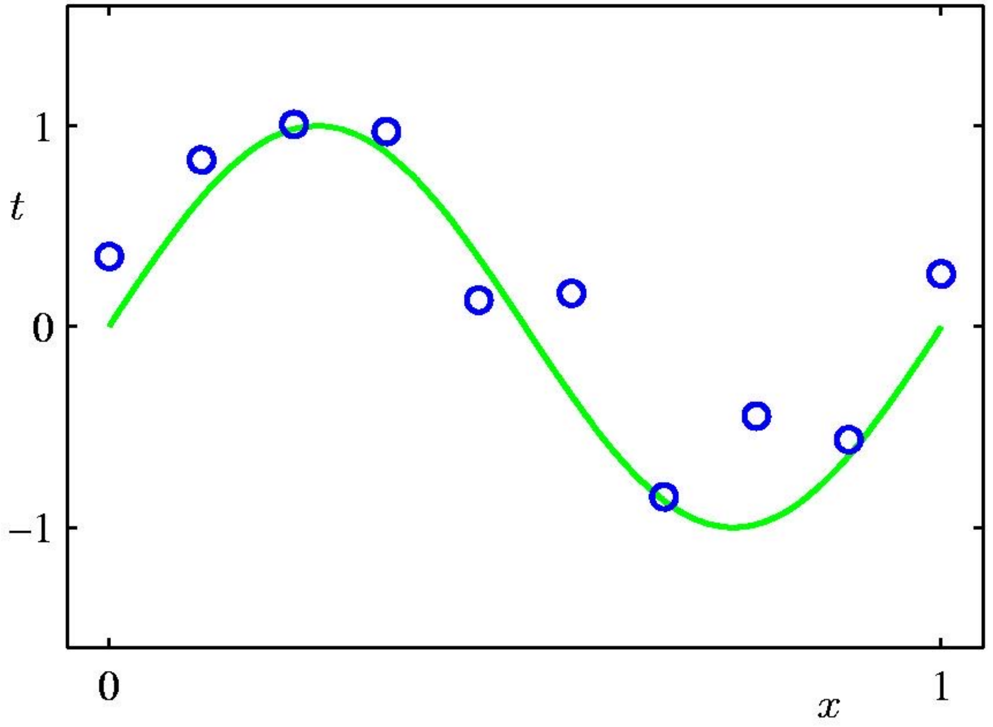
CPE/EE 695: Applied Machine Learning

Lecture 2-1: Linear Regression

Dr. Shucheng Yu, Associate Professor
Department of Electrical and Computer Engineering
Stevens Institute of Technology

