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:
    $\Rightarrow$ $h$ is able to fit data that it has seen.
  – $h$ also matches $t$ well on test data:
    $\Rightarrow$ $h$ 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”?
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.
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:
    \[
    J(w) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(x_n) - t_n)^2
    \]
    \[
    \hat{w} = \text{argmin}_w J(w)
    \]
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 $h(\mathbf{x}_n, \mathbf{w})$ and $t_n$.
  - Expect that $h(\mathbf{x}, \mathbf{w})$ performing well on training examples $\mathbf{x}_n \Rightarrow h(\mathbf{x}, \mathbf{w})$ will perform well on arbitrary test examples $\mathbf{x} \in X$.

- Sum-of-Squares error function:
  $$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(\mathbf{x}_n) - t_n)^2$$

Inductive Learning Hypothesis
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:
  \[
  \begin{align*}
  w_0N + w_1 \sum_{n=1}^{N} x_n &= \sum_{n=1}^{N} t_n \\
  w_0 \sum_{n=1}^{N} x_n + w_1 \sum_{n=1}^{N} x_n^2 &= \sum_{n=1}^{N} t_n x_n
  \end{align*}
  \]
Polynomial Basis Functions

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

- **A**: Engineer / Learn 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 $t(x)$. 

![Diagram](image-url)

Learned $h$ and target $t$. The goal is to have $h(x)$ close to $t(x)$ for arbitrary $x \in X$.
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:

\[ x = [1, x, x^2, \ldots, x^M]^T \]

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

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

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

- How to find $\mathbf{w}^*$ that minimizes $E(\mathbf{w})$, i.e. $\mathbf{w}^* = \arg \min_w E(\mathbf{w})$
- 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:

\[
Aw = T
\]

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

• 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 $M$. 
0th Order Polynomial

\[ M = 0 \]
$1^{st}$ Order Polynomial

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{1st_order_polynomial.png}
\end{figure}
3rd Order Polynomial

\[ M = 3 \]

Diagram showing a 3rd order polynomial with data points and a curve fitting through them.
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:

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

- 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>-1.27</td>
<td>7.99</td>
<td></td>
<td>232.37</td>
</tr>
<tr>
<td>$w^*_2$</td>
<td></td>
<td>-25.43</td>
<td>-5321.83</td>
<td></td>
</tr>
<tr>
<td>$w^*_3$</td>
<td></td>
<td>17.37</td>
<td>48568.31</td>
<td></td>
</tr>
<tr>
<td>$w^*_4$</td>
<td></td>
<td></td>
<td>-231639.30</td>
<td></td>
</tr>
<tr>
<td>$w^*_5$</td>
<td></td>
<td></td>
<td>640042.26</td>
<td></td>
</tr>
<tr>
<td>$w^*_6$</td>
<td></td>
<td></td>
<td>-1061800.52</td>
<td></td>
</tr>
<tr>
<td>$w^*_7$</td>
<td></td>
<td></td>
<td>1042400.18</td>
<td></td>
</tr>
<tr>
<td>$w^*_8$</td>
<td></td>
<td></td>
<td>-557682.99</td>
<td></td>
</tr>
<tr>
<td>$w^*_9$</td>
<td></td>
<td></td>
<td>125201.43</td>
<td></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

\[
w^* = \arg \min_w E(w)
\]
9\textsuperscript{th} 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_{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>
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
    - $\text{error}[\lambda] = \text{average error over the K folds}$
      - choose the value of $\lambda$ that gives the smallest error.

Model Evaluation

• **K-fold evaluation**
  – randomly partition dataset in K equally sized subsets $P_1, P_2, \ldots P_k$
  – 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 evaluation
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:**
  \[ \mathbf{x} = [1, x, x^2, ..., x^M]^T \]
  \[ = [x_0, x_1, ..., x_M]^T \]
  \[ h(x) = h(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} \]

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

- **Training examples:** \((\mathbf{x}^{(1)}, t_1), (\mathbf{x}^{(2)}, t_2), ..., (\mathbf{x}^{(N)}, t_N)\)
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.

