Bias-Variance Decomposition
Error Estimators
Cross-Validation
Bias-Variance tradeoff – Intuition

- Model too “simple” → does not fit the data well – a biased solution.

- Model too complex → small changes to the data, solution changes a lot – high-variance solution.
Expected Loss

- Let \( h(x) = E[t|x] \) be the optimal predictor and \( y(x) \) our actual predictor, which will incur the following expected loss

\[
E(y(x) - t)^2 = \int \int (y(x) - h(x) + h(x) - t)^2 p(x, t) dx dt
\]

\[
= E\left( (y(x) - h(x))^2 + 2(y(x) - h(x))(h(x) - t) + (h(x) - t)^2 \right)
\]

\[
= \int (y(x) - h(x))^2 p(x) dx + \int \int (h(x) - t)^2 p(x) dx dt
\]

since \( \int (E[t|x] - t)p(t \mid x)dt = 0 \)

\[
E(y(x) - t)^2 = \int (y(x) - h(x))^2 p(x) dx + \int \int (h(x) - t)^2 p(x) dt
\]

Noise term

Source of error 1
Sources of error 1 – noise

- What if we have perfect learner, infinite data?
  - Our learning solution \( y(x) \) satisfies \( y(x) = h(x) \)
  - Still have remaining, *unavoidable error* of \( \sigma^2 \) due to noise \( \varepsilon \)

\[
\int \int (y(x) - t)^2 \, p(f(x) = t \mid x) \, p(x) \, dt \, dx
\]

\[
\phi^2, \quad \varepsilon \sim N(0, \sigma^2)
\]

\[
= E[\varepsilon^2] = \sigma^2
\]
Bias-variance decomposition

- Focus on $\int (y(x) - h(x))^2 p(x) dx$
- Let us first examine expected loss averaging over data sets.
- For one data set $D$ and one test point $x$:

$$\begin{align*}
(y(x, D) - h(x))^2 &= (y(x, D) - E_D[y(x, D)] + E_D[y(x, D)] - h(x))^2 \\
&= (y(x, D) - E_D[y(x, D)])^2 + (E_D[y(x, D)] - h(x))^2 \\
&+ 2(y(x, D) - E_D[y(x, D)])(E_D[y(x, D)] - h(x)) \\
&\rightarrow E_D \text{ of this cancels to zero}
\end{align*}$$

- Hence

$$\begin{align*}
E_D(y(x, D) - h(x))^2 &= (E_D[y(x, D)] - h(x))^2 + E_D(y(x, D) - E_D[y(x, D)])^2 \\
&= \text{bias}^2 + \text{variance}
\end{align*}$$
Bias

- Suppose you are given a dataset $D$ with $m$ samples from some distribution.
- You learn function $y(x)$ from data $D$
- If you sample a different datasets, you will learn different $y(x)$
- **Expected hypothesis**: $E_D[y(x,D)]$
- **Bias**: difference between what you expect to learn and the truth.
  - Measures how well you expect to represent true solution
  - Decreases with more complex model

\[
\text{bias}^2 = \int_x \left( E_D[y(x, D)] - h(x) \right)^2 p(x) dx
\]

\(x\) Expected to learn

\(h(x)\) True model
Variance

- **Variance**: difference between what you expect to learn and what you learn from a particular dataset
  - Measures how sensitive learner is to a specific dataset
  - Decreases with simpler model

\[
\text{variance} = \int_{x} E_D \left( \left( y(x, D) - E_D[y(x, D)] \right)^2 \right) p(x) dx
\]

Learned for D    Expected to learn, averaged over datasets
Bias-Variance Tradeoff

- Choice of hypothesis class introduces learning bias
- More complex class → less bias
- More complex class → more variance
The Expected Prediction Error

- The expected prediction squared error over fixed size training sets $D$ drawn from $P(X,T)$ can be expressed as sum of three components:

$$\text{unavoidable error} + \text{bias}^2 + \text{variance}$$

where

$$\text{unavoidable error} = \sigma^2$$

$$\text{bias}^2 = \int (E_D[y(x)] - h(x))^2 p(x) dx$$

$$\text{variance} = \int E_D[(y(x) - E_D[y(x)])^2] p(x) dx$$
Higher variance

Lower bias

Highly regularized have low variance but high bias.
Training Error

- Given a training data, choose a loss function. (e.g., squared error (L2) for regression)

