Overfitting

Machine Learning – CSE546
Sham Kakade
University of Washington

October 6, 2016

Announcements:

- HW1 posted
- Today:
  - Review: bias variance
  - Overfitting, overfitting overfitting
  - Ridge regression
  - Checking for overfitting: cross validation

Review: Bias-Variance Tradeoff

Machine Learning – CSE546
Sham Kakade
University of Washington

Oct 4, 2016

Bias-Variance tradeoff – Intuition

- Model too "simple" ➔ does not fit the data well
  - A biased solution

best fit line

- Model too complex ➔ small changes to the data, solution changes a lot
  - A high-variance solution

some new datapoint

(Squared) Bias of learner

- Given dataset *D* with *N* samples, learn function $h_D(x)$
- If you sample a different dataset *D'* with *N* samples, you will learn different $h_D'(x)$
- **Expected hypothesis**: $E_D[h_D(x)]$
- **Bias:** difference between what you expect to learn and truth
  - Measures how well you expect to represent true solution
  - Decreases with more complex model
  - Bias^2 at one point *x*:
  - Average Bias^2:

$D \to$ learn h_D

$D = \{(x_1, t(x_1)), \ldots, (x_N, t(x_N))\}$

$E_D[h_D(x) \mid X] = \bar{h}_N$

$(t(x) - \bar{h}_N(x))^2$

$E_x[(t(x) - \bar{h}_N(x))^2]$

Variance of learner

- Given dataset *D* with *N* samples, learn function $h_D(x)$
- If you sample a different dataset *D'* with *N* samples, you will learn different $h_D'(x)$
- **Variance:** difference between what you expect to learn and what you learn from a particular dataset
  - Measures how sensitive learner is to specific dataset
  - Decreases with simpler model
  - Variance at one point *x*:
  - Average variance:

$D \to h_D$

$E_D[(h_D(x) - \bar{h}_N(x))^2 \mid X]$

$E_X E_D[(h_D(x) - \bar{h}_N(x))^2]$.

Overfitting

Machine Learning – CSE546
Sham Kakade
University of Washington

October 6, 2016

Training set error

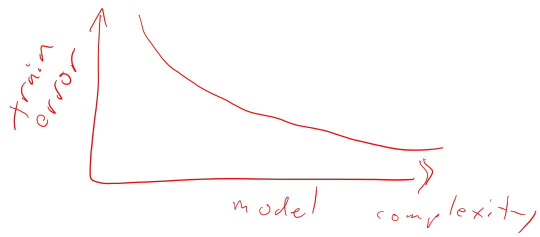
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

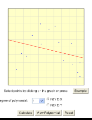
- Given a dataset (Training data)
- Choose a loss function
  - e.g., squared error (L_2) for regression
- **Training set error:** For a particular set of parameters, loss function on training data:

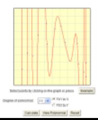
$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_{i=1}^{d} w_i h_i(\mathbf{x}_j) \right)^2$$

Training set error as a function of model complexity

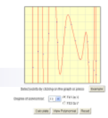
$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$







Prediction error



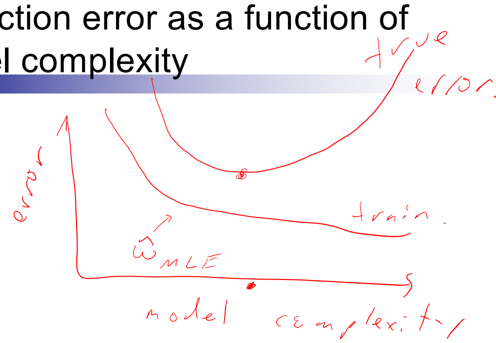
- Training set error can be poor measure of "quality" of solution
- **Prediction error:** We really care about error over all possible input points, not just training data:

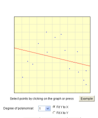
$$\begin{aligned} error_{true}(\mathbf{w}) &= E_{\mathbf{x}}\left[\left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 \right] \\ &= \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x} \end{aligned}$$

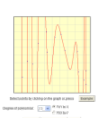
Prediction error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$







Computing prediction error

- Computing prediction
  - Hard integral
  - May not know t(**x**) for every **x**

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

- Monte Carlo integration (sampling approximation)
  - Sample a set of i.i.d. points $\{\mathbf{x}_1,\ldots,\mathbf{x}_M\}$ from p(**x**)
  - Approximate integral with sample average

$$error_{true}(\mathbf{w}) \approx \frac{1}{M} \sum_{j=1}^{M} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

Why training set error doesn't approximate prediction error?

- Sampling approximation of prediction error:

$$error_{true}(\mathbf{w}) \approx \frac{1}{M}\sum_{j=1}^{M}\left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

- Training error :

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}}\sum_{j=1}^{N_{train}}\left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

$\hat{w}_{MLE} = \arg\min_{w} \; err_{train}(w)$

- Very similar equations!!!
  - Why is training set a bad measure of prediction error???

Why training set error doesn't approximate prediction error?

Because you cheated!!!

