)

- Yes
  - Safe
- No
  - Risky

Output: \( \hat{y} = f(x) \)
- Either +1 or -1
Ensemble methods: Each classifier “votes” on prediction.

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

- If \( \text{Income} > \$100K \):
  - Yes: \( f_1(x_i) = +1 \)
  - No: \( f_2(x_i) = -1 \)
- Credit history?
  - Safe: \( f_3(x_i) = -1 \)
  - Risky: \( f_4(x_i) = +1 \)
- Savings > $100K?
  - Safe: \( f_3(x_i) = -1 \)
  - Risky: \( f_4(x_i) = +1 \)
- Market conditions?
  - Good:
    - Safe: \( f_3(x_i) = -1 \)
    - Risky: \( f_4(x_i) = +1 \)
  - Bad:
    - Safe: \( f_3(x_i) = -1 \)
    - Risky: \( f_4(x_i) = +1 \)

Combine the predictions:

\[
F(x_i) = \text{sign}(w_1 f_1(x_i) + w_2 f_2(x_i) + w_3 f_3(x_i) + w_4 f_4(x_i))
\]
Prediction with ensemble

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

Combine?

\[ F(x) = \text{sign}(w_1 f_1(x) + w_2 f_2(x) + w_3 f_3(x) + w_4 f_4(x)) \]

<table>
<thead>
<tr>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>( w_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.5</td>
<td>1.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

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:**
    \[ \hat{y} = \text{sign} \left( \sum_{t=1}^{T} \hat{w}_t f_t(x) \right) \]
Boosting

Training a classifier

\[ \hat{y} = \text{sign}(f(x)) \]
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>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>
<tr>
<td>A</td>
<td>$98K</td>
<td>Safe</td>
</tr>
</tbody>
</table>

Income?

$\text{\textgreater} \ 100K$

$\hat{y} = \text{Safe}$

$\leq \ 100K$

$\hat{y} = \text{Safe}$

Boosting = Focus learning on “hard” points

1. **Training data**: Input data used for training.
2. **Learn classifier**: Train a classifier $f(x)$.
3. **Predict** $\hat{y} = \text{sign}(f(x))$.
4. **Evaluate**: Check the performance of $f(x)$.
5. **Learn where $f(x)$ makes mistakes**: Identify points where the classifier performs poorly.

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., $\alpha_i = 2 \Rightarrow$ count point twice

Learning a decision stump on weighted data

<table>
<thead>
<tr>
<th>Credit</th>
<th>Income</th>
<th>$y$</th>
<th>Weight $\alpha$</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

Income?

$>$ $100K$

<table>
<thead>
<tr>
<th>$\hat{y} =$ Safe</th>
<th>$\hat{y} =$ Risky</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.2</td>
</tr>
</tbody>
</table>

$\leq \$100K

<table>
<thead>
<tr>
<th>$\hat{y} =$ Safe</th>
<th>$\hat{y} =$ Risky</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6.5</td>
</tr>
</tbody>
</table>
Learning from weighted data in general

• Usually, learning from weighted data
  - Data point $i$ counts as $\alpha_i$ data points

• E.g., gradient ascent for logistic regression:

$$w_{j+1}^{(t)} \leftarrow w_j^{(t)} + \eta \sum_{i=1}^{N} B_j(x_i) \left( \mathbb{1}[y_i = +1] - P(y = +1 \mid x_i, w^{(t)}) \right)$$

Weigh each point by $\alpha_i$

Boosting = Greedy learning ensembles from data

Training data

Learn classifier

Predict $\hat{y} = \text{sign}(f_1(x))$

Weighted data

Learn classifier & coefficient

Predict $\hat{y} = \text{sign}(w_1 f_1(x) + w_2 f_2(x))$

Higher weight for points where $f_1(x)$ is wrong
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
Weighted classification error

Learned classifier
\[ \hat{y} = + \]

Data point
(Sushi was great, \( \alpha = 1.2 \))

Weight of correct
Weight of mistakes

Sushi was great
Correct!

\( \alpha = 1.2 \)

Food was OK
Mistake!

\( \alpha = 0.5 \)

- Total weight of mistakes:
  \[ \sum_{i=1}^{N} \alpha_i \cdot \mathbb{1} \left( \hat{y}_i \neq y_i \right) \]
- Total weight of all points:
  \[ \sum_{i=1}^{N} \alpha_i \]
- Weighted error measures fraction of weight of mistakes:
  \[ \text{weighted_error} = \frac{\text{total weight of mistakes}}{\text{total weight}} \]

- Best possible value is 0.0
  \[ \text{worst} = 1 \text{, random classifier} = 0.5 \]
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 \( \hat{w}_t \) \( \hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - \text{weighted_error}(f_t)}{\text{weighted_error}(f_t)} \right) \)
  - Recompute weights \( \alpha_i \)

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

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

<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>$\hat{w}_t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>$\hat{w}_t$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Did $f_t$ get $x_i$ right?

