Machine Learning (CSE 446): Generalization and Overfitting

Sham M Kakade

University of Washington
cse446-staff@cs.washington.edu

Announcements

- Turn in Certification file that you read the website.
- HW0.
- Midterm date: Mon, Feb 11.
- Today:
  - Decision Trees, Generalization, and Overfitting.
  - training/ and development/validation sets

A Toy Data Set

Data derived from https://archive.ics.uci.edu/ml/datasets/Auto+MPG

mpg; cylinders; displacement; horsepower; weight; acceleration; year; origin

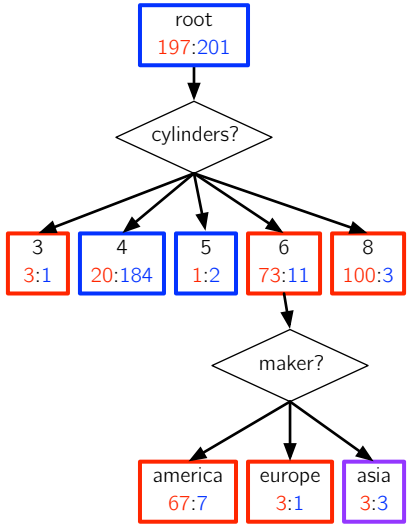
18.0	8	307.0	130.0	3504.	12.0	70	1
15.0	8	350.0	165.0	3693.	11.5	70	1
18.0	8	318.0	150.0	3436.	11.0	70	1
16.0	8	304.0	150.0	3433.	12.0	70	1
17.0	8	302.0	140.0	3449.	10.5	70	1
15.0	8	429.0	198.0	4341.	10.0	70	1
14.0	8	454.0	220.0	4354.	9.0	70	1
14.0	8	440.0	215.0	4312.	8.5	70	1
14.0	8	455.0	225.0	4425.	10.0	70	1
15.0	8	390.0	190.0	3850.	8.5	70	1
15.0	8	383.0	170.0	3563.	10.0	70	1
14.0	8	340.0	160.0	3609.	8.0	70	1
15.0	8	400.0	150.0	3761.	9.5	70	1
14.0	8	455.0	225.0	3086.	10.0	70	1
24.0	4	113.0	95.00	2372.	15.0	70	3
22.0	6	198.0	95.00	2833.	15.5	70	1
18.0	6	199.0	97.00	2774.	15.5	70	1
21.0	6	200.0	85.00	2587.	16.0	70	1
27.0	4	97.00	88.00	2130.	14.5	70	3
26.0	4	97.00	46.00	1835.	20.5	70	2
25.0	4	110.0	87.00	2672.	17.5	70	2
24.0	4	107.0	90.00	2430.	14.5	70	2

Input: a row in this table; "features" are columns.

Goal: predict whether mpg is < 23 ("bad" $= 0$) or above ("good" $=$ 1) given other attributes (other columns).

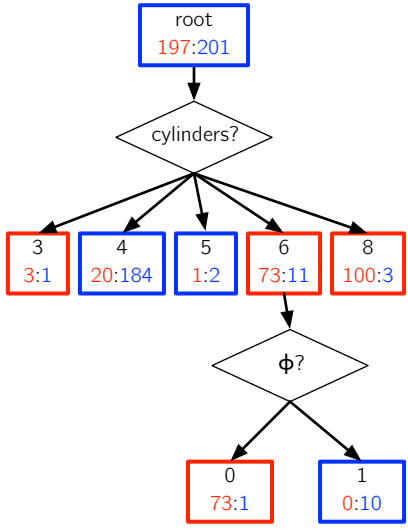
201 "good" and 197 "bad"; guessing the most frequent class (good) will get 50.5% accuracy.

Decision Tree Example



Error reduction compared to the cylinders stump?

Decision Tree Example



Error reduction compared to the cylinders stump?

A Candidate Greedy Algorithm (pseudo-code)

A decision tree $\mathcal{T} : \mathcal{X} \to \mathcal{Y}$:

- The nodes in the tree are associated with a feature ϕ_i.
- The associated decision rule for each feature either: 1) branches (based on the value of the feature) or 2) outputs a prediction.

