Machine Learning
ITCS 4156

Linear Regression

Razvan C. Bunescu
Department of Computer Science @ CCI
rbunescu@uncc.edu
Supervised Learning

- **Task** = learn an (unknown) function $t : X \rightarrow T$ that maps 
  input instances $x \in X$ to output targets $t(x) \in T$:
  - **Classification**:
    - The output $t(x) \in T$ is one of a finite set of discrete categories.
  - **Regression**:
    - The output $t(x) \in T$ is continuous, or has a continuous component.

- Target function $t(x)$ is known (only) through (noisy) set of training examples:
  $$(x_1, t_1), (x_2, t_2), \ldots, (x_n, t_n)$$
Supervised Learning

- **Task** = learn an (unknown) function $t : X \rightarrow T$ that maps input instances $x \in X$ to output targets $t(x) \in T$:
  - function $t$ is known (only) through (noisy) set of training examples:
    - Training/Test data: $(x_1,t_1), (x_2,t_2), \ldots (x_n,t_n)$

- **Task** = build a function $h(x)$ such that:
  - $h$ matches $t$ well on the *training data*:
    => $h$ is able to fit data that it has seen.
  - $h$ also matches target $t$ well on *test data*:
    => $h$ is able to generalize to unseen data.
Parametric Approaches to Supervised Learning

• **Task** = build a function $h(x)$ such that:
  - $h$ matches $t$ well on the training data:
    $=> h$ is able to fit data that it has seen.
  - $h$ also matches $t$ well on test data:
    $=> h$ is able to generalize to unseen data.

• **Task** = choose $h$ from a “nice” *class of functions* that depend on a vector of parameters $w$:
  - $h(x) ≡ h_w(x) ≡ h(w,x)$
  - what classes of functions are “nice”?
# Linear Regression

1. (Simple) Linear Regression
   - House price prediction

2. Linear Regression with Polynomial Features
   - Polynomial curve fitting
   - Regularization
   - Ridge regression

3. Multiple Linear Regression
   - House price prediction
   - Normal equations
House Price Prediction

• Given the floor size in square feet, predict the selling price:
  – $x$ is the size, $t$ is the price
  – Need to learn a function $h$ such that $h(x) \approx t(x)$.

• Is this classification or regression?
  – **Regression**, because price is real valued.
    • and there are many possible prices.
  – (Simple) linear regression, because one input value.
  – Would a problem with only two labels $t_1 = 0.5$ and $t_2 = 1.0$ still be regression?
House Prices in Athens

50 houses, randomly selected from the 106 houses or townhomes:
• sold recently in Athens, OH.
• built 1990 or later.
House Prices in Athens

[Scatter plot showing the relationship between price (x $1000) and floor size (square feet).]
Parametric Approaches to Supervised Learning

- **Task** = build a function $h(x)$ such that:
  - $h$ matches $t$ well on the training data:
    => $h$ is able to fit data that it has seen.
  - $h$ also matches $t$ well on test data:
    => $h$ is able to generalize to unseen data.

- **Task** = choose $h$ from a “nice” class of functions that depend on a vector of parameters $w$:
  - $h(x) = h_w(x) = h(w,x)$
  - what classes of functions are “nice”?
House Prices in Athens
House Prices in Athens
Linear Regression

- Use a linear function approximation:
  \[ h_w(x) = w^T x = [w_0, w_1]^T [1, x] = w_1 x + w_0. \]
  - \( w_0 \) is the intercept (or the bias term).
  - \( w_1 \) controls the slope.

- Learning = optimization:
  - Find \( w \) that obtains the best fit on the training data, i.e. find \( w \) that minimizes the sum of square errors:

\[
J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2
\]

\[
\hat{w} = \arg\min_w J(w)
\]
Univariate Linear Regression

- Learning = finding the “right” parameters $\mathbf{w}^T = [w_0, w_1]$
  - Find $\mathbf{w}$ that minimizes an error function $E(\mathbf{w}) = J(\mathbf{w})$ which measures the misfit between $h(x_n, \mathbf{w})$ and $t_n$.
  - Expect that $h(x, \mathbf{w})$ performing well on training examples $x_n \Rightarrow h(x, \mathbf{w})$ will perform well on arbitrary test examples $x \in X$.

