# Machine Learning (CSE 446): (continuation of overfitting &) Limits of Learning

Sham M Kakade



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

# Announcement

read CIML

- Qz section tomo: (basic) probability and linear algebra review
- Today:
  - review
  - some limits of learning

# Review

# The "i.i.d." Supervised Learning Setup

- Let $\ell$ be a **loss function**; $\ell(y, \hat{y})$ is our loss by predicting $\hat{y}$ when $y$ is the correct output.
- Let $\mathcal{D}(x, y)$ define the (unknown) underlying 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 **identical, independently, distributed** (i.i.d.) samples from $\mathcal{D}$.
- We care about our **expected error (i.e. the expected loss, the "true" loss, ...)** with regards to the underlying distribution $\mathcal{D}$.
- Goal: find a **hypothesis** which as has "low" expected error, using the training set.

$\hat{y} = f(x)$ $\quad \sum_{x,y} D(x,y) = 1$

$D(x,y) = \Pr($observing the pair $(x,y))$

# Training error

expected value of Heads

$= p \cdot 1 + (1-p) \cdot 0 = p$

- The **training error** of hypothesis $f$ is $f$'s average error on the **training data**:

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

loss on the n-th pair

- In contrast, classifier $f$'s **true** expected loss:

by def

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

prob. loss of example

- Idea: Use the training error $\hat{\epsilon}(f)$ as an empirical approximation to $\epsilon(f)$. **And hope that this approximation is good!**
- For any fixed $f$, the training error is an unbiased estimate of the true error.

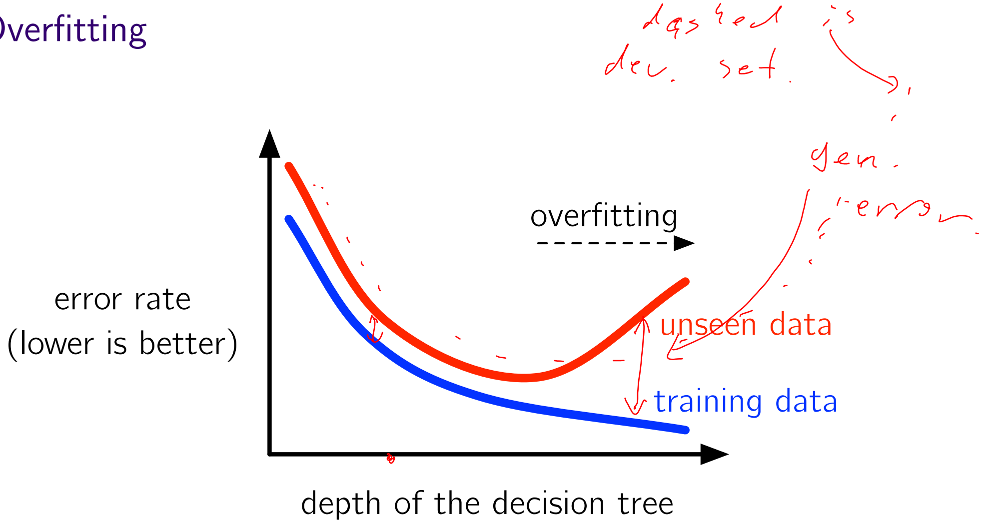
# Overfitting: this is the fundamental problem of ML

- Let $\hat{f}$ be the output of training algorithm.
  - The training error of $\hat{f}$ is (almost) never an unbiased estimate of the true error.
  - It is usually a **gross underestimate**.
- The **generalization error** of our algorithm is its **true error - training error**:

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

- Overfitting, more formally: large generalization error means we have **overfit**.
- We would like **both**:
  - our training error, $\hat{\epsilon}(\hat{f})$, to be small
  - our generalization error to be small
- If both occur, then we have low expected error :)
  - It is usually easy to get one of these two to be small.

# Danger: Overfitting



# Today's Lecture

# Test sets and Dev. Sets

- use **test set**, i.i.d. data sampled $\mathcal{D}$, to estimate the **expected error**.
  - **Don't touch your test data to learn!**
    not for hyperparam tuning, not for modifying your hypothesis!
  - Keep your test set to always give you accurate (and unbiased) estimates of how good your hypothesis is.
- **Hyperparameters** are params of our algorithm/pseudo-code
  - sometimes hyperparameters monotonically make our training error lower
    e.g. decision tree maximal width and maximal depth.
- How do we set hyperparams?
  For some hyperparams:
  - make a **dev set**, i.i.d. from $\mathcal{D}$ (hold aside some of your training set)
  - learn with training set (by trying different hyperparams); then check on your dev set.

