Lecture 4:

- Linear Regression (cont’d.)
- Optimization
- Generalization
- Model complexity
- Regularization
Administrative

• **Assignment 1** is out!
• It is due **October 26** (i.e. in two weeks).
• It includes
  – Pencil-and-paper derivations
  – Implementing kNN classifier
  – numpy/Python code
• The data has:
  - **Train**: 60K images of Rome and New York City.
  - **Test**: To be supplied later.

• You may want to split the training set into train and validation

• **Do not** use test data for training or parameter tuning

• Features:
  - Tiny image,
  - Color histogram,
  - GIST descriptors,
  - GPS locations
  - You can use your own features, *e.g.* Places CNN features ([http://places2.csail.mit.edu](http://places2.csail.mit.edu))

• Report performance on test data

adapted from Sanja Fidler
Recall from last time... **Kernel Regression**

- **1-NN for Regression**

\[
h(\mathbf{x}') = \frac{\sum_{i \in \text{knn}(\mathbf{x}')} y_i K(\mathbf{x}_i, \mathbf{x}')}{\sum_{i \in \text{knn}(\mathbf{x}')} K(\mathbf{x}_i, \mathbf{x}')} \]

- **Weighted K-NN for Regression**

\[
D = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}
\]

**Distance metrics**

- **Kernel width**

\[
w_i = \exp(-d(x_i, \text{query})^2 / \sigma^2)
\]
Recall from last time... **Least-Squares Regression**

\[ y(x) = \text{function}(x, w) \]
Recall from last time... Least-Squares Regression

• Define a model

  Linear: \( y(x) = \text{function}(x, \mathbf{w}) \)
Recall from last time... **Least-Squares Regression**

- Define a model
  - **Linear:** \( y(x) = w_0 + w_1 x \)

- Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \)

\[
\ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - y(x^{(n)}) \right]^2
\]

For a particular hypothesis (\( y(x) \) defined by a choice of \( w \), drawn in red), what does the loss represent geometrically?
Recall from last time... **Least-Squares Regression**

- **Define a model**
  
  **Linear:**  
  \[ y(x) = w_0 + w_1 x \]

- **Standard loss/cost/objective function** measures the squared error between \( y \) and the true value \( t \)

  Linear model:  
  \[ \ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2 \]

- **The loss for the red hypothesis is the sum of the squared vertical errors** (squared lengths of green vertical lines)
Recall from last time... **Least-Squares Regression**

- Define a model
  
  Linear: \[ y(x) = w_0 + w_1 x \]

- Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \)

  Linear model: 
  
  \[
  \ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2
  \]

- How do we obtain weights \( w = (w_0, w_1) \)?
Recall from last time… **Least-Squares Regression**

- **Define a model**
  
  **Linear:** \( y(x) = w_0 + w_1 x \)

- **Standard loss/cost/objective function** measures the squared error between \( y \) and the true value \( t \)

  **Linear model:**
  
  \[
  \ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2
  \]

- **How do we obtain weights** \( w = (w_0, w_1) \)? Find \( w \) that minimizes loss \( \ell(w) \)
Optimizing the Objective

• One straightforward method: gradient descent
  - initialize $w$ (e.g., randomly)
  - repeatedly update $w$ based on the gradient
    $$w \leftarrow w - \lambda \frac{\partial \ell}{\partial w}$$

• $\lambda$ is the learning rate

• For a single training case, this gives the LMS update rule:
  $$w \leftarrow w + 2\lambda \left( t^{(n)} - y(x^{(n)}) \right) x^{(n)}$$

  \[\text{error}\]

• Note: As error approaches zero, so does the update ($w$ stops changing)
Optimizing the Objective
Optimizing the Objective
Effect of learning rate $\lambda$

- Large $\lambda \Rightarrow$ Fast convergence but larger residual error
  Also possible oscillations
- Small $\lambda \Rightarrow$ Slow convergence but small residual error
Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:
  1. **Batch updates**: sum or average updates across every example \( n \), then change the parameter values

\[
\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \left( t^{(n)} - y(x^{(n)}) \right) x^{(n)}
\]

2. **Stochastic/online updates**: update the parameters for each training case in turn, according to its own gradients