- Sum-of-Squares error function:
  \[ J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_{\mathbf{w}}(x_n) - t_n)^2 \]
Minimizing Sum-of-Squares Error

- **Sum-of-Squares** error function:
  \[ J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2 \]

- How do we find \( w^* \) that minimizes \( E(w) \)?
  \[ \hat{w} = \arg \min_w J(w) \]

- Least Square solution is found by solving a system of 2 linear equations:
  \[
  \begin{align*}
  w_0 N + w_1 \sum_{n=1}^{N} x_n &= \sum_{n=1}^{N} t_n \\
  w_0 \sum_{n=1}^{N} x_n + w_1 \sum_{n=1}^{N} x_n^2 &= \sum_{n=1}^{N} t_n x_n
  \end{align*}
  \]
Polynomial Basis Functions

• \textbf{Q}: What if the raw feature is insufficient for good performance?
  – Example: non-linear dependency between label and raw feature.

• \textbf{A}: Engineer / Learn higher-level features, as functions of the raw feature.

• \textbf{Polynomial curve fitting}:
  – Add new features, as polynomials of the original feature.
Regression: Curve Fitting

- **Training**: Build a function $h(x)$, based on (noisy) training examples $(x_1,t_1), (x_2,t_2), \ldots (x_N,t_N)$
Regression: Curve Fitting

• **Training**: Build a function $h(x)$, based on (noisy) training examples $(x_1, t_1), (x_2, t_2), \ldots (x_N, t_N)$
Regression: Curve Fitting

• **Testing**: for arbitrary (unseen) instance $x \in X$, compute target output $h(x)$; want it to be close to $t(x)$. 
Regression: Polynomial Curve Fitting

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

parameters features
Polynomial Curve Fitting

- **Parametric model:**
  \[ h(x) = h(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j \]

- **Polynomial curve fitting is (Multiple) Linear Regression:**
  \[ \mathbf{x} = [1, x, x^2, \ldots, x^M]^T \]
  \[ h(x) = h(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} \]

- **Learning = minimize the Sum-of-Squares error function:**
  \[ \mathbf{\hat{w}} = \arg \min_{\mathbf{w}} J(\mathbf{w}) \quad J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_\mathbf{w}(\mathbf{x}_n) - t_n)^2 \]
Sum-of-Squares Error Function

- How to find \( w^* \) that minimizes \( E(w) \), i.e. \( w^* = \arg \min_w E(w) \)
- Solve \( \nabla J(w) = 0 \).

\[ J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2 \]
Polynomial Curve Fitting

• *Least Square* solution is found by solving a set of $M + 1$ linear equations:

$$Aw = T$$

$$\sum_{j=0}^{M} A_{ij} w_j = T_i, \text{ where } A_{ij} = \sum_{n=1}^{N} x_n^{i+j}, \text{ and } T_i = \sum_{n=1}^{N} t_n x_n^i$$

• Prove it.
Polynomial Curve Fitting

- **Generalization** = how well the parameterized $h(x,w)$ performs on arbitrary (unseen) test instances $x \in X$.

- Generalization performance depends on the value of $M$. 
$0^{th}$ Order Polynomial
1\textsuperscript{st} Order Polynomial
3rd Order Polynomial
9th Order Polynomial
Polynomial Curve Fitting

- **Model Selection**: choosing the order $M$ of the polynomial.
  - Best generalization obtained with $M = 3$.
  - $M = 9$ obtains poor generalization, even though it fits training examples perfectly:
    - But $M = 9$ polynomials subsume $M = 3$ polynomials!

