Decision Tree: Making a Prediction

Data: decision tree \( t \), input example \( x \)
Result: predicted class

if \( t \) has the form \text{LEAF} (y) then
    return \( y \);
else
    \# \( t.\phi \) is the feature associated with \( t \);
    \# \( t.\text{child}(v) \) is the subtree for value \( v \);
    return \( \text{DTreeTest}(t.\text{child}(t.\phi(x)), x) \);
end

Algorithm 1: \text{DTreeTest}
**Algorithm 2: DTreeTrain**

**Data:** data $D$, feature set $\Phi$

**Result:** decision tree

1. **if** all examples in $D$ have the same label $y$, or $\Phi$ is empty and $y$ is the best guess **then**
   1. return **LEAF**($y$);
2. **else**
   1. **for** each feature $\phi$ in $\Phi$ **do**
      1. partition $D$ into $D_0$ and $D_1$ based on $\phi$-values;
      2. let mistakes($\phi$) = (non-majority answers in $D_0$) + (non-majority answers in $D_1$);
   2. let $\phi^*$ be the feature with the smallest number of mistakes;
   3. return **NODE**($\phi^*$, $\{0 \rightarrow \text{DTreeTrain}(D_0, \Phi \setminus \{\phi^*\}), 1 \rightarrow \text{DTreeTrain}(D_1, \Phi \setminus \{\phi^*\})\}$);

**Algorithm 2: DTreeTrain**
Danger: Overfitting

- Error rate (lower is better)
- Depth of the decision tree
- Training data
- Unseen data
- Overfitting
Some Notation

- Let $\ell$ be a loss function; $\ell(y, \hat{y})$ is what we lose by outputting $\hat{y}$ when $y$ is the correct output. For classification:

$$\ell(y, \hat{y}) = [y \neq \hat{y}]$$
Some Notation

- Let $\ell$ be a loss function; $\ell(y, \hat{y})$ is what we lose by outputting $\hat{y}$ when $y$ is the correct output. For classification:

$$\ell(y, \hat{y}) = [y \neq \hat{y}]$$

- Let $D(x, y)$ define the true probability of input/output pair $(x, y)$, in “nature.” We never “know” this distribution.
Some Notation

- Let $\ell$ be a loss function; $\ell(y, \hat{y})$ is what we lose by outputting $\hat{y}$ when $y$ is the correct output. For classification:

$$\ell(y, \hat{y}) = \mathbb{I}[y \neq \hat{y}]$$

- Let $D(x, y)$ define the true probability of input/output pair $(x, y)$, in “nature.” We never “know” this distribution.

- The training data $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ are assumed to be i.i.d. samples from $D$. 


Some Notation

- Let $\ell$ be a loss function; $\ell(y, \hat{y})$ is what we lose by outputting $\hat{y}$ when $y$ is the correct output. For classification:

$$\ell(y, \hat{y}) = [y \neq \hat{y}]$$

- Let $D(x, y)$ define the true probability of input/output pair $(x, y)$, in “nature.” We never “know” this distribution.

- The training data $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ are assumed to be i.i.d. samples from $D$.

- The space of classifiers we’re considering is $F$; $f$ is a classifier from $F$, chosen by our learning algorithm.
Overfitting, More Formally

- Classifier $f$’s average loss on **training data**:

$$\hat{\epsilon}(f) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n))$$

- $f$ has overfit $D$ when:

$$\exists f' \in F \text{ s.t. } \hat{\epsilon}(f) < \hat{\epsilon}(f') \land \epsilon(f') < \epsilon(f)$$

This is the fundamental problem of ML.
Overfitting, More Formally

- Classifier $f$’s average loss on **training data**:

$$\hat{\epsilon}(f) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n))$$

- Classifier $f$’s **true** expected loss:

$$\epsilon(f) = \sum_{(x, y)} D(x, y) \cdot \ell(y, f(x)) = \mathbb{E}_{(x, y) \sim D}[\ell(y, f(x))]$$
Overfitting, More Formally

