This week

• Linear regression problem
  - continuous outputs
  - simple model

• Introduce key concepts:
  - loss functions
  - generalization
  - optimization
  - model complexity – regularization

Classification

• Input: X
  - Real valued, vectors over real.
  - Discrete values (0,1,2,...)
  - Other structures (e.g., strings, graphs, etc.)

• Output: Y
  - Discrete (0,1,2,...)

Regression

• Input: X
  - Real valued, vectors over real.
  - Discrete values (0,1,2,...)
  - Other structures (e.g., strings, graphs, etc.)

• Output: Y
  - Real valued, vectors over real.
Choosing a restaurant

• In everyday life we need to make decisions by taking into account lots of factors
• The question is what weight we put on each of these factors (how important are they with respect to the others).
• Assume we would like to build a recommender system based on an individual’s preferences
• If we have many observations we may be able to recover the weights

<table>
<thead>
<tr>
<th>Reviews (out of 5 stars)</th>
<th>$</th>
<th>Distance</th>
<th>Cuisine (out of 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20</td>
<td>21</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>27</td>
<td>53</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

Some other examples

• Weight + height ➔ cholesterol level
• Age + gender time ➔ time spent in front of the TV
• Past choices of a user ➔ 'Netflix score'
• Profile of a job (user, machine, time) ➔ Memory usage of a submitted process.

Example: Polynomial Curve Fitting

The green curve is the true function (which is not a polynomial) – not known
The data points are uniform in x but have noise in t.

\[ t(x) = f(x) + \varepsilon \]

Aim: fit a curve to these points
Key questions:
– How do we parametrize the model (the curve)?
– What loss (objective) function should we use to judge fit?
– How do we optimize fit to unseen test data (generalization)?

1-D regression

\[ b \\ a \\ x \]

\[ y = ax + b \]
One-dimensional regression

Find a line that represents the "best" linear relationship:

\[ b = ax \]

• Problem: the data does not go through a line

\[ e_i = b_i - a_i x \]

• Find the line that minimizes the sum:

\[ \sum (b_i - a_i x)^2 \]

• Problem: the data does not go through a line

\[ e(x) = \sum (b_i - a_i x)^2 \]

• We are looking for \( \hat{x} \) that minimizes

\[ e(x) = \sum (b_i - a_i x)^2 \]
Matrix notation

• Using the following notations

\[
a = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}
\]

• We can rewrite the error function using linear algebra as:

\[
e(x) = \sum_i (b_i - a_i x)^2 = (b - xa)^T (b - xa) = \|b - xa\|^2
\]

Example: Boston House Prizes

• Estimate median house price in a neighborhood based on neighborhood statistics

• Look at first (of 13) attributes: per capita crime rate

• Use this to predict house prices in other neighborhoods

Represent the data

• Data described as pairs \(D = \{(x_1^{(1)}, t_1^{(1)}), (x_2^{(2)}, t_2^{(2)}), \ldots, (x_n^{(N)}, t_n^{(N)})\}\)

  - \(x\) is the input feature (per capita crime rate)
  - \(t\) is the target output (median house price)

• Here \(t\) is continuous, so this is a regression problem

• Could take first 300 examples as training set, remaining 206 as test set

  - Use the training examples to construct hypothesis, or function approximator, that maps \(x\) to predicted \(y\)
  - Evaluate hypothesis on test set

https://archive.ics.uci.edu/ml/datasets/Housing
Noise

A simple model typically does not exactly fit the data – lack of fit can be considered noise

Sources of noise
– Imprecision in data attributes (input noise)
– Errors in data targets (mislabling)
– Additional attributes not taken into account by data attributes, affect target values (latent variables)
– Model may be too simple to account for data targets

Least-Squares Regression

• Standard loss/cost/objective function measures the squared error in the prediction of \( t(x) \) from \( x \).
  \[
  J(w) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w^T x^{(n)})]^2
  \]

• The loss for the red hypothesis is the sum of the squared vertical errors.

Optimizing the Objective

• One straightforward method: initialize \( w \) randomly, repeatedly update based on gradient descent in \( J \)
  \[
  w \leftarrow w - \lambda \frac{\partial J}{\partial w}
  \]

• Here \( \lambda \) is the learning rate

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

• Note: as error approaches zero, so does update

Effect of step-size \( \lambda \)

• Large \( \lambda \) => Fast convergence but larger residual error
  Also possible oscillations

• Small \( \lambda \) => Slow convergence but small residual error
Optimizing Across Training Set

• Two ways to generalize this for all examples in training set:

  1. Stochastic/online updates – update the parameters for each training case in turn, according to its own gradients
  2. Batch updates: sum or average updates across every example \( i \), then change the parameter values

\[ w \leftarrow w + 2\lambda \sum_{n=1}^{N} (t^{(n)} - y(x^{(n)}))x^{(n)} \]

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

Non-iterative Least-squares Regression

• An alternative optimization approach is non-iterative: take derivatives, set to zero, and solve for parameters.