- **Overfitting** = good performance on training examples, poor performance on test examples.
Overfitting

- Measure fit using the Root-Mean-Square (RMS) error:

\[ E_{RMS}(w) = \sqrt{\frac{\sum_n (w^T x_n - t_n)^2}{N}} \]

- Use 100 random test examples, generated in the same way:
Over-fitting and Parameter Values

<table>
<thead>
<tr>
<th></th>
<th>$M = 0$</th>
<th>$M = 1$</th>
<th>$M = 3$</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></td>
<td>-1.27</td>
<td>7.99</td>
<td>232.37</td>
</tr>
<tr>
<td>$w_2^*$</td>
<td></td>
<td></td>
<td>-25.43</td>
<td>-5321.83</td>
</tr>
<tr>
<td>$w_3^*$</td>
<td></td>
<td></td>
<td>17.37</td>
<td>48568.31</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>
Overfitting vs. Data Set Size

- More training data $\Rightarrow$ less overfitting.
- What if we do not have more training data?
  - Use regularization.
Regularization

- **Parameter norm penalties** (term in the objective).
- Limit parameter norm (constraint).
- Dataset augmentation.
- Dropout.
- Ensembles.
- Semi-supervised learning.
- Early stopping.
- Noise robustness.
- Sparse representations.
- Adversarial training.
Regularization

- Penalize large parameter values:

\[
J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2 + \frac{\lambda}{2} \|w\|^2
\]

regularizer

\[
w^* = \arg \min_{w} E(w)
\]
9th Order Polynomial with Regularization

\[ \ln \lambda = -18 \]
9\textsuperscript{th} Order Polynomial with Regularization
How do we find the optimal value of $\lambda$?
Model Selection

- Put aside an independent *validation set*.
- Select parameters giving best performance on validation set.

\[ \ln \lambda \in \{-40, -35, -30, -25, -20, -15\} \]

<table>
<thead>
<tr>
<th>(\ln \lambda)</th>
<th>-40</th>
<th>-35</th>
<th>-30</th>
<th>-25</th>
<th>-20</th>
<th>-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E_{\text{RMS}})</td>
<td>1.05</td>
<td>0.30</td>
<td><strong>0.25</strong></td>
<td>0.27</td>
<td>0.52</td>
<td>0.55</td>
</tr>
</tbody>
</table>
K-fold Cross-Validation

K-fold Cross-Validation

- Split the training data into K folds and try a wide range of tuning parameter values:
  - split the data into K folds of roughly equal size
  - iterate over a set of values for $\lambda$
    - iterate over $k=1,2,..., K$
      - use all folds except $k$ for training
      - validate (calculate test error) in the $k$-th fold
    - error[$\lambda$] = average error over the K folds
  - choose the value of $\lambda$ that gives the smallest error.

Model Evaluation

• K-fold evaluation
  – randomly partition dataset in K equally sized subsets P₁, P₂, … Pₖ
  – for each fold i in {1, 2, …, k}:
    • test on Pᵢ, train on P₁ ∪ … ∪ Pᵢ₋₁ ∪ Pᵢ₊₁ ∪ … ∪ Pₖ
  – compute average error/accuracy across K folds.
Multiple Linear Regression

• *Q*: What if the raw feature is insufficient for good performance?
  – Example: house prices depend not only on *floor size*, but also number of *bedrooms, age, location, …*

• *A*: Use **Multiple Linear Regression**.
Multiple Linear Regression

• Polynomial curve fitting:
  \[ \mathbf{x} = [1, x, x^2, ..., x^M]^T \]
  \[ = [x_0, x_1, ..., x_M]^T \]
  \[ h(x) = h(x, \mathbf{w}) = \mathbf{w}^T \mathbf{x} \]

• Multiple linear regression:
  \[ \mathbf{x} = [x_0, x_1, ..., x_M]^T \]
  \[ h(x) = h(x, \mathbf{w}) = \mathbf{w}^T \mathbf{x} \]

• Training examples: \((\mathbf{x}^{(1)}, t_1), (\mathbf{x}^{(2)}, t_2), \ldots (\mathbf{x}^{(N)}, t_N)\)
Multiple Linear Regression

- **Learning** = minimize the Sum-of-Squares error function:

  $$\hat{w} = \arg \min_w J(w) \quad J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x^{(n)}) - t_n)^2$$

- Computing the gradient $\nabla J(w)$ and setting it to zero:

  $$\sum_{n=1}^{N} (w^T x^{(n)} - t_n) x^{(n)} = 0$$

- Solving for $w$ yields $w = (X^T X)^{-1} X^T t$
  - Prove it.
Normal Equations

• Solution is \( w = (X^T X)^{-1} X^T t \)