# Example: Avoiding Overfitting by "Early Stopping" in Decision Trees

- 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 $\Delta$.
- 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 you should **tune** on **development data**.

# One Limit of Learning: The “No Free Lunch Theorem”

need lots of examples.

small

- We want a learning algorithm which learns very quickly!
- “No Free Lunch Theorem”: (Informally) any learning algorithm that learns with very training set size on one class of problems, must do much worse on another class of problems.
- **inductive bias:** But, we do want to bias our algorithms in certain ways. Let’s see…

# An Exercise

Following **?**, chapter 2.

Class A

Class B

# An Exercise

## Following ?, chapter 2.

Test

AA AB

BB BA

This correct!

# Inductive Bias

- Just as *you* had a tendency to focus on a certain type of function $f$, machine learning algorithms correspond to classes of functions ($\mathcal{F}$) and preferences within the class.
- You want your algorithm to be "biased" towards the correct classifier.
  BUT this means it must do worse on other problems.
- Example Bias: shallow decision trees: "use a small number of features" favors one type of bias.

# Another Limit of Learning: The Bayes Optimal Hypothesis

- The best you could hope to do:

$$f^{(\mathsf{BO})}(x) = \operatorname*{argmin}_{f(x)} \epsilon(f)$$

**You cannot obtain lower loss than** $\epsilon(f^{\mathbf{BO}})$.

- Example: Let's consider classification:
**Theorem:** For classification (binary or multi-class), the Bayes optimal classifier is:

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

and it achieves minimal zero/one error ($\ell(y, \hat{y}) = [\![y \neq \hat{y}]\!]$) of any classifier.

# Proof

- Consider (deterministic) $f'$ that claims to be better than $f^{(\text{BO})}$ and $x$ such that $f^{(\text{BO})}(x) \neq f'(x)$.
- Probability that $f'$ makes an error on this input: $\left(\sum_y \mathcal{D}(x,y)\right) - \mathcal{D}(x, f'(x))$.
- Probability $f^{(\text{BO})}$ makes an error on this input: $\left(\sum_y \mathcal{D}(x,y)\right) - \mathcal{D}(x, f^{(\text{BO})}(x))$.
- By definition,

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

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

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

# The Bayes Optimal Hypothesis for the Square Loss

- For the quadratic loss and real valued $y$:

$$\epsilon(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(y - f(x))^2$$

- **Theorem:** The Bayes optimal hypothesis for the square loss is:

$$f^{(\mathsf{BO})}(x) = \mathbb{E}[y|x]$$

(where the conditional expectation is with respect to $\mathcal{D}$).

# Unavoidable Error

- Noise in the features (we don't want to "fit" the noise!)
- Insufficient information in the available features (e.g., incomplete data)
- No single correct label (e.g., inconsistencies in the data-generating process)

These have nothing to do with your choice of learning algorithm.

# General Recipe

The cardinal rule of machine learning: **Don't touch your test data.**

If you follow that rule, this recipe will give you accurate information:

1. Split data into training, development, and test sets.
2. For different hyperparameter settings:
   2.1 Train on the training data using those hyperparameter values.
   2.2 Evaluate loss on development data.
3. Choose the hyperparameter setting whose model achieved the lowest development data loss.
   Optionally, retrain on the training and development data together.
4. Evaluate that model on test data.

# Design Process for ML Applications

| | | *example* |
|---|---|---|
| 1 | real world goal | increase revenue |
| 2 | mechanism | show better ads |
| 3 | learning problem | will a user who queries $q$ click ad $a$? |
| 4 | data collection | interaction with existing system |
| 5 | collected data | query $q$, ad $a$, $\pm$click |
| 6 | data representation | ($q$ word, $a$ word) pairs |
| 7 | select model family | decision trees up to 20 |
| 8 | select training/dev. data | September |
| 9 | train and select hyperparameters | single decision tree |
| 10 | make predictions on test set | October |
| 11 | evaluate error | zero-one loss ($\pm$click) |
| 12 | deploy | $? |