The Moore-Penrose pseudo-inverse of \( X \).
Normal Equations

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

- $X$ is the data matrix, or the **design matrix**:

$$X = \begin{pmatrix} x^{(1)T} \\ x^{(2)T} \\ \vdots \\ x^{(N)T} \end{pmatrix} = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & \ldots & x_M^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \ldots & x_M^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{(N)} & x_1^{(N)} & \ldots & x_M^{(N)} \end{pmatrix}$$

- $t = [t_1, t_2, \ldots, t_N]^T$ is the vector of labels.

For poly fit:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^M \\ 1 & x_2 & x_2^2 & \ldots & x_2^M \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_N & x_N^2 & \ldots & x_N^M \end{pmatrix}$$
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 \( w = (\lambda N I + X^T X)^{-1} X^T t \)
  - 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.
Figure 3.4  Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector $w$ is denoted by $w^*$. The lasso gives a sparse solution in which $w_1^* = 0$. 
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: \textit{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:
    \[
    S_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i, \forall x \in [x_i, x_{i+1}]
    \]
  - **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: \( \phi_d(x) = x_d \).
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).
Solving Linear Regression using Maximum Likelihood
Least Squares $\iff$ Maximum Likelihood (1)

- Assume observations from a deterministic function $y$ with added Gaussian noise $\epsilon$:

  $$ t = y(x, w) + \epsilon $$

  where
  $$ p(\epsilon|\beta) = \mathcal{N}(\epsilon|0, \beta^{-1}) $$

  which is the same as saying:

  $$ p(t|x, w, \beta) = \mathcal{N}(t|y(x, w), \beta^{-1}). $$

  $\quad = \frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{-\beta \frac{\epsilon^2}{2}}$

  $\quad = \frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{-\beta \frac{(t-y(x,w))^2}{2}}$
Least Squares $\iff$ Maximum Likelihood (1)

- Assume observations from a deterministic function with added Gaussian noise:
  
  $$ t = y(x, w) + \epsilon \quad \text{where} \quad p(\epsilon|\beta) = \mathcal{N}(\epsilon|0, \beta^{-1}) $$

  which is the same as saying:

  $$ p(t|x, w, \beta) = \mathcal{N}(t|y(x, w), \beta^{-1}). $$

- Given observed i.i.d inputs $X = \{x_1, \ldots, x_N\}$ and targets $t = [t_1, \ldots, t_N]^T$, we obtain the likelihood function:

  $$ p(t|X, w, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|w^T\phi(x_n), \beta^{-1}). $$
Least Squares \iff Maximum Likelihood (2)

- Taking the logarithm, we get the log-likelihood function:

\[
\ln p(t|w, \beta) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|w^T \phi(x_n), \beta^{-1}) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(w)
\]

where

\[
E_D(w) = \frac{1}{2} \sum_{n=1}^{N} \left\{ t_n - w^T \phi(x_n) \right\}^2
\]

- \(E_D(w)\) is the sum-of-squares error!
Least Squares $\iff$ Maximum Likelihood (3)

- Minimizing square error $\iff$ maximizing likelihood:

$$w^* = \arg \min_w E_D(w) = w_{ML} = \arg \max_w \ln p(t \mid w, \beta)$$

- How do we find $w$ (and $\beta$)?
Least Squares $\iff$ Maximum Likelihood (4)

- Computing the gradient and setting it to zero yields:

$$\nabla_w \ln p(t|w, \beta) = \beta \sum_{n=1}^{N} \{t_n - w^T \phi(x_n)\} \phi(x_n)^T = 0.$$ 

- Solving for $w$, we get

$$w_{ML} = \left(\Phi^T \Phi\right)^{-1} \Phi^T t$$

The Moore-Penrose pseudo-inverse, $\Phi^\dagger$.

where

$$\Phi = \begin{pmatrix}
\phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_{M-1}(x_1) \\
\phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_{M-1}(x_2) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_{M-1}(x_N)
\end{pmatrix}.$$
Least Squares $\iff$ Maximum Likelihood (5)