---

**Algorithm 1** Stochastic gradient descent

1: Randomly shuffle examples in the training set
2: for \( i = 1 \) to \( N \) do
3:   Update:
   \[
   \mathbf{w} \leftarrow \mathbf{w} + 2\lambda (t^{(i)} - y(x^{(i)}) \rangle \langle x^{(i)} \rangle \]
   (update for a linear model)
4: end for
Optimizing Across Training Set

• Two ways to generalize this for all examples in training set:
  
  1. **Batch updates:** sum or average updates across every example n, then change the parameter values
     
     \[
     \mathbf{w} \leftarrow \mathbf{w} + 2\lambda \left( t^{(n)} - y(x^{(n)}) \right) x^{(n)}
     \]

  2. **Stochastic/online updates:** update the parameters for each training case in turn, according to its own gradients

• Underlying assumption: sample is independent and identically distributed (i.i.d.)
Analytical Solution

• For some objectives we can also find the optimal solution analytically
• This is the case for linear least-squares regression
• How?
Vectorization

• Consider our model:

\[ y(x) = w_0 + w_1 x \]

• Let

\[
\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{x}^T = [1 \ x]
\]

• Can write the model in vectorized form as

\[ y(x) = \mathbf{w}^T \mathbf{x} \]
Vectorization

• Consider our model with \( n \) instances:

\[
\mathbf{t} = \begin{bmatrix} t^{(1)}, t^{(2)}, \ldots, t^{(N)} \end{bmatrix}^T \rightarrow \mathbb{R}^{N \times 1}
\]

\[
\mathbf{X} = \begin{bmatrix} 1, x^{(1)} \\
1, x^{(2)} \\
\vdots \\
1, x^{(N)} \end{bmatrix}^T \rightarrow \mathbb{R}^{N \times 2}
\]

\[
\mathbf{w} = \begin{bmatrix} w_0 \\
w_1 \end{bmatrix} \rightarrow \mathbb{R}^{2 \times 1}
\]

• Then:

\[
\ell(\mathbf{w}) = \sum_{n=1}^{N} \left[ \mathbf{w}^T \mathbf{x}^{(n)} - t^{(n)} \right]^2
\]

\[
= (\mathbf{Xw} - \mathbf{t})^T (\mathbf{Xw} - \mathbf{t})
\]
Analytical Solution

- Instead of using GD, solve for optimal \( w \) analytically
  - Notice the solution is when \( \frac{\partial}{\partial w} \ell(w) = 0 \)

- Derivation:
  \[
  \ell(w) = (Xw - t)^T(Xw - t) = w^T X^T Xw - t^T Xw - w^T X^T t + t^T t
  = w^T X^T Xw - 2w^T X^T t + t^T t
  \]
  - Take derivative and set equal to 0, then solve for
  \[
  \frac{\partial}{\partial w} \left( w^T X^T Xw - 2w^T X^T t + t^T t \right) = 0
  \]
  \[
  (X^T X) w - X^T t = 0
  \]
  \[
  (X^T X) w = X^T t
  \]

Closed Form Solution: \( w = (X^T X)^{-1} X^T t \)

- If \( X^T X \) is not invertible (i.e., singular), may need to:
  - Use pseudo-inverse instead of the inverse
    - In Python, \( \text{numpy.linalg.pinv}(a) \)
  - Remove redundant (not linearly independent) features
  - Remove extra features to ensure that \( d \leq N \)
Multi-dimensional Inputs

- One method of extending the model is to consider other input dimensions

\[ y(x) = w_0 + w_1 x_1 + w_2 x_2 \]

- In the Boston housing example, we can look at the number of rooms

![Scatter plot of median house price vs average number of rooms](scatter_plot.png)
Linear Regression with Multi-dimensional Inputs

• Imagine now we want to predict the median house price from these multi-dimensional observations

• Each house is a data point \( n \), with observations indexed by \( j \):

\[
\mathbf{x}^{(n)} = \left( x_1^{(n)}, \ldots, x_j^{(n)}, \ldots, x_d^{(n)} \right)
\]

• We can incorporate the bias \( w_0 \) into \( w \), by using \( x_0 = 1 \), then

\[
y(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}
\]

• We can then solve for \( \mathbf{w} = (w_0, w_1, \ldots, w_d) \). How?