An (iterative) greedy algorithm:

1. Try each candidate feature ϕ_i with different candidate parent nodes, compute the error reduction. Record the feature& parent with the largest reduction in error.
2. Update $\mathcal{T}$: create a new node using this feature & parent.
3. Stop if "some criterion" is met. Else go to step 1.

What should be our stopping criterion? Will this work?

What could go wrong?

Remember, we care about doing well on 'new' data. Suppose we keep going until the training error no longer drops?

- Suppose we have "many" possible features?

- Suppose a single feature has many possible values ?
  (e.g.how could we split on a continuous feature?)

Let's back up...

- Suppose we chose some hypothesis f before we see the training data, and we want to estimate the true loss of f?
- Is training error of f a good estimate of the true loss of f?

$$\widehat{\epsilon}(f) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{y_i \neq f(x_i)\}$$

  - Is $\widehat{\epsilon}(f)$ an unbiased of the true loss??

$$E[\widehat{\epsilon}(f)] = \epsilon(f)$$

  - What about the variance of $\widehat{\epsilon}(f)$?

$$\mathrm{Var}(\widehat{\epsilon}(f)) \leq \frac{1}{4n}$$

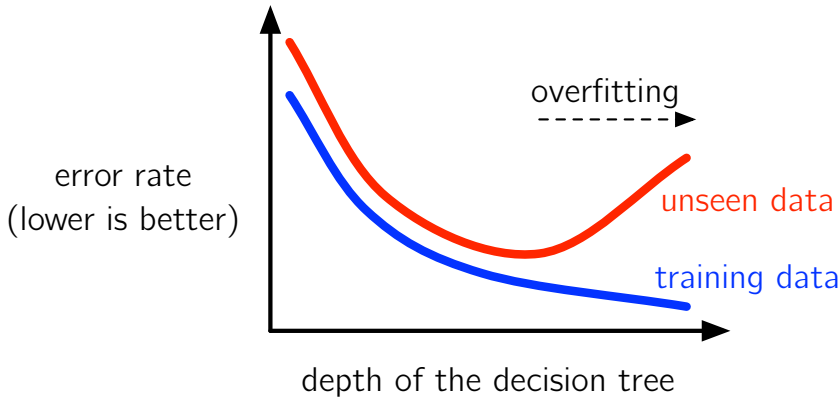
  - If the CLT starts to kick in (for N 'reasonably' large, it is should), then with probability greater than 95% we should have that

$$|\widehat{\epsilon}(f)] - \epsilon(f)| \leq 2\sqrt{\frac{1}{4n}} = \frac{1}{\sqrt{n}}$$

(due that with 95% chance a sample from a Gaussian will be within two standard deviations of its mean).

Danger: Overfitting

Let $\widehat{f}_t$ be our decision tree after we have added t nodes. Should we expect $\widehat{\epsilon}(\widehat{f}_t)$ to be close to $\epsilon(\widehat{f}_t)$? Why?



Generalization error

- We are choose $\widehat{f_t}$ based on our training data AND we are estimating the quality of $\widehat{f_t}$ on this same set of points.
- The **generalization error** is often referred to as the difference between the training error of $\hat{f}$ and the expected error of $\hat{f}$:

$$\hat{\epsilon}(\hat{f}) - \epsilon(\hat{f})$$

- rule of thumb: the 'more' we change $\widehat{f}$ based on our training set the larger the "generalization error" is.
- The fundamental problem of ML is we would like **both**:
  - our training error, $\hat{\epsilon}(\hat{f})$, to be small
  - our generalization error to be small
- It is usually easy to get one of these two to be small.

Ways to check/prevent for overfitting

- Take our dataset $\langle (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N) \rangle$ and break it up into (two or) three datasets.
- Make a training set with "most" of the data.
- Make a **dev**-elopment set (sometimes called a **val**-idation set) with some of it: use this to 'tune' parameters, e.g. when to stop.
- Make a **test set** with some of it: use this only to estimate the the true error.

Avoiding Overfitting by Stopping Early

- Set a maximum tree depth d_{max}.
  (also need to set a maximum width w)
- 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** $(d_{max}, w, \Delta, N_{min})$, which we should **tune** on our dev set.

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.