\[
\frac{dJ(w)}{dw_0} = -2 \sum_{n=1}^{N} (t^{(n)} - (w_0 + w x^{(n)})) = 0
\]

\[
w_0 = \left( \sum_{n=1}^{N} t^{(n)} - w x^{(n)} \right) / N = \bar{t} - w \bar{x}
\]

\[
w = \frac{\sum_{n} (t^{(n)} - \bar{t})(x^{(n)} - \bar{x})}{\sum_{n} (x^{(n)} - \bar{x})^2}
\]

Multi-dimensional linear regression

• Using a model with \( m \) parameters

\[
b = a_1 x_1 + ... + a_m x_m = \sum_j a_j x_j
\]
Multi-dimensional linear regression

- Using a model with $m$ parameters
  \[ b = a_1 x_1 + \ldots + a_m x_m = \sum_j a_j x_j \]

and $n$ measurements

\[
e(x) = \sum_{i=1}^{n} (b_i - \sum_{j=1}^{m} a_{i,j} x_j)^2
= \| b - [\sum_{j=1}^{m} a_{i,j} x_j] \|^2
= \| b - Ax \|^2
\]
\[
b - Ax = \begin{bmatrix}
b_1 \\
\vdots \\
b_n \\
a_{1,1} & \cdots & a_{1,m} \\
a_{n,1} & \cdots & a_{n,m}
\end{bmatrix}\begin{bmatrix}x_1 \\ \vdots \\ x_n\end{bmatrix}
\]

\[
b - Ax = \begin{bmatrix}b_1 - (a_{1,1}x_1 + \ldots + a_{1,m}x_m) \\
\vdots \\
b_n - (a_{n,1}x_1 + \ldots + a_{n,m}x_m)\end{bmatrix}
\]

Example: Boston House Prizes — revisited

- One method of extending the model is to consider other input dimensions
  \[y(x) = w_0 + w_1x_1 + w_2x_2\]

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

- We can use gradient descent to solve for each coefficient, or use linear algebra — solve system of equations
• Imagine now want to predict the median house price from these multi-dimensional observations

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

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

• Simple predictor is analogue of linear classifier, producing real-valued \( y \) for input \( x \) with parameters \( \mathbf{w} \) (effectively fixing \( x_0 = 1 \)):

\[
y = \mathbf{w}_0 + \sum_{j=1}^d w_j x_j = \mathbf{w}^T \mathbf{x}
\]

Minimizing \( e(\mathbf{x}) \)

\( \mathbf{x}_{\text{min}} \) minimizes \( e(\mathbf{x}) \) if

\[
\mathbf{x}_{\text{min}} = \mathbf{A}^+ \mathbf{b}
\]

Multi-dimensional linear regression

\[
e(\mathbf{x}) = \|\mathbf{b} - \mathbf{A} \mathbf{x}\|_2^2 = (\mathbf{b} - \mathbf{A} \mathbf{x})^T (\mathbf{b} - \mathbf{A} \mathbf{x}) = \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} - \mathbf{x}^T \mathbf{A}^T \mathbf{b} - \mathbf{b}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{b}.
\]

• A minimum occurs when
  1. The first derivative is zero,
  2. The second derivative is positive.

• Multidimensional case:
  - 1\(^{\text{st}}\) derivative of a function \( f(\mathbf{x}) \) is the gradient, \( \nabla f(\mathbf{x}) \) (a row vector)
  - 2\(^{\text{nd}}\) derivative, the Hessian, is a matrix that we will denote as \( \mathbf{H}_f(\mathbf{x}) \).

\[
\nabla e(\mathbf{x}) = 2 \mathbf{A}^T \mathbf{A} \mathbf{x} - 2 \mathbf{A}^T \mathbf{b}.
\]

\[
\mathbf{H}_e(\mathbf{x}) = 2 \mathbf{A}^T \mathbf{A}.
\]
Minimizing $e(x)$

$x_{\text{min}}$ minimizes $e(x)$ if $e(x)$ is flat at $x_{\text{min}}$

$e(x)$ is flat at $x_{\text{min}}$

$
\nabla e(x_{\text{min}}) = 0
$

$e(x)$ does not go down around $x_{\text{min}}$

$H_x(x_{\text{min}})$ is positive semi-definite
Recap: Positive semi-definite

\( A \) is positive semi-definite

\[ \iff \]

\[ x^T A x \geq 0, \text{ for all } x \]

Minimizing \( e(x) = \|b - Ax\|^2 \)

\[ A^T A \hat{x} = A^T b \]

\( \hat{x} \) minimizes \( e(x) \) if

2\( A^T A \) is positive semi-definite

Always true

Minimizing \( e(x) = \|b - Ax\|^2 \)

\[ A^T A \hat{x} = A^T b \]

The normal equation

\( \hat{x} \) minimizes \( e(x) \) if

2\( A^T A \) is positive semi-definite

Always true
Geometric interpretation

- $b$ is a vector in $\mathbb{R}^n$
- The columns of $A$ define a vector space $\text{range}(A)$