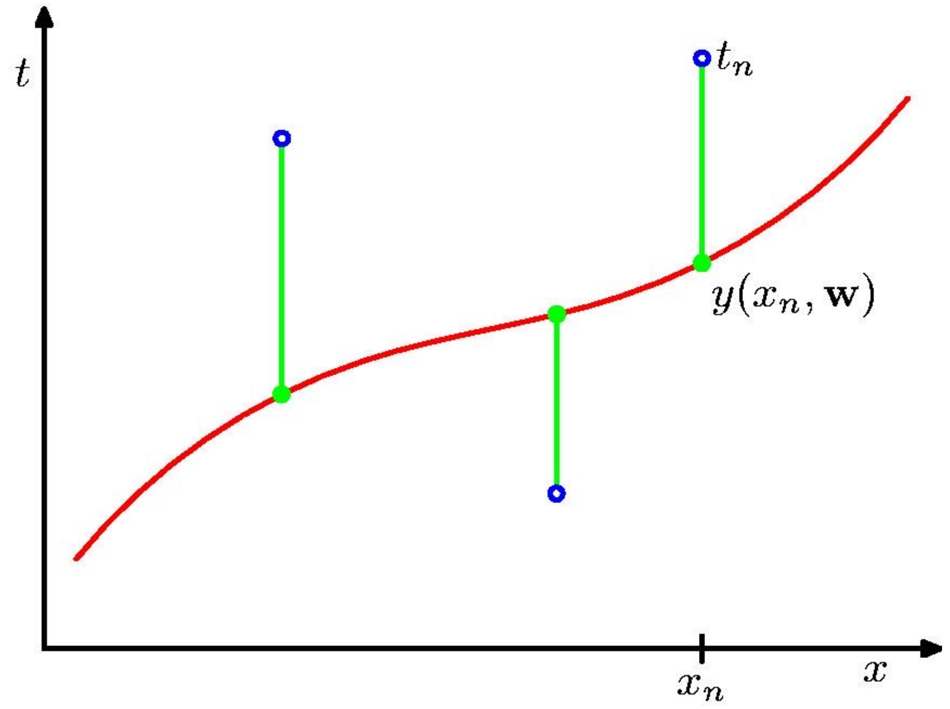
Polynomial Curve Fitting



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

Polynomial Regression

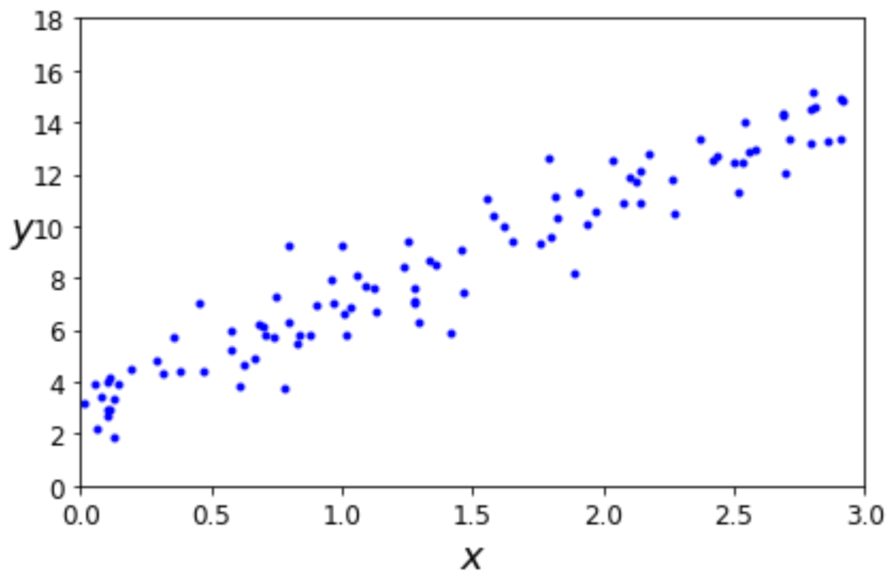
Sum-of-Squares Error Function



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

Minimize E(w) for unknown w. (maximum likelihood)

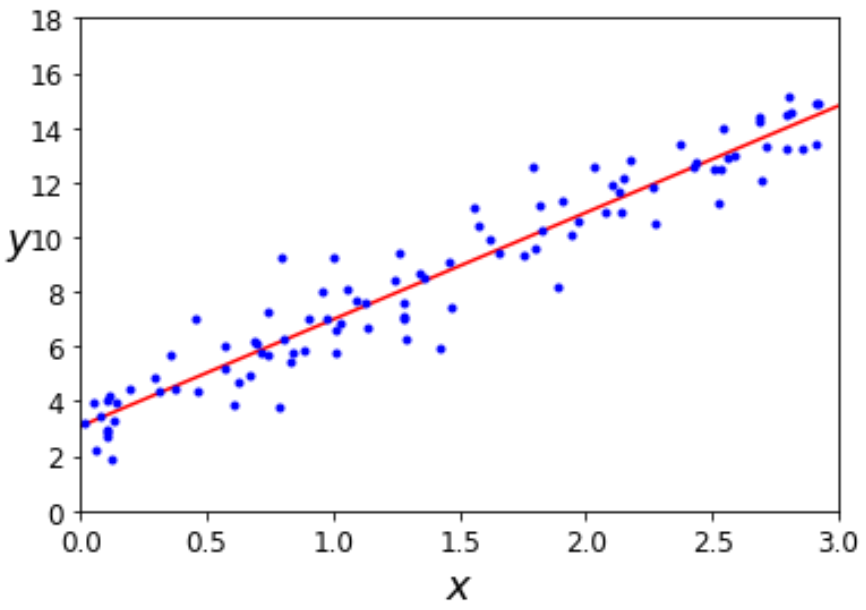
Linear Regression



Observation: Data points seem around a line

Let's assume the line is: $\hat{y} = w_0 + w_1 x_1$

Linear Regression



Observation: Data points seem around a line

Let's assume the line is: $\hat{y} = w_0 + w_1 x_1$

Learning the linear model is to estimate the unknown coefficients w_0 and w_1 that make the line fit the data points.