- Minimizing square error $\iff$ maximizing likelihood:
  \[
  \mathbf{w}^* = \arg \min_{\mathbf{w}} E_D(\mathbf{w}) = \mathbf{w}_{ML} = \arg \max_{\mathbf{w}} \ln p(\mathbf{t} | \mathbf{w}, \beta)
  \]

- Maximizing with respect to $\mathbf{w}$ gives:
  \[
  \mathbf{w}_{ML} = \left( \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}
  \]

- Maximizing with respect to $\beta$ gives:
  \[
  \frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^{N} \left\{ t_n - \mathbf{w}_{ML}^T \phi(\mathbf{x}_n) \right\}^2
  \]
Regularized Least Square

- Consider the error function:

\[ E_D(w) + \lambda E_W(w) \]

Data term + Regularization term

- With the sum-of-squares error function and a quadratic regularizer, we get:

\[
\frac{1}{2} \sum_{n=1}^{N} \left( t_n - w^T \phi(x_n) \right)^2 + \frac{\lambda}{2} w^T w
\]

which is minimized by:

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

\( \lambda \) is called the regularization coefficient.
Regularized Least Square <=> Maximum A Posteriori (MAP)

- Define a conjugate prior over $w$:
  
  $$p(w) = \mathcal{N}(w|0, \alpha^{-1}I)$$

- We also have the likelihood function:
  
  $$p(t|X, w, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|w^T\phi(x_n), \beta^{-1}).$$

- Bayes to combine prior with the likelihood $\Rightarrow$ posterior:
  
  $$p(w|t) = \frac{p(t|w)p(w)}{p(t)}$$

  $$p(w|X, t, \alpha, \beta) = \frac{p(t|w, X, \beta)p(w|\alpha)}{p(t|X, \alpha, \beta)} \propto p(t|w, X, \beta)p(w|\alpha)$$
Regularized Least Square \(\iff\) Maximum A Posteriori (MAP)

- Taking the logarithm of the posterior distribution:
  \[
  \ln p(w \mid t) = -\frac{\beta}{2} \sum_{n=1}^{N} \{t_n - w^T \varphi(x_n)\}^2 - \frac{\alpha}{2} w^T w + \text{const}
  \]

- The MAP estimate of \(w\) is:
  \[
  w_{MAP} = \arg \max_w \ln p(w \mid t)
  \]
  \[
  = \arg \max_w -\frac{1}{2} \sum_{n=1}^{N} \{t_n - w^T \varphi(x_n)\}^2 - \frac{\alpha}{2} w^T w
  \]
  \[
  = \arg \min_w \frac{1}{2} \sum_{n=1}^{N} \{t_n - w^T \varphi(x_n)\}^2 + \frac{\lambda}{2} w^T w
  \]
  \[
  = \arg \min_w E_D(w) + E_W(w)
  \]
Regularized Least Square <=> Maximum A Posteriori (MAP)

- Define a conjugate prior over $w$:

$$p(w) = \mathcal{N}(w|0, \alpha^{-1}I)$$

- We also have the likelihood function:

$$p(t|X, w, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|w^T\phi(x_n), \beta^{-1}).$$

- Using Bayes and results for marginal and conditional Gaussian distributions, gives the posterior

$$p(w|t) = \mathcal{N}(w|m_N, S_N) \quad \text{where} \quad \begin{cases} m_N &= \beta S_N \Phi^T t \\ S_N^{-1} &= \alpha I + \beta \Phi^T \Phi. \end{cases}$$

$$\hat{w} = m_N = \left(\frac{\alpha}{\beta}I + \Phi^T\Phi\right)^{-1} \Phi^T t = (\lambda I + \Phi^T\Phi)^{-1}\Phi^T t$$
Supplemental Readings

- **PRML:**
  - Section 1.1 (Polynomial curve fitting).
  - Section 1.2 (up to and including 1.2.5).
  - Section 3.1.4 (Regularized least squares).