$Ax$ is an arbitrary vector in $\text{range}(A)$
Geometric interpretation

- \( b \) is a vector in \( \mathbb{R}^n \)
- The columns of \( A \) define a vector space \( \text{range}(A) \)
- \( Ax \) is an arbitrary vector in \( \text{range}(A) \)

\[
x_1a_1 + x_2a_2 = Ax
\]

The normal equation: \( A^T\hat{x} = A^Tb \)

Geometric interpretation

- \( Ax \) is the orthogonal projection of \( b \) onto \( \text{range}(A) \)
  \( \iff A^T(b - Ax) = 0 \iff A^TA\hat{x} = A^Tb \)

\[
x_1\hat{a}_1 + x_2\hat{a}_2 = A\hat{x}
\]

The normal equation: \( A^T\hat{x} = A^Tb \)

- Existence: \( A^T\hat{x} = A^Tb \) has always a solution
The normal equation: \( A^T A \hat{x} = A^T b \)

- **Existence:** \( A^T A \hat{x} = A^T b \) has always a solution
- **Uniqueness:** the solution is unique if the columns of \( A \) are linearly independent

---

**Linear models**

- It is mathematically easy to fit linear models to data.
  - We can learn a lot about model-fitting in this relatively simple case.
- There are many ways to make linear models more powerful while retaining their nice mathematical properties:
  - By using non-linear, non-adaptive basis functions, we can get generalized linear models that learn non-linear mappings from input to output but are linear in their parameters — only the linear part of the model learns.
  - By using kernel methods we can handle expansions of the raw data that use a huge number of non-linear, non-adaptive basis functions.
  - By using large margin kernel methods we can avoid overfitting even when we use huge numbers of basis functions.
- But linear methods will not solve most AI problems.
  - They have fundamental limitations.

---

**Some types of basis functions in 1-D**

- **Sigmoid functions**
  \[ \phi_s(x) = \sigma \left( \frac{x - \mu_j}{s} \right) \]

  \[ \sigma(a) = \frac{1}{1 + \exp(-a)} \]

- **Gaussian functions**
  \[ \phi_g(x) = \exp \left( -\frac{(x - \mu_j)^2}{2\sigma^2} \right) \]

- **Polynomials**
Two types of linear model that are equivalent with respect to learning

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

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

- 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).
- So we'll just focus on the first model.

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 notation we get:

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

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

\[ \frac{\partial}{\partial w} \sum_i (y_i - \sum_j w_j \phi_j(x_i))^2 = 2 \sum_i (y_i - \sum_j w_j \phi_j(x_i)) \phi(x_i)^T \]

Equating to 0 we get:

\[ 2 \sum_i (y_i - w^T \phi(x_i)) \phi(x_i)^T = 0 \]

\[ \sum_i y_i \phi(x_i)^T = w^T \left[ \sum_i \phi(x_i) \phi(x_i)^T \right] \]

LMS for the general linear regression problem

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**

\[ J(w) = \sum (y - \phi(x))^2 \]

Deriving \( w \) we get:

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

This solution is also known as ‘pseudoinverse’

**Fitting a polynomial**

- Now we use one of these basis functions: an \( M \)th order polynomial function

\[ y(x, w) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j \]

- We can use the same approaches to optimize the values of the weights on each coefficient: analytic, and iterative

**0th order polynomial**

\[ M = 0 \]

**1st order polynomial**

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

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{3rd_order_polynomial.png}
\caption{3\textsuperscript{rd} order polynomial}
\end{figure}

9\textsuperscript{th} order polynomial

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{9th_order_polynomial.png}
\caption{9\textsuperscript{th} order polynomial}
\end{figure}

Root Mean Square (RMS) Error

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

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 \( t \).

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{RMS_error.png}
\caption{Root Mean Square (RMS) Error}
\end{figure}

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

The overfitting problem
## 1-D regression illustrates key concepts

- **Data fits** – is linear model best (model selection)?
  - Simplest models do not capture all the important
  - 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

### Root Mean Square (RMS) Error

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>w_1^*</td>
<td>-1.27</td>
<td>7.99</td>
<td>232.37</td>
</tr>
<tr>
<td>w_2^*</td>
<td>-25.43</td>
<td>-5321.83</td>
<td></td>
</tr>
<tr>
<td>w_3^*</td>
<td>17.37</td>
<td>48568.31</td>
<td></td>
</tr>
<tr>
<td>w_4^*</td>
<td>-231639.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w_5^*</td>
<td>640042.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w_6^*</td>
<td>-1061800.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w_7^*</td>
<td>1042400.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w_8^*</td>
<td>-557682.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w_9^*</td>
<td>125201.43</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**The overfitting problem**

### Increasing the size of training data

- For a given model complexity, the over-fitting problem become less severe as the size of the data set increases.
- Another way to say this is that the larger the data set, the more complex (in other words more flexible) the model that we can afford to fit to the data.

### 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

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

\[
\|w\|^2 = w^T w = w_0^2 + w_1^2 + \ldots + w_M^2
\]

which is minimized by

\[
w = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T t.
\]
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

$$\frac{1}{2} \sum_{n=1}^{N} \{t_n - w^T \phi(x_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^{M} |w_j|^q$$