Linear Regression

More generally, we can extend the linear model to (n+1)-dimension:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Due to errors/noises (e.g., caused by sensors), observed or measured value of $\hat{y}$ would be:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \boldsymbol{\varepsilon}$$

Learning the linear model is to estimate the unknown coefficients that **make the linear model fit the noisy data samples**.

For simplicity, we usually use vectorized representation: $\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \cdot \boldsymbol{x}$

Linear Regression

Linear model to be learned:

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \cdot \boldsymbol{x}$$

Observed noisy data (x, y), where:

$$y = w^T \cdot x + \varepsilon$$

Linear Regression

Linear model to be learned:

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \cdot \boldsymbol{x}$$

Observed noisy data (x, y), where:

$$y = w^T \cdot x + \varepsilon$$

Performance measure**: mean Square Error (MSE) cost function:**

$$MSE(X, h_w) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^T \cdot \boldsymbol{x}^{(\boldsymbol{i})} - \mathrm{y}^{(\mathrm{i})})^2$$

More frequently, root mean square error (RMSE) = $\sqrt{MSE(W)}$ is used.
Both leads to the same solution:

$\widehat{w} = argmin\big(MSE(w)\big)$, i.e., the w that minimizes $MSE(w)$.

Linear Regression

The Normal Equation

Actually, there is a closed-form solution (the normal equation) to the problem.

$$\hat{\boldsymbol{w}} = (\boldsymbol{X}^{\boldsymbol{T}} \cdot \boldsymbol{X})^{-\mathbf{1}} \cdot \boldsymbol{X}^{\boldsymbol{T}} \cdot \boldsymbol{y}$$

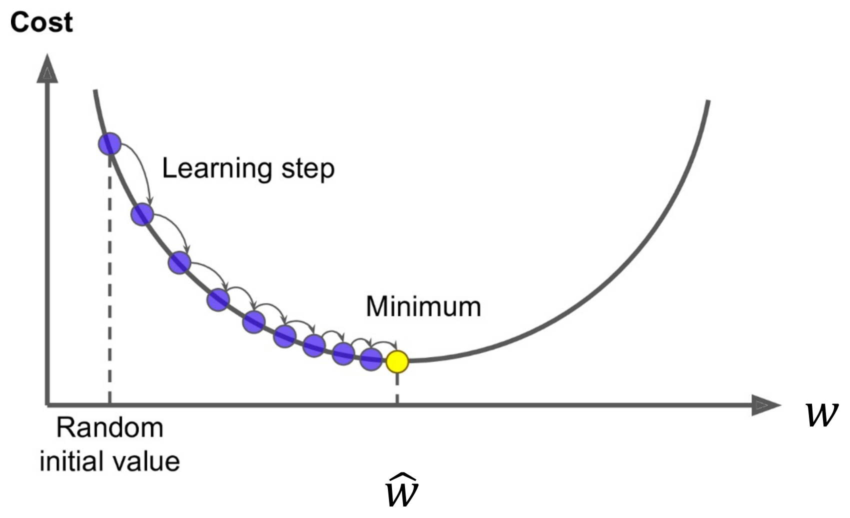
where $\hat{w}$: the value of that minimizes the cost function

X : the training data set, i.e., the matrix $(x^{(1)}\ x^{(2)} \ldots\ x^{(m)})^{\mathrm{T}}$.

$\boldsymbol{y}$: the vector of target values containing $(\mathrm{y}^{(1)}\ \mathrm{y}^{(2)} \ldots\ \mathrm{y}^{(\mathrm{m})})^{\mathrm{T}}$.

