Machine Learning (CSE 446): Train, Dev, and Test Sets

Sham M Kakade

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

Announcements

- HW0 due.
- HW1 posted this week (due in a weeks time).
- Today:
  - Model complexity; parameters; and hyperparameters
  - Training/ 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.

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?

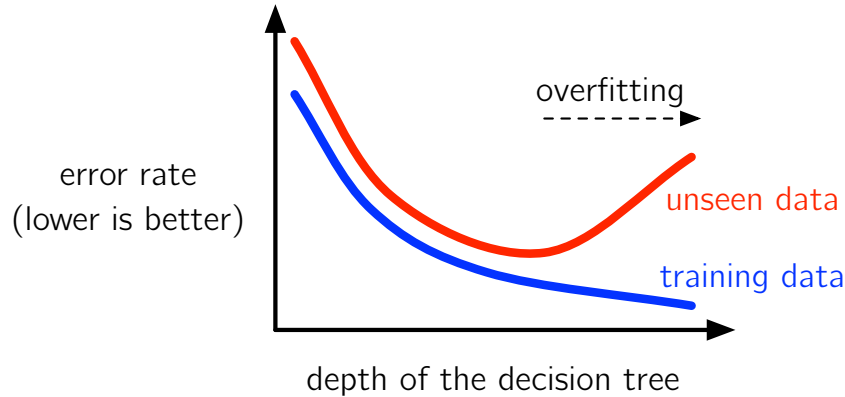
Parameter choices

- How can we use a real valued feature $x[i]$, where i is the coordinate of the vector x (e.g. horsepower)?
  - a binary: choose a z, and let $\phi(x) = \mathbf{1}\{x[i] > z\}$
  - a k-ary feature ("bucketing"): set $\phi(x) = j$ (for $j \in \{0, 1, 2, \ldots k-1\}$ if $z_j \leq x[i] \leq z_{j+1}$

  How to choose z/the "buckets"?
- other choices: the depth of the tree? the width of the tree? the total number of nodes?

Danger: Overfitting

- the 'x-axis' is typically something like our 'model complexity' or 'how long we run our algorithm'
- parameters: some parameter choices make sense to fit on the training set (e.g. z)
- hyper-parameters: some don't make any sense (e.g. 'depth' of the tree).
- How should we fit these parameters? When should we stop our algorithm?



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.

Model Complexity and Generalization error

- Suppose we choose our hypothesis among only a *finite* set of hypothesis $\{f_1, \dots f_K\}$ (the set of things we choose from is our 'hypothesis class').
- Suppose our algorithm chooses the $\widehat{f}$ which is the best on our training error (sometimes called 'empirical risk minimization').
- **generalization error:**

$$\text{gen. error} = |\hat{\epsilon}(\hat{f}) - \epsilon(\hat{f})|$$

  Remember: we want both small training error and small generalization error.
- Overfitting: we might be choosing $\widehat{f}$ due to that (by chance) $\widehat{f}$ ended up looking much better than it actually is. The more things we try (the larger K is the more likely it is one ends up looking good just due to chance.)
- With 'high probability' (say probability greater than 95%), we will have that:

$$\text{gen. error} \leq \sqrt{\frac{\log(K)}{n}}$$

- Crudely: Logarithmic dependence on K is very mild.
  But note that $\log(K)$ scales linearly in the depth of the tree. (since the K scales exponentially in the depth).