Lec Mon

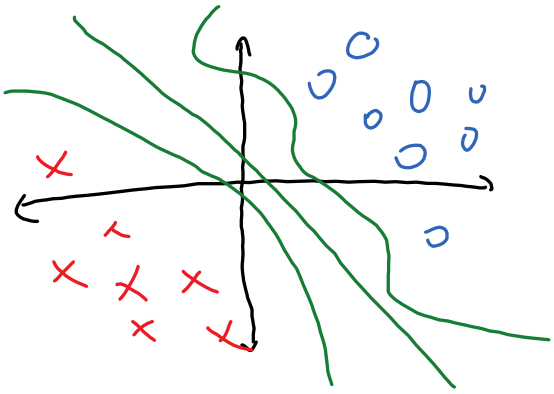
Monday, October 8, 2018 4:05 PM

(some review from last lecture)

Views of Learning

- Learning is the removal of our remaining uncertainty
- Learning requires guessing a good, small hypothesis class
- Our hypothesis space could be wrong (too simple, too complicated)
    - and our model not perform well

Ex:



Any function separating red from blue could be in our hypothesis space

Key Issue: Representation
How we represent data (features) can dramatically impact the quality and simplicity of our model

How to learn

Linear Function

- Challenges
    - hypothesis space contains infinite # functions
    - several of those functions will be consistent with the data
- Possible Algorithm: Local Search
    - start with a linear threshold function
    - see how well you are doing
    - correct your function
    - repeat until you converge

The search can be done bin a better way. Let's use math.
How can we model this as an optimization problem?

General Framework for Learning

- Problem Setting
    - Set of possible instances X
    - Set of possible labels Y
    - Unknown target function $f: X \rightarrow Y$
    - Set of function hypotheses $H = \{h \mid h: X \rightarrow Y\}$
- Input: Training instances drawn from data generating distribution *p*
- Output: A Hypothesis *h* in *H* that best approximates *f*.
    - h should do well on future instances. How do we measure how well it does?
    - formally, *h* should have low expected (test) loss/Risk:

$$E_{(x,y)\sim p}[L(y, h(x)] = \sum_{x,y} p(x,y) L(y, h(x))$$

Expected value of the loss (difference between our hypothesis and the actual function y)

p is the probability that (x, y) shows up

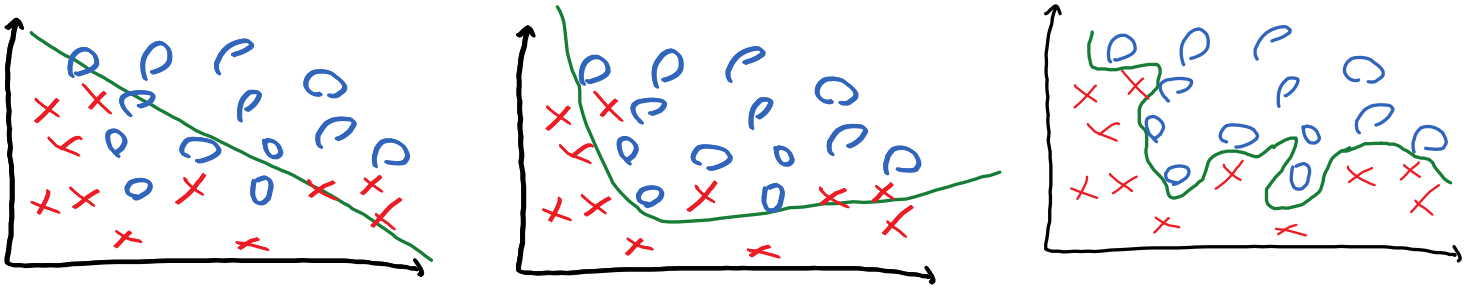
training error/empirical risk:

$$\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))$$

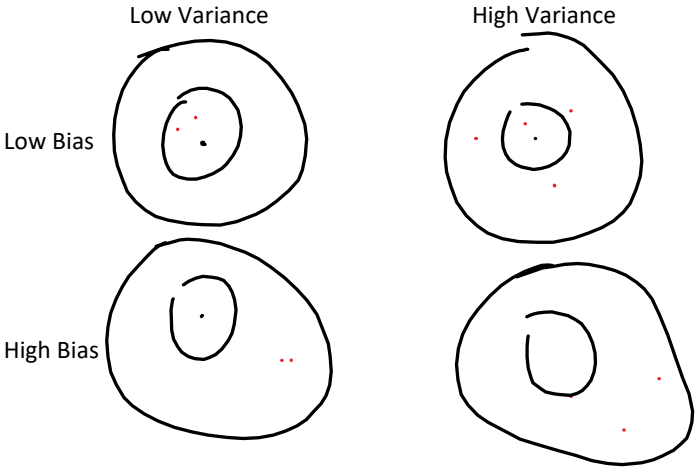
Challenge: Minimization is in general a NP-hard problem
Algorithmic view of Learning: an optimization problem

- We want to minimize the Loss Function
- There are many loss functions we can define
  - eg for classification, it might be 0 if guessed correctly, 1 o.w.
  - but again in general it is hard to minimize
  - we might try to minimize another function, that acts as an upper bound to our loss function, if it's easier to minimize. This would be an approximation, but at least it's something

Underfitting - model is too simple and doesn't generalize well enough
Overfitting - when we have an overly complicated model that fits our training data very well, but won't generalize to other instances
Ex:



Bias vs. Variance (illustrated with targets)



How to prevent overfitting?

- use a less expressive model
  - like a linear model
- adding regularization (we'll cover this later)
- data perturbation
- stop optimization process earlier
  - sounds bad in theory but works well in practice

Machine Learning Practice - putting it all together

Developing ML algorithms

1. collect data
2. design your algorithm
3. verify your algorithm, tune hyperparameters
    a. option 1: split data to Train/Dev
    b. option 2: use Cross-Validation
4. retrain the model using the best parameter on the whole dataset
5. deploy your model on real test data

Ex: simple regression problem
$f: X \to Y$
$x \in R$
$y \in R$
Question 1: How do we pick the hypothesis space?

- could use degree M polynomials

There are 2 types of parameters

- hyper-parameters - needs to be tuned
    - eg what degree of polynomial to use
    - how do we decide?
- model parameters that can be learned from the training set

How can we evaluate hyper parameters?
We can split our initial data set into two - training and development (dev). Training is used to train the model, and dev is used to evaluate the model. Train ~70-90%

Train	Dev	Test

Train, Dev $\subset$ original dataset

What if I don't have enough data? then dev set will be too small

Instead of splitting into Train and Dev, we could use:
n-fold Cross Validation - split data into n partitions, and treat each of those partitions as a separate dev set. train n models, and test against all dev sets

<u>Machine Learning Libraries</u>
Macine Learning Toolbox:

- Scikit learn (python)
- Weka (java)

Deep Learning Models

- Pytorch
- DyNet
- Tensorflow

Big Data

- Apache Spark