• \( X \) is the data matrix, or the **design matrix**:

\[
X = \begin{pmatrix}
(x^{(1)^T} \\
x^{(2)^T} \\
... \\
(x^{(N)^T})
\end{pmatrix} = \begin{pmatrix}
x_0^{(1)} & x_1^{(1)} & ... & x_M^{(1)} \\
x_0^{(2)} & x_1^{(2)} & ... & x_M^{(2)} \\
... & ... & ... & ...
\end{pmatrix}
\]

• \( t = [t_1, \ t_2, \ ..., \ t_N]^T \) is the vector of labels.
Ridge Regression

- Multiple linear regression with L2 regularization:

\[
J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2 + \frac{\lambda}{2} \|w\|^2
\]

\[
\hat{w} = \arg \min_w J(w)
\]

- Solution is \( w = (\lambda NI + X^TX)^{-1}X^Tt \)
  - Prove it.
Regularization: Ridge vs. Lasso

- Ridge regression:
  \[ J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2 + \frac{\lambda}{2} \sum_{j=1}^{M} w_j^2 \]

- Lasso:
  \[ J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2 + \frac{\lambda}{2} \sum_{j=1}^{M} |w_j| \]
  - If \( \lambda \) is sufficiently large, some of the coefficients \( w_j \) are driven to 0
    \( \Rightarrow \) \textit{sparse} model.
Regularization: Ridge vs. Lasso
Regularization

- Regularization alleviates overfitting when using models with high capacity (e.g. high degree polynomials):
  - Want high capacity because we do not know how complicated the data is.

- \( Q \): Can we achieve high capacity when doing curve fitting without using high degree polynomials?

- \( A \): Use piecewise polynomial curves.
  - Example: Cubic spline smoothing.
Cubic Spline Smoothing

- **Cubic spline smoothing** is a regularized version of cubic spline interpolation.
  - **Cubic spline interpolation**: given \( n \) points \( \{(x_i, y_i)\} \), connect adjacent points using cubic functions \( S_i \), requiring that the spline and its first and second derivative remain continuous at all points:
    \[
    S_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i, \quad \forall x \in [x_i, x_{i+1}]
    \]
  - **Cubic spline smoothing**: the spline \( S = \{S_i\} \) is allowed to deviate from the data points and has **low curvature** \( \Rightarrow \) constrained optimization problem with objective:
    \[
    L = \sum_{i=1}^{n} \frac{w_i}{Z} (S_i(x_i) - y_i)^2 + \frac{\lambda}{x_n - x_1} \int_{x_1}^{x_n} |S''(x)|^2 \, dx
    \]
    \[
    w_i = \begin{cases} 
    C, & \text{if } (x_i, y_i) \text{ is a significant local optima} \\
    1, & \text{otherwise}
    \end{cases}
    \]
Cubic Spline Smoothing


Fig. 3. Cubic spline smoothing with $\lambda = e^{-20}$ and $C = 1000$. 
Polynomial Curve Fitting (Revisited)

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

parameters
features
Generalization: Basis Functions as Features

- Generally

\[ y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x) \]

where \( \phi_j(x) \) are known as *basis functions*.

- Typically \( \phi_0(x) = 1 \), so that \( w_0 \) acts as a bias.

- In the simplest case, use linear basis functions: \( \varphi_d(x) = x_d \).
Linear Basis Function Models (1)

- Polynomial basis functions:

  \[ \phi_j(x) = x^j. \]

- Global behavior:
  - A small change in \( x \) affect all basis functions.
Linear Basis Function Models (2)

• Gaussian basis functions:

\[ \phi_j(x) = \exp \left\{-\frac{(x - \mu_j)^2}{2s^2}\right\} \]

• Local behavior:
  – a small change in \( x \) only affects nearby basis functions.
  – \( \mu_j \) and \( s \) control location and scale (width).
Linear Basis Function Models (3)

- Sigmoidal basis functions:
  \[ \phi_j(x) = \sigma \left( \frac{x - \mu_j}{s} \right) \]
  where \( \sigma(a) = \frac{1}{1 + \exp(-a)} \).

- Local behavior:
  - a small change in \( x \) only affect nearby basis functions.
  - \( \mu_j \) and \( s \) control location and scale (slope).