Machine Learning
CS 4900/5900

Lecture 02

Razvan C. Bunescu
School of Electrical Engineering and Computer Science
bunescu@ohio.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*:
    \( \Rightarrow h \) is able to fit data that it has seen.
  - \( h \) also matches target \( t \) well on *test data*:
    \( \Rightarrow h \) 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:
    $\implies h$ is able to fit data that it has seen.
  – $h$ also matches $t$ well on test data:
    $\implies h$ is able to generalize to unseen data.

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

1. Univariate Linear Regression
   - House price prediction

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

3. Multivariate 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.
  – Univariate regression, because one input value.
  – Would a problem with only two labels $t_1 = 0.5$ and $t_2 = 1.0$ still be regression?
50 houses, randomly selected from the 106 houses or townhomes:
- sold recently in Athens, OH.
- built 1990 or later.
House Prices in Athens
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 $w$:
  - $h(x) \equiv h_w(x) \equiv h(w,x)$
  - what classes of functions are “nice”? 

Lecture 01
House Prices in Athens
House Prices in Athens

![Graph showing the relationship between house prices and floor size in Athens.](image)
Univariate 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:
    \[
    J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2
    \]
    \[\hat{w} = \text{argmin}_w J(w)\]

Lecture 01
Univariate Linear Regression

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

- **Sum-of-Squares** error function:

$$J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2$$
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:
  \[ w_0 N + w_1 \sum_{n=1}^{N} x_n = \sum_{n=1}^{N} t_n \]
  \[ w_0 \sum_{n=1}^{N} x_n + w_1 \sum_{n=1}^{N} x_n^2 = \sum_{n=1}^{N} t_n x_n \]
Polynomial Basis Functions

- **Q**: What if the raw feature is insufficient for good performance?
  - Example: non-linear dependency between label and raw feature.

- **A**: Engineer [CS 4900] /Learn [CS 6890] higher-level features, as functions of the raw feature.

- **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 \quad features
Polynomial Curve Fitting

• Parametric model:

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

• Polynomial curve fitting is (Multivariate) Linear Regression:

\[ x = \begin{bmatrix} 1, x, x^2, \ldots, x^M \end{bmatrix}^T \]

\[ h(x) = h(x, w) = w^T x \]

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

\[ \hat{w} = \arg \min_w J(w) \quad J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2 \]
Sum-of-Squares Error Function

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

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

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

$$\mathbf{A}\mathbf{w} = \mathbf{T}$$

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

- **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
9\textsuperscript{th} 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** $\equiv$ 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

\[ \ln \lambda = -18 \]
9th Order Polynomial with Regularization

\[ \ln \lambda = 0 \]
Training & Test error vs. $\ln \lambda$

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>0.25</td>
<td>0.27</td>
<td>0.52</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Lecture 01
Model Evaluation

• K-fold cross-validation
  – randomly partition dataset in K equally sized subsets \( P_1, P_2, \ldots, P_k \)
  – for each fold \( i \) in \( \{1, 2, \ldots, k\} \):
    • test on \( P_i \), train on \( P_1 \cup \ldots \cup P_{i-1} \cup P_{i+1} \cup \ldots \cup P_k \)
    • compute average error/accuracy across K folds.

4-fold cross validation
Multivariate 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 **Multivariate Linear Regression**.
Multivariate Linear Regression

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

• Multivariate linear regression:
  \[ x = [x_0, x_1, ..., x_M]^T \]
  \[ h(x) = h(x, w) = w^Tx \]

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

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

\[
\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} J(\mathbf{w}) \quad J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}^{(n)}) - t_n)^2
\]

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

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

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

- Solution is \( \mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} \)

- \( \mathbf{X} \) is the data matrix, or the **design matrix**:

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

- \( \mathbf{t} = [t_1, t_2, \ldots, t_N]^\top \) is the vector of labels.
Ridge Regression

• Multivariate 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 \( w = (\lambda N I + X^T X)^{-1} X^T t \)
  
  – 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

\( \Rightarrow \) **sparse** model.
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
- Lecture 01
- 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 \textit{basis functions}.

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

- In the simplest case, use linear basis functions : \( \phi_d(x) = x_d \).
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).