• We can use gradient descent to solve for each coefficient, or compute \( \mathbf{w} \) analytically (how does the solution change?)

recall: \( \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t} \)
More Powerful Models?

• What if our linear model is not good? How can we create a more complicated model?
Fitting a Polynomial

• What if our linear model is not good? How can we create a more complicated model?

• We can create a more complicated model by defining input variables that are combinations of components of \( x \).

• Example: an \( M \)-th order polynomial function of one dimensional feature \( x \):

\[
y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j
\]

where \( x^j \) is the \( j \)-th power of \( x \).

• We can use the same approach to optimize for the weights \( w \).

• How do we do that?
Some types of basis functions in 1-D

\begin{align*}
\phi_j(x) &= \sigma \left( \frac{x - \mu_j}{s} \right) \\
\sigma(a) &= \frac{1}{1 + \exp(-a)} \\
\phi_j(x) &= \exp \left\{ -\frac{(x - \mu_j)^2}{2s^2} \right\}
\end{align*}

Sigmoid functions

Gaussian functions

Polynomial functions
Two types of linear model that are equivalent with respect to learning

- The first model has the same number of adaptive coefficients as the dimensionality of the data + 1.

- The second model has the same number of adaptive coefficients as the number of basis functions + 1.

- Once we have replaced the data by the outputs of the basis functions, fitting the second model is exactly the same problem as fitting the first model (unless we use the kernel trick).

\[
y(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \ldots = w^T x
\]

\[
y(x, w) = w_0 + w_1 \phi_1(x) + w_2 \phi_2(x) + \ldots = w^T \Phi(x)
\]
General linear regression problem

• Using our new notations for the basis function linear regression can be written as

\[ y = \sum_{j=0}^{n} w_j \phi_j(x) \]

where \( \phi_j(x) \) can be either \( x_j \) for multivariate regression or one of the nonlinear basis we defined.

• Once again we can use “least squares” to find the optimal solution.
LMS for the general linear regression problem

Our goal is to minimize the following loss function:

$$J(w) = \sum_{i} (y^i - \sum_{j} w_j \phi_j(x^i))^2$$

Moving to vector notations we get:

$$J(w) = \sum_{i} (y^i - w^T \phi(x^i))^2$$

We take the derivative w.r.t $w$

$$\frac{\partial}{\partial w} \sum_{i} (y^i - w^T \phi(x^i))^2 = 2 \sum_{i} (y^i - w^T \phi(x^i))\phi(x^i)^T$$

Equating to 0 we get

$$2 \sum_{i} (y^i - w^T \phi(x^i))\phi(x^i)^T = 0 \Rightarrow$$

$$\sum_{i} y^i \phi(x^i)^T = w^T \left[ \sum_{i} \phi(x^i)\phi(x^i)^T \right]$$
LMS for the general linear regression problem

We take the derivative w.r.t \( w \)

\[
\frac{\partial}{\partial w} \sum_i (y^i - w^T \phi(x^i))^2 = 2 \sum_i (y^i - w^T \phi(x^i))\phi(x^i)^T
\]

Equating to 0 we get

\[
2 \sum_i (y^i - w^T \phi(x^i))\phi(x^i)^T = 0 \Rightarrow \\
\sum_i y^i \phi(x^i)^T = w^T \left[ \sum_i \phi(x^i)\phi(x^i)^T \right]
\]

Define:

\[
\Phi = \begin{pmatrix}
\phi_0(x^1) & \phi_1(x^1) & \cdots & \phi_m(x^1) \\
\phi_0(x^2) & \phi_1(x^2) & \cdots & \phi_m(x^2) \\
\vdots & \vdots & & \vdots \\
\phi_0(x^n) & \phi_1(x^n) & \cdots & \phi_m(x^n)
\end{pmatrix}
\]

Then deriving \( w \) we get:

\[
w = (\Phi^T\Phi)^{-1}\Phi^T y
\]
LMS for the general linear regression problem

$$J(w) = \sum_i (y_i - w^T \phi(x_i))^2$$

Deriving $w$ we get:

$$w = (\Phi^T \Phi)^{-1} \Phi^T y$$

$k+1$ entries vector

$n$ entries vector

$n$ by $k+1$ matrix

This solution is also known as ‘pseudo inverse’
$0^{th}$ order polynomial

$M = 0$
1st order polynomial

\[ M = 1 \]
3\textsuperscript{rd} order polynomial

\[ M = 3 \]
9\textsuperscript{th} order polynomial
Which Fit is Best?