- Classifier $f$’s average loss on training data:

$$\hat{\epsilon}(f) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n))$$

- Classifier $f$’s true expected loss:

$$\epsilon(f) = \sum_{(x,y)} D(x,y) \cdot \ell(y, f(x)) = \mathbb{E}_{(x,y) \sim D}[\ell(y, f(x))]$$

- $f$ has overfit $D$ when:

$$\exists f' \in \mathcal{F} \text{ s.t. } \hat{\epsilon}(f) < \hat{\epsilon}(f') \land \epsilon(f') < \epsilon(f)$$

This is the fundamental problem of ML.
Inductive, Supervised Machine Learning

- Input: loss function $\ell$ and training data $D$ drawn i.i.d. from $\mathcal{D}$
- Output: $f$ such that $\epsilon(f)$ is low over $\mathcal{D}$, with respect to $\ell$

Never forget that $\epsilon(f) \neq \hat{\epsilon}(f)$.

Is your training data $D$ really drawn from $\mathcal{D}$?
Back to decision trees . . .
Avoiding Overfitting by Stopping Early

- Set a maximum tree depth $d_{\text{max}}$. 
Avoiding Overfitting by Stopping Early

- Set a maximum tree depth $d_{max}$.

- Only consider splits that decrease error by at least some $\Delta$. 

Avoiding Overfitting by Stopping Early

- Set a maximum tree depth $d_{max}$.

- Only consider splits that decrease error by at least some $\Delta$.

- Only consider splitting a node with more than $N_{min}$ examples.
Avoiding Overfitting by Stopping Early

- Set a maximum tree depth $d_{max}$.

- Only consider splits that decrease error by at least some $\Delta$.

- Only consider splitting a node with more than $N_{min}$ examples.

In each case, we have a hyperparameter $(d_{max}, \Delta, N_{min})$, which you should tune on development data.
Avoiding Overfitting by Pruning

- Build a big tree (i.e., let it overfit), call it $t_0$. 
Avoiding Overfitting by Pruning

▶ Build a big tree (i.e., let it overfit), call it $t_0$.

▶ For $i \in \{1, \ldots, |t_0|\}$: greedily choose a set of sibling-leaves in $t_{i-1}$ to collapse that increases error the least; collapse to produce $t_i$. 
Avoiding Overfitting by Pruning

- Build a big tree (i.e., let it overfit), call it $t_0$.

- For $i \in \{1, \ldots, |t_0|\}$: greedily choose a set of sibling-leaves in $t_{i-1}$ to collapse that increases error the least; collapse to produce $t_i$.

(Alternately, collapse the split whose contingency table is least surprising under chance assumptions.)
Avoiding Overfitting by Pruning

- Build a big tree (i.e., let it overfit), call it $t_0$.

- For $i \in \{1, \ldots, |t_0|\}$: greedily choose a set of sibling-leaves in $t_{i-1}$ to collapse that increases error the least; collapse to produce $t_i$.

  (Alternately, collapse the split whose contingency table is least surprising under chance assumptions.)

- Choose the $t_i$ that performs best on development data.
More Things to Know

▶ Instead of using the number of mistakes, we often use information-theoretic quantities to choose the next feature.
More Things to Know

- Instead of using the number of mistakes, we often use information-theoretic quantities to choose the next feature.

- For continuous-valued features, we use thresholds, e.g., \( \phi(x) \leq \tau \).
  In this case, you must choose \( \tau \).
  If the sorted values of \( \phi \) are \( \langle v_1, v_2, \ldots, v_N \rangle \), you only need to consider
  \[
  \tau \in \left\{ \frac{v_n + v_{n+1}}{2} \right\}_{n=1}^{N-1} \quad \text{(midpoints between consecutive feature values)}.
  \]
More Things to Know

- Instead of using the number of mistakes, we often use information-theoretic quantities to choose the next feature.

