Evaluating algorithms

• How do we determine when an algorithm achieves “good” performance?

• How should we tune the parameters of the learning algorithms (regularization parameter, choice of features, parameters of kernel, etc?)

• How do we report the performance of learning algorithms?
• One possibility: just look at the loss function

\[ J(\theta) = \sum_{i=1}^{m} \ell(\theta^T \phi(x_i), y_i) \]

• The problem: adding more features will always decrease the loss

• Example example: random outputs, random features, we can get zero loss for enough features

```matlab
m = 500;
y = randn(m,1);
Phi = randn(m,m);
theta = (Phi' * Phi) \ (Phi' * y);
norm(Phi*theta - y)^2
```

ans =

\[ 2.3722e-22 \]
• A better criterion: *training* and *testing* loss
  
  **Training set:** $x_i \in \mathbb{R}^n, y_i \in \mathbb{R}, \ i = 1, \ldots, m$

  **Testing set:** $x'_i \in \mathbb{R}^n, y'_i \in \mathbb{R}, \ i = 1, \ldots, m'$

• Find parameters by minimizing loss on the training set, but evaluate on the testing set

  \[
  \text{Training: } \theta^* = \arg\min_{\theta} \sum_{i=1}^{m} \ell(\theta^T \phi(x_i), y_i)
  \]

  \[
  \text{Evaluation: Average Loss} = \frac{1}{m'} \ell((\theta^*)^T \phi(x'_i), y'_i)
  \]

  • Performance on test set called *generalization* performance.
• Sometimes, there is a natural breakdown between training and testing data (e.g., train system on one year, test on the next)

• More common, simply device up the data: for example, use 70% for training, 30% for testing

```matlab
% Phi, y, m are all the data
m_train = ceil(0.7*m);
m_test = m - m_train;
p = randperm(m);

Phi_train = Phi(p(1:m_train),:);
y_train = y(p(1:m_train));
Phi_test = Phi(p(m_train+1:end),:);
y_test = y(p(m_train+1:end));
```
High temperature / peak demand observations
Testing loss versus degree of polynomial
Testing loss (log-scale) versus degree of polynomial
Testing loss versus number of RBF bases
Testing loss (log-scale) versus number of RBF bases
Testing loss (log-scale) versus regularization parameter (log-scale), for 70 RBF bases
A common mistake: split the data into training/testing sets, use testing set to find best performing features, regularization parameter, kernel parameters, etc (*hyperparameters*), then report the testing error for these best features.
• This is not a valid method for evaluating error: the problem is that we effectively used the testing set to “train” the system.

• What we need to do instead: break the training set itself into two sets (training and cross-validation) sets.
Cross-validation Procedure:

1. Break all data into training/testing sets (e.g., 70%/30%)

2. Break training set into training/cross-validation set (e.g., 70%/30% again)

3. Choose hyperparameters using cross-validation set

4. (Optional) Once we have selected hyperparameters, retrain using all the training set

5. Evaluate performance on the testing set
• $k$-fold cross-validation: Split training set into $k$ different “folds” (equally sized random subsets)
  – For each fold $i$, train on $k-1$ only folks, evaluate on held out fold $i$

• The extreme case, leave one out cross validation: folds are individual examples
Reporting Errors

• If we want to report performance of an algorithm, how do we do this?

• Reporting just test error doesn’t give a sense of our “confidence” in the prediction
  – If we have a testing set of size 1000, doesn’t this imply more confidence in result than a testing set of size 10?

  – What about variance in predictions? Are we getting some almost completely right and others very wrong?
• Setting: in our test set, we have a number of actual labels $y'_i$, and predictions $\hat{y}'_i$ of our algorithm

• There are really two things we may care about:
  1. What is the distribution of our errors $y'_i - \hat{y}'_i$?
  2. If we want to report some average loss

$$\text{Average loss} = \frac{1}{m'} \sum_{i=1}^{m'} \ell(\hat{y}'_i, \hat{y}_i)$$

how confident are we in this value?
Some basic probability notation

- We’ll use $Z$ to denote a random variable (with distribution $\mathcal{D}$), and use $p(z)$ to denote its probability density.