---

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:

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

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

$$\alpha_i \leftarrow \begin{cases} 
\alpha_i e^{-\hat{w}_t}, & \text{if } f_t(x_i) = y_i \\
\alpha_i e^{\hat{w}_t}, & \text{if } f_t(x_i) \neq y_i 
\end{cases}$$

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$
AdaBoost example

t = 1: Just learn a classifier on original data
Updating weights $\alpha_i$

Learned decision stump $f_1(x)$

Increase weight $\alpha_i$ of misclassified points

New data weights $\alpha_i$

Boundary

$t=2$: Learn classifier on weighted data

Learned decision stump $f_2(x)$ on weighted data

Weighted data: using $\alpha_i$ chosen in previous iteration
Ensemble becomes weighted sum of learned classifiers

\[
\hat{w}_1 f_1(x) + \hat{w}_2 f_2(x) = 0.61 \hat{w}_1 + 0.53 \hat{w}_2
\]

Decision boundary of ensemble classifier after 30 iterations

training_error = 0
Boosted decision stumps

- Start same weight for all points: $\alpha_i = 1/N$
- For $t = 1, \ldots, T$
  - Learn $f_t(x)$: pick decision stump with lowest weighted training error according to $\alpha_i$
  - Compute coefficient $\hat{w}_t$
  - Recompute weights $\alpha_i$
  - Normalize weights $\alpha_i$
- Final model predicts by:
  $$\hat{y} = \text{sign} \left( \sum_{t=1}^{T} \hat{w}_t f_t(x) \right)$$
Finding best next decision stump $f_t(x)$

Consider splitting on each feature:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Income&gt;$100K?</td>
<td>Safe</td>
<td>Risky</td>
</tr>
<tr>
<td>Credit history?</td>
<td>Risky</td>
<td>Safe</td>
</tr>
<tr>
<td>Savings&gt;$100K?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Market conditions?</td>
<td>Risky</td>
<td>Safe</td>
</tr>
</tbody>
</table>

- $\text{weighted_error} = 0.2$
- $\text{weighted_error} = 0.35$
- $\text{weighted_error} = 0.3$
- $\text{weighted_error} = 0.4$

$\hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - \text{weighted_error}(f_t)}{\text{weighted_error}(f_t)} \right) = 0.69$

Boosted decision stumps

- Start same weight for all points: $\alpha_i = 1/N$
- For $t = 1, \ldots, T$
  - Learn $f_t(x)$: pick decision stump with lowest weighted training error according to $\alpha_i$
  - Compute coefficient $\hat{w}_t$
  - Recompute weights $\alpha_i$
  - Normalize weights $\alpha_i$
- Final model predicts by:
  $$\hat{y} = \text{sign} \left( \sum_{t=1}^{T} \hat{w}_t f_t(x) \right)$$
Updating weights $\alpha_i$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{\frac{\hat{w}_t}{\alpha_i}} = \alpha_i e^{-0.69} = \frac{\alpha_i}{2}, & \text{if } f_t(x_i) = y_i \\ \alpha_i e^{\frac{\hat{w}_t}{\alpha_i}} = \alpha_i e^{0.69} = 2\alpha_i, & \text{if } f_t(x_i) \neq y_i \end{cases}$$

<table>
<thead>
<tr>
<th>Credit</th>
<th>Income</th>
<th>$y$</th>
<th>$\hat{y}$</th>
<th>Previous weight $\alpha$</th>
<th>New weight $\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$130K$</td>
<td>Safe</td>
<td>Safe</td>
<td>0.5</td>
<td>0.5/2 = 0.25</td>
</tr>
<tr>
<td>B</td>
<td>$80K$</td>
<td>Risky</td>
<td>Risky</td>
<td>1.5</td>
<td>0.75</td>
</tr>
<tr>
<td>C</td>
<td>$110K$</td>
<td>Risky</td>
<td>Safe</td>
<td>1.5</td>
<td>2 * 1.5 = 3</td>
</tr>
<tr>
<td>A</td>
<td>$110K$</td>
<td>Safe</td>
<td>Safe</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>$90K$</td>
<td>Safe</td>
<td>Risky</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>$120K$</td>
<td>Safe</td>
<td>Safe</td>
<td>2.5</td>
<td>1.25</td>
</tr>
<tr>
<td>C</td>
<td>$30K$</td>
<td>Risky</td>
<td>Risky</td>
<td>3</td>
<td>1.5</td>
</tr>
<tr>
<td>C</td>
<td>$60K$</td>
<td>Risky</td>
<td>Risky</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>$95K$</td>
<td>Safe</td>
<td>Risky</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>$60K$</td>
<td>Safe</td>
<td>Risky</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>$98K$</td>
<td>Safe</td>
<td>Risky</td>
<td>0.5</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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)

**Amongst most useful ML methods ever created**

- Extremely useful in computer vision
  - Standard approach for face detection, for example

- Used by most winners of ML competitions (Kaggle, KDD Cup, ...)
  - Malware classification, credit fraud detection, ads click through rate estimation, sales forecasting, ranking webpages for search, Higgs boson detection, ...

- Most deployed ML systems use model ensembles
  - Coefficients chosen manually, with boosting, with bagging, or others
What you can do now...

- Identify notion ensemble classifiers
- Formalize ensembles as the 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