- For continuous-valued features, we use thresholds, e.g., \( \phi(x) \leq \tau \). In this case, you must choose \( \tau \).
  
  If the sorted values of \( \phi \) are \( \langle v_1, v_2, \ldots, v_N \rangle \), you only need to consider
  
  \[
  \tau \in \left\{ \frac{v_n + v_{n+1}}{2} \right\}_{n=1}^{N-1} \quad \text{(midpoints between consecutive feature values)}.
  \]

- For continuous-valued outputs, what value makes sense as the prediction at a leaf? What loss should we use instead of \( [y \neq \hat{y}] \)?
Machine Learning (CSE 446):
Limits of Learning

Sham M Kakade

© 2018

University of Washington
skakade@cs.washington.edu
The Bayes Optimal Classifier

\[ f^{(BO)}(x) = \arg\max_y D(x, y) \]
The Bayes Optimal Classifier

\[ f^{(BO)}(x) = \arg\max_y D(x, y) \]

**Theorem:** The Bayes optimal classifier achieves minimal zero/one error \((\ell(y, \hat{y}) = [y \neq \hat{y}])\) of any deterministic classifier.
Consider (deterministic) $f'$ that claims to be better than $f^{(BO)}$ and $x$ such that $f^{(BO)}(x) \neq f'(x)$. 
Proof

Consider (deterministic) $f'$ that claims to be better than $f^{(BO)}$ and $x$ such that $f^{(BO)}(x) \neq f'(x)$. 

Probability that $f'$ makes an error on this input: $\left(\sum_y D(x, y)\right) - D(x, f'(x))$. 
Proof

Consider (deterministic) $f'$ that claims to be better than $f^{(BO)}$ and $x$ such that $f^{(BO)}(x) \neq f'(x)$.

Probability that $f'$ makes an error on this input: \( \left( \sum_y D(x, y) \right) - D(x, f'(x)) \).

Probability that $f^{(BO)}$ makes an error on this input: \( \left( \sum_y D(x, y) \right) - D(x, f^{(BO)}(x)) \).
Proof

Consider (deterministic) $f'$ that claims to be better than $f^{(BO)}$ and $x$ such that $f^{(BO)}(x) \neq f'(x)$.

Probability that $f'$ makes an error on this input: $\left( \sum_y \mathcal{D}(x, y) \right) - \mathcal{D}(x, f'(x))$.

Probability that $f^{(BO)}$ makes an error on this input: $\left( \sum_y \mathcal{D}(x, y) \right) - \mathcal{D}(x, f^{(BO)}(x))$.

By definition,

$$\mathcal{D}(x, f^{(BO)}(x)) = \max_y \mathcal{D}(x, y) \geq \mathcal{D}(x, f'(x))$$

$$\Rightarrow \left( \sum_y \mathcal{D}(x, y) \right) - \mathcal{D}(x, f^{(BO)}(x)) \leq \left( \sum_y \mathcal{D}(x, y) \right) - \mathcal{D}(x, f'(x))$$
Proof

Consider (deterministic) $f'$ that claims to be better than $f^{(BO)}$ and $x$ such that $f^{(BO)}(x) \neq f'(x)$.

Probability that $f'$ makes an error on this input: $\left( \sum_y D(x, y) \right) - D(x, f'(x))$.

Probability that $f^{(BO)}$ makes an error on this input: $\left( \sum_y D(x, y) \right) - D(x, f^{(BO)}(x))$.

By definition,

$$ D(x, f^{(BO)}(x)) = \max_y D(x, y) \geq D(x, f'(x)) $$

$$ \Rightarrow \left( \sum_y D(x, y) \right) - D(x, f^{(BO)}(x)) \leq \left( \sum_y D(x, y) \right) - D(x, f'(x)) $$

This must hold for all $x$. Hence $f'$ is no better than $f^{(BO)}$. 


You cannot do better than $\epsilon(f^{BO})$. 