- Expected value, or mean:
  \[ \mu = E[Z] = \int zp(z)\,dz \]

- Variance
  \[ \sigma^2 = E[(Z - \mu)^2] \]

- If you haven’t seen any of this notation before, there are a number of good reviews available.
• Suppose we have $m$ samples drawn from the probability distribution $D$, written as $z_1, \ldots, z_m \sim D$

• Then we can form *empirical estimates* of the mean and variance of the distribution

$$\hat{\mu} = \frac{1}{m} \sum_{i=1}^{m} z_i$$

$$\hat{\sigma}^2 = \frac{1}{m} \sum_{i=1}^{m} (z_i - \mu)^2 \approx \frac{1}{m} \sum_{i=1}^{m} (z_i - \hat{\mu})^2$$

[You may have seen variance estimates with a $\frac{1}{m-1}$ term instead; this is needed to make the estimator *unbiased*, but we’ll typically deal with large $m$, so there isn’t much difference]
Reporting errors

- As mentioned before, we might want to know about the distribution over our prediction errors $\hat{y}_i - y_i$
• Treat $\hat{y}_i' - y_i'$ as samples from a distribution

• Might want to know about the mean (also called bias), or variance of this distribution

• If we assume prediction errors are zero-mean (but this is not always the case), then

$$\hat{\sigma}^2 = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i' - y_i')^2$$

which is the mean squared error
If we want to report some *average loss*, then we can treat \( \ell(\hat{y}_i', y_i') \) (for any loss) as the random samples (the average loss is just the mean of these samples).

![Histogram of losses](histogram.png)

**Histogram of losses** \( \ell(\hat{y}_i', y_i') \) for absolute loss
• How confident are we in our estimate of the mean (i.e., the average loss)?

• Here we’ll exploit the central limit theorem: If \( z_1, \ldots, z_m \) are (independent, identically distributed) samples from any distribution with mean \( \mu \) and variance \( \sigma^2 \), then

\[
\frac{1}{m} \sum_{i=1}^{m} z_i \rightarrow \mathcal{N}(\mu, \sigma^2/m)
\]

– I.e., the mean of any set of random variables is normally distributed

• For a normal distribution, 95\% of the data falls within 1.96 standard deviations \( \sigma \).
• This suggests a method for computing “confidence intervals” of our estimate of the average loss
  1. Form estimate of the mean:

\[ \hat{\mu} = \frac{1}{m'} \sum_{i=1}^{m'} \ell(\hat{y}^*_i, y_i) \]

2. Form estimate of the variance:

\[ \hat{\sigma}^2 = \frac{1}{m'} \sum_{i=1}^{m'} (\ell(\hat{y}^*_i, y^*_i) - \hat{\mu})^2 \]

3. With 95% confidence, the “true” mean lies within

\[ \hat{\mu} \pm 1.96\hat{\sigma}/\sqrt{m'} \]

• This procedure is technically wrong (we should be using the a different estimate of the variance, and a Student-\(t\) distribution instead of Gaussian), but it is close enough when \(m'\) is reasonably large, which is usually our setting
• Should report errors relative to some baseline (i.e., degree zero polynomial)

<table>
<thead>
<tr>
<th>Degree</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.2414 ± 0.0039</td>
</tr>
<tr>
<td>1</td>
<td>0.2407 ± 0.0027</td>
</tr>
<tr>
<td>2</td>
<td>0.1505 ± 0.0013</td>
</tr>
<tr>
<td>3</td>
<td>0.1255 ± 0.0009</td>
</tr>
<tr>
<td>4</td>
<td>0.1257 ± 0.0009</td>
</tr>
<tr>
<td>5</td>
<td>0.1267 ± 0.0009</td>
</tr>
</tbody>
</table>

• A better way of determining how algorithms compare: pairwise hypothesis testing