Overfitting

Machine Learning – CSE546
Carlos Guestrin
University of Washington
October 2, 2013

Bias-Variance Tradeoff

- Choice of hypothesis class introduces learning bias
  - More complex class → less bias
  - More complex class → more variance

![](image)
Training set error

Given a dataset (Training data)
Choose a loss function
- e.g., squared error ($L_2^2$) for regression

Training set error: For a particular set of parameters, loss function on training data:

$$\text{error}_{\text{train}}(\mathbf{w}) = \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \left( t(x_j) - \sum_i w_i h_i(x_j) \right)^2$$

Training set error as a function of model complexity
Prediction error

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

\[
\text{error}_{\text{true}}(w) = E_x \left[ \left( t(x) - \sum_i w_i h_i(x) \right)^2 \right] = \int_x \left( t(x) - \sum_i w_i h_i(x) \right)^2 p(x) dx
\]

Prediction error as a function of model complexity
Computing prediction error

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

\[
error_{\text{true}}(w) = \int_x \left( t(x) - \sum_i w_i h_i(x) \right)^2 p(x) dx
\]

- Monte Carlo integration (sampling approximation)
  - Sample a set of i.i.d. points \( \{x_1, \ldots, x_M\} \) from \( p(x) \)
  - Approximate integral with sample average

\[
error_{\text{true}}(w) \approx \frac{1}{M} \sum_{j=1}^M \left( t(x_j) - \sum_i w_i h_i(x_j) \right)^2
\]

Why training set error doesn’t approximate prediction error?

- Sampling approximation of prediction error:

\[
error_{\text{true}}(w) \approx \frac{1}{M} \sum_{j=1}^M \left( t(x_j) - \sum_i w_i h_i(x_j) \right)^2
\]

- Training error:

\[
error_{\text{train}}(w) = \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \left( t(x_j) - \sum_i w_i h_i(x_j) \right)^2
\]

- 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

$w^* = \arg \min_w \sum_j \left( t(x_j) - \sum_i w_i h_i(x_j) \right)^2$

- Given a dataset, randomly split it into two parts:
  - Training data – $\{x_1, \ldots, x_{N_{\text{train}}}\}$
  - Test data – $\{x_1, \ldots, x_{N_{\text{test}}}\}$

- Use training data to optimize parameters $w$

- Test set error: For the final output $\hat{w}$, evaluate the error using:

$$error_{\text{test}}(w) = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} \left( t(x_j) - \sum_i w_i h_i(x_j) \right)^2$$
Test set error as a function of model complexity

Overfitting

- **Overfitting**: a learning algorithm overfits the training data if it outputs a solution \( \hat{w} \) when there exists another solution \( w' \) such that:

\[
\text{error}_{\text{train}}(\hat{w}) < \text{error}_{\text{train}}(w') \land \text{error}_{\text{true}}(w') < \text{error}_{\text{true}}(\hat{w})
\]
How many points to I use for training/testing?

- Very hard question to answer!
  - Too few training points, learned $\mathbf{w}$ is bad
  - Too few test points, you never know if you reached a good solution
- Bounds, such as Hoeffding's inequality can help:
  \[ P(|\hat{\theta} - \theta^*| \geq \epsilon) \leq 2e^{-2N\epsilon^2} \]
- More on this later this quarter, but still hard to answer
- Typically:
  - 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

\[
\text{error}_{true}(\mathbf{w}) = \int \left( t(x) - \sum_i w_i h_i(x) \right)^2 p(x) dx \quad \text{← gold standard}
\]

\[
\text{error}_{train}(\mathbf{w}) = \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \left( t(x_j) - \sum_i w_i h_i(x_j) \right)^2 \quad \text{← data for learning, optimistically biased solution}
\]

\[
\text{error}_{test}(\mathbf{w}) = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} \left( t(x_j) - \sum_i w_i h_i(x_j) \right)^2 \quad \text{← Evaluate final model}
\]
Error as a function of number of training examples for a fixed model complexity

\[
\text{ERROR}_\text{train}(w) = \frac{1}{N_{\text{total}}} \sum_{j=1}^{N_{\text{total}}} \left( y_j - \sum_{i} w_i h_i(x_j) \right)^2
\]

little data          infinite data

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)
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
Regularization in Linear Regression

- Overfitting usually leads to very large parameter choices, e.g.:
  
  \[-2.2 + 3.1 X - 0.30 X^2\]  
  \[-1.1 + 4,700,910.7 X - 8,585,638.4 X^2 + \ldots\]

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

Ridge Regression

- Ameliorating issues with overfitting:

- New objective:

  \[\sum_{j=1}^{\infty} \left( f(x_j) - w_{12}^T h_1(x_j) \right)^2 + \lambda \sum_{i=1}^{K} w_i^2 \]

  \[\text{train error} + \lambda \text{regularization cost} \]

  \[\lambda > 0\]

