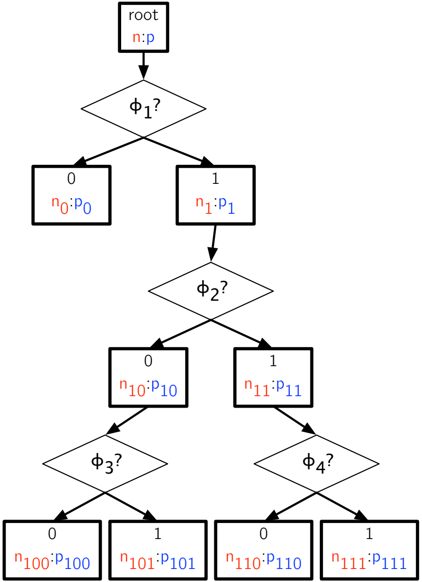
Machine Learning (CSE 446): Decision Trees (continued)

Noah Smith

University of Washington
nasmith@cs.washington.edu

October 2, 2017

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.child(v) is the subtree for value v;
    return $\text{DTreeTest}(t.\text{child}(t.\phi(x)), x)$;
end

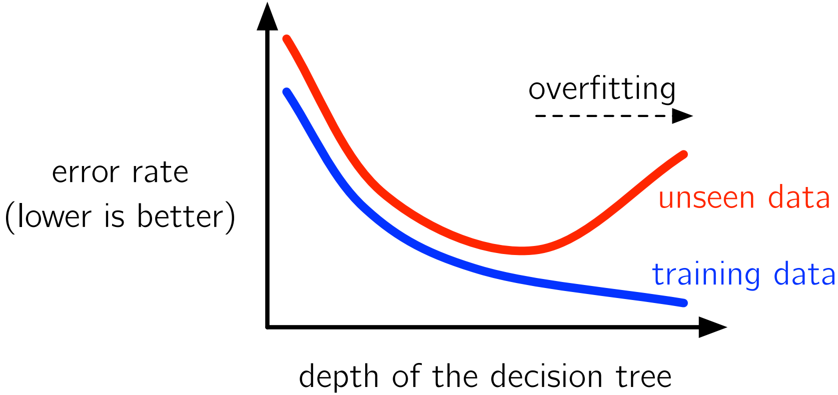
Algorithm 1: DTreeTest

Greedily Building a Decision Tree (Binary Features)

Data: data D, feature set Φ
Result: decision tree
if *all examples in* D *have the same label* y*, or* Φ *is empty and* y *is the best guess* **then**
  return $\textsc{Leaf}(y)$;
else
  for *each feature* ϕ *in* Φ **do**
    partition D into D_0 and D_1 based on ϕ-values;
    let mistakes(ϕ) $=$ (non-majority answers in D_0) $+$ (non-majority answers in D_1);
  end
  let ϕ^* be the feature with the smallest number of mistakes;
  return $\textsc{Node}(\phi^*, \{0 \rightarrow \textsc{DTreeTrain}(D_0, \Phi \setminus \{\phi^*\}), 1 \rightarrow \textsc{DTreeTrain}(D_1, \Phi \setminus \{\phi^*\})\})$;
end

Algorithm 2: $\textsc{DTreeTrain}$

Danger: Overfitting



Some Notation

- Let ℓ 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 ℓ 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 $\mathcal{D}(x, y)$ define the true probability of input/output pair (x, y), in "nature." **We never "know" this distribution.**

Some Notation

- Let ℓ 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 $\mathcal{D}(x, y)$ define the true probability of input/output pair (x, y), in "nature." **We never "know" this distribution.**
- The training data $D = \langle (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N) \rangle$ are assumed to be i.i.d. samples from $\mathcal{D}$.

Some Notation

- Let ℓ 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 $\mathcal{D}(x, y)$ define the true probability of input/output pair (x, y), in "nature." **We never "know" this distribution.**
- The training data $D = \langle (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N) \rangle$ are assumed to be i.i.d. samples from $\mathcal{D}$.
- The space of classifiers we're considering is $\mathcal{F}$; f is a classifier from $\mathcal{F}$, chosen by our learning algorithm.

Overfitting, More Formally

- Classifier f's average loss on **training data**:

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

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:

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

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:

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

- f has overfit D when:

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

This is the fundamental problem of ML.

Inductive, Supervised Machine Learning

- Input: loss function ℓ 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 ℓ

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_{max}.

Avoiding Overfitting by Stopping Early

- Set a maximum tree depth d_{max}.

- Only consider splits that decrease error by at least some Δ.

Avoiding Overfitting by Stopping Early

- Set a maximum tree depth d_{max}.
- Only consider splits that decrease error by at least some Δ.
- 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 Δ.
- 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 threshholds, e.g., $\phi(x) \le \tau$.
  In this case, you must choose τ.
  If the sorted values of ϕ are $\langle v_1, v_2, \ldots, v_N \rangle$, you only need to consider $\tau \in \left\{ \frac{v_n + v_{n+1}}{2} \right\}_{n=1}^{N-1}$ (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 threshholds, e.g., $\phi(x) \leq \tau$.
  In this case, you must choose τ.
  If the sorted values of ϕ are $\langle v_1, v_2, \ldots, v_N \rangle$, you only need to consider $\tau \in \left\{ \frac{v_n + v_{n+1}}{2} \right\}_{n=1}^{N-1}$ (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

Noah Smith

University of Washington
nasmith@cs.washington.edu

October 2, 2017

The Bayes Optimal Classifier

$$f^{(\mathsf{BO})}(x) = \operatorname*{argmax}_y \mathcal{D}(x, y)$$

The Bayes Optimal Classifier

$$f^{(\mathsf{BO})}(x) = \underset{y}{\operatorname{argmax}}\, \mathcal{D}(x, y)$$

Theorem: The Bayes optimal classifier achieves minimal zero/one error ($\ell(y, \hat{y}) = [\![y \neq \hat{y}]\!]$) of any deterministic classifier.

Proof

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

Proof

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

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

Proof

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

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

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

Proof

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

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

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

By definition,

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

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

Proof

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

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

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

By definition,

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

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

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

One Limit of Learning

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