from Bishop

$M = 0$

$M = 1$

$M = 3$

$M = 9$
Root Mean Square (RMS) Error

\[ E(w) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2 \]

\[ E_{\text{RMS}} = \sqrt{2E(w^*)/N} \]

The division by \( N \) allows us to compare different sizes of data sets on an equal footing, and the square root ensures that \( E_{\text{RMS}} \) is measured on the same scale (and in the same units) as the target variable \( t \).
Root Mean Square (RMS) Error

\[ E_{\text{RMS}} = \sqrt{\frac{2E(w^*)}{N}} \]

\[ E(w) = \frac{1}{2} \sum_{n=1}^{N} (t_n - \phi(x_n)^T w)^2 = \frac{1}{2} \| t - \Phi w \|^2 \]
Generalization

• **Generalization** = model’s ability to predict the held out data
• What is happening?
• Our model with $M = 9$ **overfits** the data (it models also noise)

![Graph of the root-mean-square error](chart.png)

For $M = 9$, the training set error goes to zero, as we might expect because this polynomial contains $10$ degrees of freedom corresponding to the $10$ coefficients $w^\star_0, \ldots, w^\star_9$, and so can be tuned exactly to the $10$ data points in the training set. However, the test set error has become very large and, as we saw in Figure 1.4, the corresponding function $y(x, w^\star)$ exhibits wild oscillations. This may seem paradoxical because a polynomial of given order contains all lower order polynomials as special cases. The $M = 9$ polynomial is therefore capable of generating results at least as good as the $M = 3$ polynomial. Furthermore, we might suppose that the best predictor of new data would be the function $\sin(2\pi x)$ from which the data was generated (and we shall see later that this is indeed the case). We know that a power series expansion of the function $\sin(2\pi x)$ contains terms of all orders, so we might expect that results should improve monotonically as we increase $M$.

We can gain some insight into the problem by examining the values of the coefficients $w^\star$ obtained from polynomials of various order, as shown in Table 1.1.

<table>
<thead>
<tr>
<th>$M$</th>
<th>$w^\star_0$</th>
<th>$w^\star_1$</th>
<th>$w^\star_2$</th>
<th>$w^\star_3$</th>
<th>$w^\star_4$</th>
<th>$w^\star_5$</th>
<th>$w^\star_6$</th>
<th>$w^\star_7$</th>
<th>$w^\star_8$</th>
<th>$w^\star_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.19</td>
<td>0.82</td>
<td>0.31</td>
<td>0.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-1.27</td>
<td>7.99</td>
<td>232.37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-25.43</td>
<td>-5321.83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>17.37</td>
<td>48568.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-231639.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>640042.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1061800.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1042400.18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-557682.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>125201.43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generalization

- **Generalization** = model’s ability to predict the held out data
- What is happening?
- Our model with $M = 9$ **overfits** the data (it models also noise)
- Not a problem if we have lots of training examples
Generalization

- Generalization = model’s ability to predict the held out data
- What is happening?
- Our model with $M = 9$ overfits the data (it models also noise)
- Let’s look at the estimated weights for various $M$ in the case of fewer examples

<table>
<thead>
<tr>
<th></th>
<th>$M = 0$</th>
<th>$M = 1$</th>
<th>$M = 6$</th>
<th>$M = 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_0^*$</td>
<td>0.19</td>
<td>0.82</td>
<td>0.31</td>
<td>0.35</td>
</tr>
<tr>
<td>$w_1^*$</td>
<td>-1.27</td>
<td></td>
<td>7.99</td>
<td>232.37</td>
</tr>
<tr>
<td>$w_2^*$</td>
<td></td>
<td>-25.43</td>
<td></td>
<td>-5321.83</td>
</tr>
<tr>
<td>$w_3^*$</td>
<td></td>
<td></td>
<td>17.37</td>
<td></td>
</tr>
<tr>
<td>$w_4^*$</td>
<td></td>
<td></td>
<td></td>
<td>-231639.30</td>
</tr>
<tr>
<td>$w_5^*$</td>
<td></td>
<td></td>
<td></td>
<td>640042.26</td>
</tr>
<tr>
<td>$w_6^*$</td>
<td></td>
<td></td>
<td></td>
<td>-1061800.52</td>
</tr>
<tr>
<td>$w_7^*$</td>
<td></td>
<td></td>
<td></td>
<td>1042400.18</td>
</tr>
<tr>
<td>$w_8^*$</td>
<td></td>
<td></td>
<td></td>
<td>-557682.99</td>
</tr>
<tr>
<td>$w_9^*$</td>
<td></td>
<td></td>
<td></td>
<td>125201.43</td>
</tr>
</tbody>
</table>
Generalization