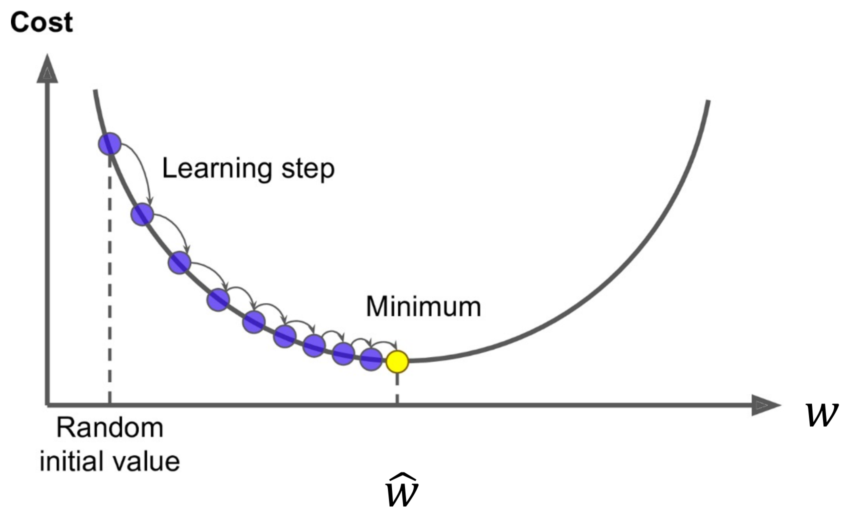
The complexity for computing the normal equation is about $O(n^{2.4})$ to $O(n^{3})$ where n is the number of features. When n is large (e.g., tens of thousands), computing the normal equation will become very slow.

Gradient Descent



$$w^{(next\ step)} = w - \lambda \cdot \nabla_w MSE(w)$$

Gradient Descent



1. Start with random values of w, obtaining a random polynomial: $\hat{y} = h_w(x) = \boldsymbol{w}^T \cdot \boldsymbol{x}$
2. Compute MSE using $\hat{y}$ and data (x, y)
3. If MSE is not small enough, update w with the gradient descent approach.
4. Repeat Steps 2 and 3.

$$MSE(X, h_w) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^T \cdot \boldsymbol{x}^{(\boldsymbol{i})} - \mathrm{y}^{(\mathrm{i})})^2$$

$$w^{(next\ step)} = w - \lambda \cdot \nabla_w MSE(w)$$

What is Grade Descent?

A first-order iterative optimization algorithm to find the minimum of a multivariable function F(**x**).

Rational:

If F(**x**) is differentiable around a point A, F decreases **fastest** from A in the direction of negative gradient of F(x) at A (i.e., $-\boldsymbol{\nabla F(A)}$). In other words, let

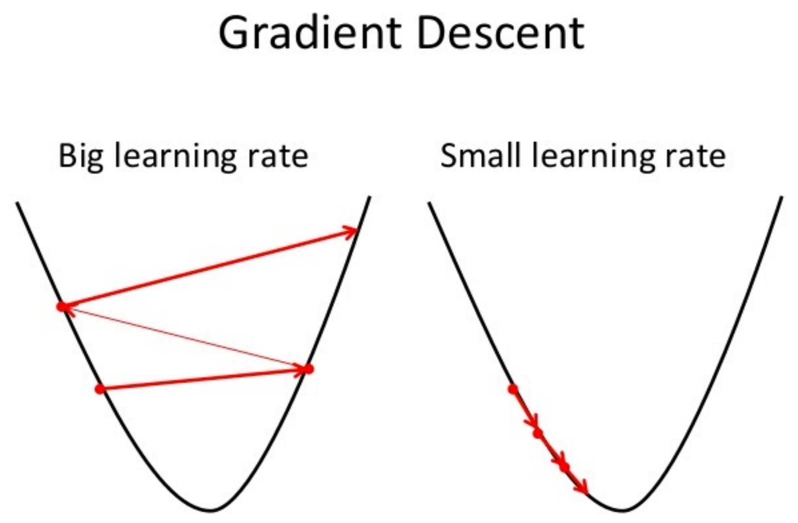
$$\boldsymbol{A_{n+1} = A_n - \lambda * \nabla F(A)}$$

for small enough $\boldsymbol{\lambda}$, we have $\boldsymbol{F(A_{n+1}) \leq F(A_n)}$.

