

Lecture 5 – Multivariate Linear Regression

CS 335

Dan Sheldon

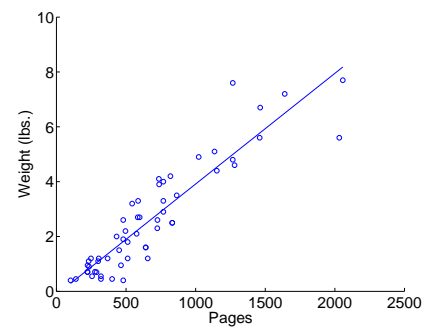
Linear algebra wrap-up

- ▶ Inverse (other slides)

Multivariate linear regression

- ▶ Model
- ▶ Cost function
- ▶ Normal equations
- ▶ Gradient descent
- ▶ Features

Book Data



$$y = 0.004036x - .122291$$

Book Data

Can we predict better with multiple features?

Width	Thickness	Height	# Pages	Hardcover	Weight
8	1.8	10	1152	1	4.4
8	0.9	9	584	1	2.7
7	1.8	9.2	738	1	3.9
6.4	1.5	9.5	512	1	1.8

Training data

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})$$

$\mathbf{x}^{(i)}$ is a **feature vector**

Multivariate Linear Regression

- ▶ Input: $\mathbf{x} \in \mathbb{R}^d$
- ▶ Output: $y \in \mathbb{R}$
- ▶ Model (hypothesis class): ?
- ▶ Cost function: ?

Model

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} = \mathbf{x}^T \boldsymbol{\theta}$$

(Augment feature vector with 1)

Geometry of high dimensional linear functions

n -dimensional linear function $h_{\boldsymbol{\theta}} : \mathbb{R}^n \rightarrow \mathbb{R}$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

MATLAB demo

- ▶ Contours = parallel hyperplanes
- ▶ Gradient = $\boldsymbol{\theta}$ (a vector, orthogonal to contours)
- ▶ The norm $\|\boldsymbol{\theta}\|$ can be interpreted as slope

The Problem

Find $\boldsymbol{\theta}$ such that

$$y^{(i)} \approx h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \quad i = 1, \dots, m$$

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{bmatrix} \approx \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix}$$

$$\mathbf{y} \approx X\boldsymbol{\theta}$$

Inputs: Data Matrix and Label Vector

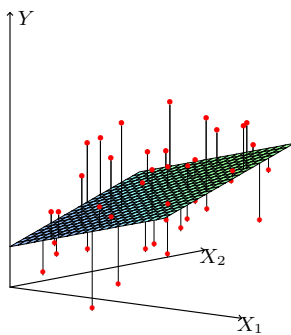
Data matrix, label vector

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{bmatrix}$$

Width	Thickness	Height	# Pages	Hardcover	Weight
8	1.8	10	1152	1	4.4
8	0.9	9	584	1	2.7
7	1.8	9.2	738	1	3.9
6.4	1.5	9.5	512	1	1.8

Illustration

Find $\boldsymbol{\theta}$ such that $y^{(i)} \approx h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \quad i = 1, \dots, m$



Elements of Statistical Learning (2nd Ed.) ©Hastie, Tibshirani & Friedman 2009 Chap 3

Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Exercise: write this succinctly in matrix-vector notation

Cost Function

Answer:

$$J(\boldsymbol{\theta}) = \frac{1}{2}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

The Problem

Given training data \mathbf{X} and \mathbf{y} , find $\boldsymbol{\theta}$ to minimize cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{2}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Solution 1: Normal Equations

Normal equations

$$\boldsymbol{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Heuristic derivation:

Proper Approach

- ▶ Set all partial derivatives to zero

$$0 = \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

- ▶ Solve a system of $n + 1$ linear equations for $\theta_0, \dots, \theta_n$
- ▶ Tedious, but leads to normal equations

Gradient!

Definition: gradient = vector of partial derivatives:

$$\nabla J(\boldsymbol{\theta}) := \begin{bmatrix} \frac{\partial}{\partial \theta_0} J(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\boldsymbol{\theta}) \end{bmatrix}$$

- ▶ Also written as $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ or $\frac{d}{d\boldsymbol{\theta}} J(\boldsymbol{\theta})$
- ▶ Fact: the gradient is a *vector* that points in the steepest uphill direction

Aside: Matrix Calculus

Succinct (and cool!) way to solve for normal equations:

$$\begin{aligned} 0 = \nabla J(\boldsymbol{\theta}) &= \frac{d}{d\boldsymbol{\theta}} \frac{1}{2}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \\ 0 &= (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T \mathbf{X} \\ 0 &= \mathbf{X}^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \\ \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} &= \mathbf{X}^T \mathbf{y} \\ \boldsymbol{\theta} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

(Note: not responsible vector derivative in first line, but should understand rest of derivation.)

Solution 2: Gradient Descent

1. Initialize $\theta_0, \theta_1, \dots, \theta_n$ arbitrarily
2. Repeat until convergence

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}), \quad j = 0, \dots, n.$$

Partial derivatives:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \sum_{i=1}^m (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

Vectorized Gradient Descent

1. Initialize $\boldsymbol{\theta}$ arbitrarily
2. Repeat until convergence

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \underbrace{X^T(X\boldsymbol{\theta} - \mathbf{y})}_{\nabla J(\boldsymbol{\theta})}$$

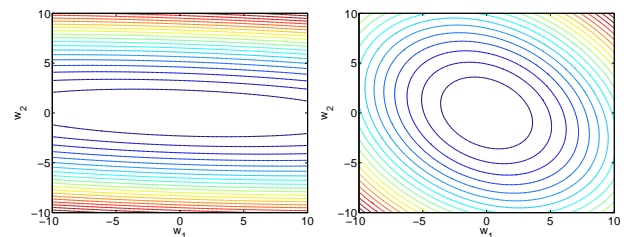
Feature Normalization

- Advice: normalize your features so they have the similar numeric ranges!

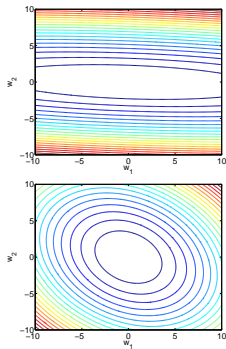
$$\begin{bmatrix} 94 & .99 \\ 116 & 1 \\ 83 & 1.01 \end{bmatrix} \mapsto \begin{bmatrix} -0.22 & -1 \\ 1.09 & 0 \\ -0.87 & 1 \end{bmatrix}$$

Feature Normalization: why?

Example: cost function contours before and after normalization



Feature Normalization: why?



Why is the bottom cost function better?

1. It has a smaller minimum value, which allows you to find a better hypothesis.
2. It is easier to optimize (e.g. can use bigger step size)
3. It is not better. The top one is better (explain why).

Feature Normalization: how?

Adjust columns of data matrix to have mean zero and standard deviation equal to one.

- For each feature j , compute the mean μ_j and standard deviation σ_j of that feature over training set.

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}, \quad \sigma_j = \sqrt{\frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2}$$

- Then, subtract mean and divide by standard deviation:

$$x_j^{(i)} \leftarrow (x_j^{(i)} - \mu_j) / \sigma_j$$

Feature Design

It is possible to fit *nonlinear* functions using linear regression:

$$(x_1, x_2, x_3) \mapsto (x_1, x_2, x_3, x_1^2, \log(x_2), x_1 + x_3)$$

Approaches

- ▶ Try standard transformations
- ▶ Design features you think will work

Polynomial Regression