Training error good estimate for a single **w,**
But you optimized **w** with respect to the training error, and found **w** that is good for this set of samples

Training error is a (optimistically) biased estimate of prediction error

- Very similar equations!!!
  - Why is training set a bad measure of prediction error???

Test set error

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

- Given a dataset, **randomly** split it into two parts:
  - Training data – $\{\mathbf{x}_1, \ldots, \mathbf{x}_{Ntrain}\}$
  - Test data – $\{\mathbf{x}_1, \ldots, \mathbf{x}_{Ntest}\}$
- Use training data to optimize parameters **w**
- **Test set error:** For the ***final output*** $\hat{\mathbf{w}}$, evaluate the error using:

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}}\sum_{j=1}^{N_{test}}\left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

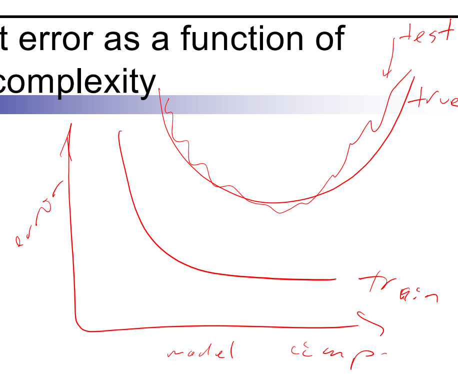
def

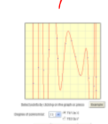
Test set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}}\sum_{j=1}^{N_{train}}\left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}}\left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}$$

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}}\sum_{j=1}^{N_{test}}\left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$





How many points to I use for training/testing?

- Very hard question to answer!
  - Too few training points, learned **w** is bad
  - Too few test points, you never know if you reached a good solution
- Bounds, such as Hoeffding's inequality can help:

$$P(|\,\hat{\theta} - \theta^* \,| \geq \epsilon) \;\leq\; 2e^{-2N\epsilon^2}$$

fixed w

- More on this later this quarter, but still hard to answer
- Typically:

$$\left| error_{true}(w) - error_{test}(w) \right| \leq \epsilon$$

  - If you have a reasonable amount of data, pick test set "large enough" for a "reasonable" estimate of error, and use the rest for learning
  - If you have little data, then you need to pull out the big guns…
    - e.g., bootstrapping

Error estimators

$$error_{true}(\mathbf{w}) \;=\; \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2 p(\mathbf{x})\,d\mathbf{x}$$

what we care about

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

estimate of (becomes bad if you minimize over w)

$$error_{test}(\mathbf{w}) \;=\; \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

evaluate model

Error estimators

Be careful!!!

Test set only unbiased if you never never ever ever do any any any any learning on the test data

For example, if you use the test set to select the degree of the polynomial… no longer unbiased!!!
(We will address this problem later)

$$error_{test}(\mathbf{w}) \;=\; \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j)\right)^2$$

What you need to know

- True error, training error, test error
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data
  - Never learn on the test data
- Overfitting

Regularization

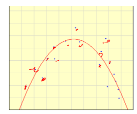
Machine Learning – CSE546
Sham Kakade
University of Washington

October 6, 2016

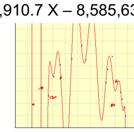
Regularization in Linear Regression

- Overfitting usually leads to very large parameter choices, e.g.:

-2.2 + 3.1 X – 0.30 X^2



-1.1 + 4,700,910.7 X – 8,585,638.4 X^2 + …



- ***Regularized*** or ***penalized*** regression aims to impose a "complexity" penalty by penalizing large weights
  - "Shrinkage" method

Ridge Regression

Regularizer

- Aleviating issues with overfitting:
- New objective:

$$\sum_{j=1} \left(t(x_j) - \sum_{i=1}^{d} w_i h_i(x_j) \right)^2$$

$$+ \lambda \sum_{i=1}^{d} w_i^2$$

Ridge Regression in Matrix Notation

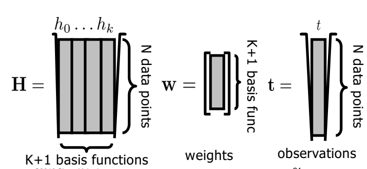
$+ \lambda \|\vec{w}\|_2^2$

dim.

$$\hat{\mathbf{w}}_{ridge} = \arg\min_w \sum_{j=1}^{N} \left(t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j)) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

$$= \arg\min_{\mathbf{w}} \underbrace{(\mathbf{H}\mathbf{w} - \mathbf{t})^T (\mathbf{H}\mathbf{w} - \mathbf{t})}_{\text{residual error}} + \lambda\, \mathbf{w}^T I_{0+k} \mathbf{w}$$

choice



$$\begin{bmatrix} w_0 \\ \vdots \\ w_k \end{bmatrix}^T \begin{bmatrix} 0 & & 0 \\ & 1 & \\ 0 & & \ddots \; 1 \end{bmatrix} \begin{bmatrix} w_0 \\ \vdots \\ w_k \end{bmatrix}$$

Minimizing the Ridge Regression Objective

$$\hat{\mathbf{w}}_{ridge} = \arg\min_w \sum_{j=1}^{N}\left(t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j))\right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

$$= \underbrace{(H\mathbf{w} - \mathbf{t})^T(H\mathbf{w} - \mathbf{t}) + \lambda\, \mathbf{w}^T I_{0+k}\mathbf{w}}_{F(w)}$$