Through a set of such points $\boldsymbol{A_0, A_1}$, ..., it converges to a **local minimum**.

If function F(**x**) is **convex**, the local minimum is the **global minimum**.

Learning Rate



A big learning rate may cause the algorithm diverge without finding a good solution.

A small learning rate may cause the algorithm to converge very slowly.

Appropriate learning rate λ is very important

Gradient Descent

Gradient descent approach computes gradient $\nabla_{w_j} MSE(w) = \frac{\partial}{\partial w_j}$ for w_j at each iteration:

$$\frac{\partial}{\partial w_j} = \frac{2}{m} \sum_{i=1}^{m} (w^T \cdot x^{(i)} - y^{(i)})\, x_j{}^{(i)}$$

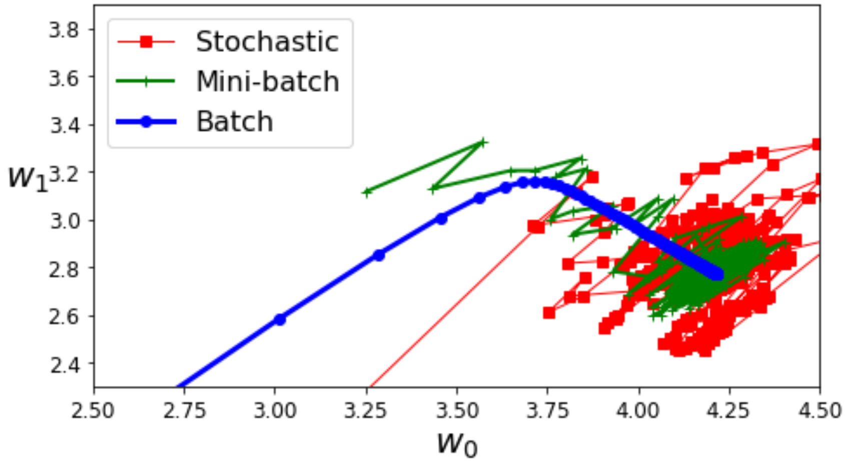
1) **Batch Gradient Descent**
Using *ALL* data sets to calculate the gradient and update w, i.e., *m* = *N,* where *N* is the number of all data sets.

2) **Stochastic Gradient Descent**
Using *ONE* RANDOM sample to calculate the gradient and update w, i.e., *m* = *1*.

3) **Mini-batch Gradient Descent**
Using *a small random set* of samples, i.e., *m* = *r* where *r* ≪ *m*.

Gradient Descent

Three Modes – Batch, Stochastic and Mini-batch GD



Picture generated with Github code of textbook 2 (Aurelien Geron)

In practice, **Mini-batch** GD is widely used because of the good tradeoffs. The size of the “mini batches” is a hyperparameter.

Convergence rate:

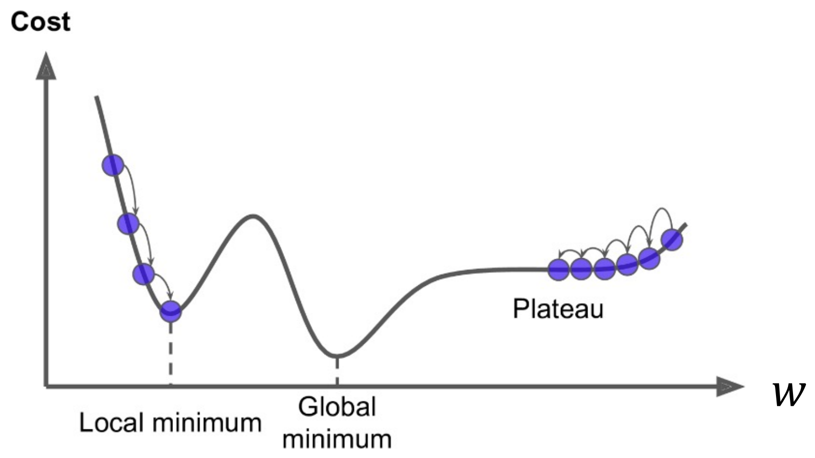
- It takes lest iterations for the Batch GD to converge
- It takes the most iterations for the Stochastic GD to converge.
- Mini-batch is in the middle.