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

\[
Err_{\text{train}}(w) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left( t(x_i) - \sum_{j=1}^{M} w_i y(x_j) \right)^2
\]
Training Error vs. Model Complexity

Training error

Error

low Model complexity high
Prediction (Generalization) 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:

\[
\text{Err}_{\text{true}}(w) = E_X \left[ \left( t(x) - \sum_{j=1}^{M} w_i y(x_j) \right)^2 \right]
\]

\[
= \int_X \left( t - \sum_{j=1}^{M} w_i y(x_j) \right)^2 \, dx
\]
Prediction Error vs. Model Complexity

- Error vs. Model complexity
- High vs. Low
- True Error vs. Training error
- Want to minimize error
- Balance between simplicity and complexity
Why training set error doesn’t approximate prediction error?

- Sampling approximation of prediction error:

\[
Err_{true}(w) \approx \frac{1}{p} \sum_{i=1}^{p} \left( t - \sum_{j=1}^{M} w_i y(x_j) \right)^2
\]

- Training error

\[
Err_{train}(w) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \left( t(x_i) - \sum_{j=1}^{M} w_i y(x_j) \right)^2
\]

- Very similar equations!!!

Why is training set a bad measure of prediction error???
Why is training set a bad measure of prediction error???

- Training error is a good estimate for a single $w$,
- But we 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
**Test Error**

- 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 solution \( w^* \), evaluate the error using:
  \[
  \text{Err}_{\text{test}}(w^*) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left( t(x_i) - \sum_{j=1}^{M} w^*_i y(x_j) \right)^2
  \]

  Unbiased but has variance

Test set only unbiased if you never do any learning on the test data
For example, if you use the test set to select the degree of the polynomial...no longer unbiased!!!
Test Set Error vs. Model Complexity

- Test Error
- True Error
- Training Error

Error vs. Model Complexity:
- Low
- High

Want:
- Low error
- Balanced model complexity
How many points to 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
- Theory proposes error bounds (advanced course)
- 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 use some special techniques e.g., bootstrapping
Error as a function of number of training examples for a fixed model complexity
Overfitting

- With too few training examples our model may achieve zero training error but never the less has a large generalization error.

- When the training error no longer bears any relation to the generalization error the model overfits the data.
Cross-validation for detecting and preventing overfitting

Andrew W. Moore
Professor
School of Computer Science
Carnegie Mellon University
www.cs.cmu.edu/~awm
awm@cs.cmu.edu
412-268-7599

Note to other teachers and users of these slides. Andrew would be delighted if you found this source material useful in giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. PowerPoint originals are available. If you make use of a significant portion of these slides in your own lecture, please include this message, or the following link to the source repository of Andrew’s tutorials: http://www.cs.cmu.edu/~awm/tutorials. Comments and corrections gratefully received.
A Regression Problem

\[ y = f(x) + \text{noise} \]

Can we learn \( f \) from this data?

Let’s consider three methods…
Linear Regression
Quadratic Regression
Join-the-dots

Also known as piecewise linear nonparametric regression if that makes you feel better.
Which is best?

Why not choose the method with the best fit to the data?
What do we really want?

Why not choose the method with the best fit to the data?

“How well are you going to predict future data drawn from the same distribution?”
The test set method

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
The test set method

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set

(Linear regression example)
The test set method

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

(Linear regression example)
Mean Squared Error = 2.4
The test set method

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

(Quadratic regression example)
Mean Squared Error = 0.9
The test set method

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

(Join the dots example)
Mean Squared Error = 2.2
The test set method

Good news:
• Very very simple
• Can then simply choose the method with the best test-set score

Bad news:
• What’s the downside?
The test set method

Good news:
• Very very simple
• Can then simply choose the method with the best test-set score

Bad news:
• Wastes data: we get an estimate of the best method to apply to 30% less data
• If we don’t have much data, our test-set might just be lucky or unlucky

We say the “test-set estimator of performance has high variance”
LOOCV (Leave-one-out Cross Validation)