$\nabla F(w) = 0$

$= 2H^T(Hw - t) + 2\lambda I_{0+k}\, w$

$\hat{w}_{ridge} = (H^T H + \lambda I_{0+k})^{-1} H^T t$

Shrinkage Properties

$$\hat{\mathbf{w}}_{ridge} = (H^T H + \lambda\, I_{0+k})^{-1} H^T \mathbf{t}$$

- If orthonormal features/basis: $H^T H = I$

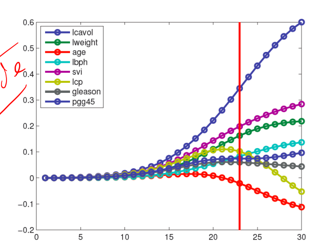
Ridge Regression: Effect of Regularization

$$\hat{\mathbf{w}}_{ridge} = \arg\min_w \sum_{j=1}^{N}\left(t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j))\right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

- Solution is indexed by the regularization parameter λ
- Larger λ
- Smaller λ
- As λ → 0
- As λ → ∞

Ridge Coefficient Path

weight magnitude



From Kevin Murphy textbook

$1/\lambda$ → λ smaller.

- Typical approach: select λ using cross validation, more on this later in the quarter

What you need to know…

- Regularization
  - Penalizes for complex models
- Ridge regression
  - L_2 penalized least-squares regression
  - Regularization parameter trades off model complexity with training error

Cross-Validation

Machine Learning – CSE546
Sham Kakade
University of Washington

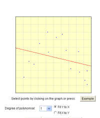
October 6, 2016

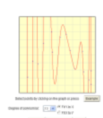
Test set error as a function of model complexity

$$error_{train}(\mathbf{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$

$$error_{true}(\mathbf{w}) = \int_{\mathbf{x}} \left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x}) \right)^2 p(\mathbf{x}) d\mathbf{x}$$

$$error_{test}(\mathbf{w}) = \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2$$





How… How… How???????

- How do we pick the regularization constant λ…
  - And all other constants in ML, 'cause one thing ML doesn't lack is constants to tune… ☹
- We could use the test data, but…

(LOO) Leave-one-out cross validation

- Consider a **validation set with 1 example**:
  - *D* – training data
  - *D*\j – training data with *j* th data point moved to validation set
- **Learn classifier $h_{D\backslash j}$ with *D*\j dataset**
- **Estimate true error** as squared error on predicting $t(\mathbf{x}_j)$:
  - Unbiased estimate of $error_{true}(h_{D\backslash j})$!
  - Seems really bad estimator, but wait!
- **LOO cross validation**: Average over all data points *j*:
  - **For each data point you leave out, learn a new classifier $h_{D\backslash j}$**
  - **Estimate error** as:

$$error_{LOO} = \frac{1}{N}\sum_{j=1}^{N}\left(t(\mathbf{x}_j) - h_{\mathcal{D}\backslash j}(\mathbf{x}_j)\right)^2$$

LOO cross validation is (almost) unbiased estimate of true error of h_D!

- When computing **LOOCV error, we only use *N-1* data points**
  - So it's not estimate of true error of learning with *N* data points!
  - Usually pessimistic, though – learning with less data typically gives worse answer
- **LOO is almost unbiased**!
- **Great news!**
  - **Use LOO error for model selection!!!**
  - **E.g., picking λ**

Computational cost of LOO

- Suppose you have 100,000 data points
- You implemented a great version of your learning algorithm
  - Learns in only 1 second
- Computing LOO will take about 1 day!!!
  - If you have to do for each choice of basis functions, it will take fooooooreeeve'!!!
- Solution 1: Preferred, but not usually possible
  - Find a cool trick to compute LOO (e.g., see homework)

Solution 2 to complexity of computing LOO: (More typical) **Use *k*-fold cross validation**

- Randomly **divide training data into *k* equal parts**
  - $D_1,\ldots,D_k$
- For each *i*
  - **Learn classifier $h_{D\backslash Di}$ using data point not in D_i**
  - **Estimate error of $h_{D\backslash Di}$ on validation set D_i:**

$$error_{\mathcal{D}_i} = \frac{k}{N}\sum_{\mathbf{x}_j \in \mathcal{D}_i}\left(t(\mathbf{x}_j) - h_{\mathcal{D}\backslash \mathcal{D}_i}(\mathbf{x}_j)\right)^2$$

- ***k*-fold cross validation error is average** over data splits:

$$error_{k-fold} = \frac{1}{k}\sum_{i=1}^{k} error_{\mathcal{D}_i}$$

- *k*-fold cross validation properties:
  - **Much faster to compute** than LOO
  - **More (pessimistically) biased** – using much less data, only *m(k-1)/k*
  - Usually, **k = 10** ☺

What you need to know…

- Never ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever train on the test data
- Use cross-validation to choose magic parameters such as λ
- Leave-one-out is the best you can do, but sometimes too slow
  - In that case, use k-fold cross-validation