Computation at each iteration:

- Batch GD uses all training example to compute the gradient
- Stochastic GD uses one random training example to compute the gradient
- Mini-batch is in the middle

Gradient Descent

Gradient Descent:



Local minimum issue

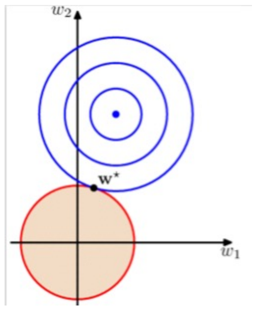
Batch Gradient Descent: more likely to have local minimum
Stochastic Gradient Descent: less likely to have local minimum
Mini-Batch Gradient Descent: in the middle of the two.

Regularized Linear Models

Ridge Regression

Cost function: $E(w) = \sum_{i=1}^{m}(\boldsymbol{w}^T \cdot \boldsymbol{x}^{(i)} - \mathrm{y}^{(i)})^2 + \lambda \sum_{i=1}^{m} {w_i}^2$

It has a closed-form solution $\mathbf{w} = \left(\lambda \mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}.$

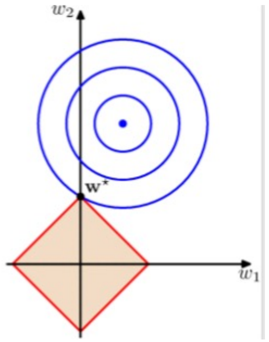


Lasso Regression

Cost function: $E(w) = \sum_{i=1}^{m}(\boldsymbol{w}^T \cdot \boldsymbol{x}^{(i)} - \mathrm{y}^{(i)})^2 + \lambda \sum_{i=1}^{m} |w_i|$

No closed-from solution for w;

But it tends to eliminate the weights of least important features.



Elastic Net

Cost function: $E(w) = \sum_{i=1}^{m}(\boldsymbol{w}^T \cdot \boldsymbol{x}^{(i)} - \mathrm{y}^{(i)})^2 + \lambda_1 \sum_{i=1}^{m} {w_i}^2 + \lambda_2 \sum_{i=1}^{m} |w_i|$

In the middle of Ridge and Lasso.

Usually is preferred over Lasso or Ridge.

Cost Functions

Mean Square Error (MSE):

$$MSE(X,h) = \frac{1}{m}\sum_{i=1}^{m}(h(\boldsymbol{x}^{(i)}) - \mathrm{y}^{(\mathrm{i})})^2$$

Root Mean Square Error (RMSE):

$$RMSE(X,h) = \sqrt{MSE}$$ (Euclidean norm)

Mean Absolute Error (MAE):

$$MAE(X,h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(x^{(i)}\right) - y^{(i)}\right|$$ (Manhattan norm)

$\boldsymbol{l_k}$ ***norm*** of a vector v with n elements: $||v||_k = (|v_0|^k + \cdots + |v_n|^k)^{\frac{1}{k}}$

The higher the norm index, the more it focuses on large values and neglect small ones. Therefore, RMSE is more sensitive to outliers than MAE.

Basis Function

Convert Non-Linear Model to Linear Model

Many models are non-linear, for example, polynomial function:

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \cdots + w_n x^n = \sum_{i=0}^{n} w_i x^i$$

To utilize linear regression, we can use a so-called *basis function* $\phi_i(x)$to do a “mapping”:

$$\phi_i(x) = x^i$$

Therefore, the polynomial function is represented as:

$$\hat{y} = \sum_{i=0}^{n} w_i \phi_i(x)$$

More generally, the basis function $\phi_i(x)$ can be any linear or non-linear function. We can train the model using linear regression. The normal equation becomes:

$$\hat{w} = (\Phi^T \cdot \Phi)^{-1} \cdot \Phi^T \cdot y, \quad \text{where } \Phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_m(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \dots & \phi_m(x_2) \\ & \dots & & \dots \\ \phi_0(x_n) & \phi_1(x_n) & \dots & \phi_m(x_n) \end{pmatrix}$$

1870

STEVENS

INSTITUTE *of* TECHNOLOGY

THE INNOVATION UNIVERSITY®

stevens.edu