For $k=1$ to $R$

1. Let $(x_k, y_k)$ be the $k^{th}$ record
**LOOCV** (Leave-one-out Cross Validation)

For $k=1$ to $R$

1. Let $(x_k, y_k)$ be the $k^{th}$ record

2. Temporarily remove $(x_k, y_k)$ from the dataset
LOOCV (Leave-one-out Cross Validation)

For \( k = 1 \) to \( R \)

1. Let \((x_k, y_k)\) be the \( k^{th} \) record

2. Temporarily remove \((x_k, y_k)\) from the dataset

3. Train on the remaining \( R-1 \) datapoints
LOOCV (Leave-one-out Cross Validation)

For k=1 to R

1. Let \((x_k, y_k)\) be the \(k^{th}\) record
2. Temporarily remove \((x_k, y_k)\) from the dataset
3. Train on the remaining \(R-1\) datapoints
4. Note your error \((x_k, y_k)\)
LOOCV (Leave-one-out Cross Validation)

For $k=1$ to $R$

1. Let $(x_k,y_k)$ be the $k^{th}$ record
2. Temporarily remove $(x_k,y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k,y_k)$

When you’ve done all points, report the mean error.
**LOOCV** (Leave-one-out Cross Validation)

For $k=1$ to $R$

1. Let $(x_k, y_k)$ be the $k^{th}$ record
2. Temporarily remove $(x_k, y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k, y_k)$

When you've done all points, report the mean error.

$$\text{MSE}_{\text{LOOCV}} = 2.12$$
LOOCV for Quadratic Regression

For $k=1$ to $R$

1. Let $(x_k, y_k)$ be the $k^{th}$ record.

2. Temporarily remove $(x_k, y_k)$ from the dataset.

3. Train on the remaining $R-1$ datapoints.

4. Note your error $(x_k, y_k)$.

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 0.962$$
LOOCV for Join The Dots

For k = 1 to R

1. Let \((x_k, y_k)\) be the \(k^{th}\) record

2. Temporarily remove \((x_k, y_k)\) from the dataset

3. Train on the remaining \(R - 1\) datapoints

4. Note your error \((x_k, y_k)\)

When you've done all points, report the mean error.

\[
MSE_{LOOCV} = 3.33
\]
Which kind of Cross Validation?

<table>
<thead>
<tr>
<th></th>
<th>Downside</th>
<th>Upside</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test-set</strong></td>
<td>Variance: unreliable estimate of future performance</td>
<td>Cheap</td>
</tr>
<tr>
<td><strong>Leave-one-out</strong></td>
<td>Expensive. Has some weird behavior</td>
<td>Doesn’t waste data</td>
</tr>
</tbody>
</table>

..can we get the best of both worlds?
k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we’ll have k=3 partitions colored Red Green and Blue)
Randomly break the dataset into $k$ partitions (in our example we’ll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.
Randomly break the dataset into $k$ partitions (in our example we’ll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.
k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we’ll have k=3 partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.
Randomly break the dataset into $k$ partitions (in our example we’ll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

Linear Regression

$\text{MSE}_{3\text{FOLD}} = 2.05$
Randomly break the dataset into $k$ partitions (in our example we’ll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error $MSE_{3FOLD}=1.11$
Randomly break the dataset into \( k \) partitions (in our example we’ll have \( k=3 \) partitions colored Red Green and Blue)

- For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.
- For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.
- For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

\[
\text{Joint-the-dots } MSE_{3\text{FOLD}} = 2.93
\]
## Which kind of Cross Validation?

<table>
<thead>
<tr>
<th></th>
<th>Downside</th>
<th>Upside</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test-set</strong></td>
<td>Variance: unreliable estimate of future performance</td>
<td>Cheap</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Leave-one-out</strong></td>
<td>Expensive.  Has some weird behavior</td>
<td>Doesn’t waste data</td>
</tr>
<tr>
<td><strong>10-fold</strong></td>
<td>Wastes 10% of the data. 10 times more expensive than test set</td>
<td>Only wastes 10%. Only 10 times more expensive instead of R times.</td>
</tr>
<tr>
<td><strong>3-fold</strong></td>
<td>Wastier than 10-fold. Expensivier than test set</td>
<td>Slightly better than test-set</td>
</tr>
<tr>
<td><strong>R-fold</strong></td>
<td>Identical to Leave-one-out</td>
<td></td>
</tr>
</tbody>
</table>