• Generalization = model’s ability to predict the held out data
• What is happening?
• Our model with M = 9 overfits the data (it models also noise)
• Let’s look at the estimated weights for various M in the case of fewer examples
• The weights are becoming huge to compensate for the noise
• One way of dealing with this is to encourage the weights to be small (this way no input dimension will have too much influence on prediction). This is called regularization.
1-D regression illustrates key concepts

• Data fits – is linear model best (model selection)?
  – Simplest models do not capture all the important variations (signal) in the data: underfit
  – More complex model may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model

• One method of assessing fit:
  – test generalization = model’s ability to predict the held out data

• Optimization is essential: stochastic and batch iterative approaches; analytic when available
Regularized Least Squares

- A technique to control the overfitting phenomenon
- Add a penalty term to the error function in order to discourage the coefficients from reaching large values

**Ridge regression**

\[
\widetilde{E}(w) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2 + \frac{\lambda}{2} \|w\|^2
\]

which is minimized by

\[
w = \left(\lambda I + \Phi^T \Phi\right)^{-1} \Phi^T t.
\]
The effect of regularization

1. INTRODUCTION

One technique that is often used to control the over-fitting phenomenon in such cases is that of regularization, which involves adding a penalty term to the error function (1.2) in order to discourage the coefficients from reaching large values. The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified error function of the form

\[
\tilde{E}(w) = \frac{1}{2} N \sum_{n=1}^{N} \left\{ y(x_n, w) - t_n \right\}^2 + \lambda \|w\|^2 (1.4)
\]

where \( \|w\|^2 \equiv w^T w = w_0^2 + w_1^2 + \ldots + w_M^2 \), and the coefficient \( \lambda \) governs the relative importance of the regularization term compared with the sum-of-squares error term. Note that often the coefficient \( w_0 \) is omitted from the regularizer because its inclusion causes the results to depend on the choice of origin for the target variable (Hastie et al., 2001), or it may be included but with its own regularization coefficient (we shall discuss this topic in more detail in Section 5.5.1). Again, the error function in (1.4) can be minimized exactly in closed form. Techniques such as this are known in the statistics literature as shrinkage methods because they reduce the value of the coefficients. The particular case of a quadratic regularizer is called ridge regression (Hoerl and Kennard, 1970). In the context of neural networks, this approach is known as weight decay.

Figure 1.7 shows the results of fitting the polynomial of order \( M = 9 \) to the same data set as before but now using the regularized error function given by (1.4). We see that, for a value of \( \ln \lambda = -18 \), the over-fitting has been suppressed and we now obtain a much closer representation of the underlying function \( \sin(2\pi x) \). If, however, we use too large a value for \( \lambda \) then we again obtain a poor fit, as shown in Figure 1.7 for \( \ln \lambda = 0 \). The corresponding coefficients from the fitted polynomials are given in Table 1.2, showing that regularization has the desired effect of reducing.
The effect of regularization

The corresponding coefficients from the fitted polynomials, showing that regularization has the desired effect of reducing the magnitude of the coefficients.
A more general regularizer

\[
\frac{1}{2} \sum_{n=1}^{N} \left\{ t_n - w^T \phi(x_n) \right\}^2 + \frac{\lambda}{2} \sum_{j=1}^{M} |w_j|^q
\]
1-D regression illustrates key concepts

- Data fits – is linear model best (model selection)?
  - Simplest models do not capture all the important variations (signal) in the data: underfit
  - More complex model may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model
- One method of assessing fit:
  - test generalization = model’s ability to predict the held out data
- Optimization is essential: stochastic and batch iterative approaches; analytic when available
Next Lecture:
Machine Learning Methodology