©2005-2013 Carlos Guestrin
Ridge Regression in Matrix Notation

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

\[ = \arg \min_w \underbrace{(Hw - t)^T(Hw - t)}_{\text{residual error}} + \lambda w^T I_{0+k} w \]

\[ H = \begin{bmatrix} h_0 \ldots h_k \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \ldots w_k \end{bmatrix}^T, \quad t = \begin{bmatrix} t \end{bmatrix} \]

Minimizing the Ridge Regression Objective

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

\[ F(w) = (Hw - t)^T(Hw - t) + \lambda w^T I_{0+k} w \]

\[ \nabla F(w) = 2H^T(Hw - t) + 2\lambda I_{0+k} w = 0 \]

\[ \hat{w}_{\text{ridge}} = \left( H^T H + \lambda I_{0+k} \right)^{-1} H^T \begin{bmatrix} t \end{bmatrix} = \left( H^T H + \lambda I_{0+k} \right)^{-1} H^T t \]

Adding \( \lambda I_{0+k} \) can only make \( H^T H \) "more invertible" (ops!!)

So regularization is also a numerical stabilization...
Shrinkage Properties

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

- If orthonormal features/basis: \( H^T H = I \) \[ \hat{w}_{MLE} : (H^T H)^{-1} H^T t \]

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

\[ = \begin{pmatrix}
1 + \lambda & 0 \\
0 & 1 + \lambda & 0 \\
1 + \lambda & 0 & \ddots
\end{pmatrix}^{-1} H^T t \]

Ridge Regression: Effect of Regularization

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

- Solution is indexed by the regularization parameter \( \lambda \)
- Larger \( \lambda \) high regularization
- Smaller \( \lambda \) low regularization
- As \( \lambda \to 0 \) \( \hat{w}_{ridge} \to \hat{w}_{MLE} \)
- As \( \lambda \to \infty \) \( \hat{w}_{ridge} \to w_0 \in \text{avrage } t(x_j) \) flat line
Ridge Coefficient Path

Typical approach: select \( \lambda \) using cross validation, more on this later in the quarter

Error as a function of regularization parameter for a fixed model complexity

\[ \text{error}_{\text{train}}(w) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} (t(x_i) - \sum_{j} w_j h_j(x_i))^2 + \frac{\lambda}{2} \sum_{j} w_j^2 \]

\[ \text{error}_{\text{test}}(w) = \int \left( t(x) - \sum_{j} w_j h_j(x) \right)^2 p(x)dx \]

Adjust \( \lambda \) helps deal with bias-variance tradeoff.
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
Test set error as a function of model complexity

\[
\text{error}_{\text{test}}(w) = \frac{1}{N_{\text{test}}} \sum_{x_{\text{test}}} \left( f(x) - \sum_{i} w_i h_i(x) \right)^2.
\]

How... How... How???????

- How do we pick the regularization constant $\lambda$...
  - And all other constants in ML, 'cause one thing ML doesn't lack is constants to tune... 😅

- We could use the test data, but...
  1. don't learn on test data
  2. may not have enough data to separate some to pick $\lambda$
(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 $h_{Dj}$ with $D_j$ dataset
- Estimate true error as squared error on predicting $t(x_j)$:
  - Unbiased estimate of $err_{true}(h_{Dj})$
  - 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 $h_{Dj}$
  - Estimate error as:
    $$error_{LOO} = \frac{1}{N} \sum_{j=1}^{N} \left( t(x_j) - h_{D \setminus j}(x_j) \right)^2$$

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!
  $$error_{true}(h_D) \sim error_{LOO}$$

- Great news!
  - Use LOO error for model selection!!!
  - E.g., picking $\lambda$
Using LOO to Pick $\lambda$

Error LOO

For each $\lambda$ exclude error $\lambda$

learn $N$ models

pick highest $\lambda$

Using LOO error for model selection
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 foootoooreeeev'!!!
- 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 $h_{D_Di}$ using data point not in $D_i$
  - Estimate error of $h_{D_Di}$ on validation set $D_i$:
    $$error_{D_i} = \frac{1}{N} \sum_{x_j \in D_i} (t(x_j) - h_{D_Di}(x_j))^2$$
- **k-fold cross validation error** is average over data splits:
  $$error_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} error_{D_i}$$

- **k-fold cross validation properties**:
  - Much faster to compute than LOO
  - More (pessimistically) biased – using much less data, only $m(k-1)/k$
  - Usually, $k = 10$ ☞ (the more the better)
What you need to know…

- Use cross-validation to choose magic parameters such as $\lambda$
- Leave-one-out is the best you can do, but sometimes too slow
  - In that case, use k-fold cross-validation