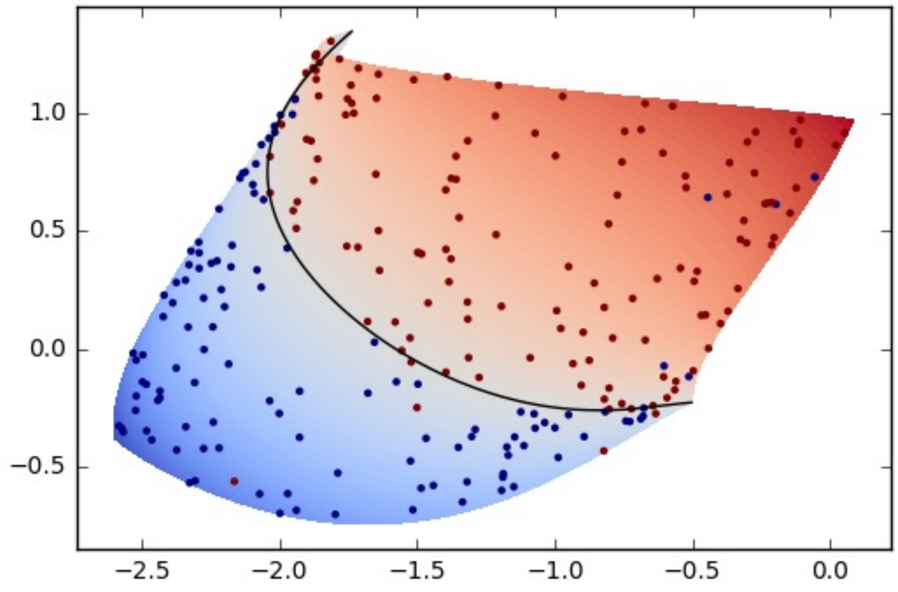
CS178 Lecture 3: Linear Regression



Gavin Kerrigan

Spring 2023

Some materials courtesy Padhraic Smyth, Alex Ihler.

Announcements/Reminders

- HW1 due next Friday (4/14)
  - After today's lecture: you should be able to do Problems 1 & 2

- Problem 1: Numpy and data visualization
- Problem 2: Linear regression via gradient descent

Today's Lecture

Linear Regression

Learning Linear Models with Gradient Descent

Today’s Lecture

Linear Regression

Learning Linear Models with Gradient Descent

Regression

Feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ (real-valued)

Target variable y (real-valued)

Training Data: set of pairs $(\mathbf{x}_i, y_i)$, with $i = 1, \ldots, n$
where x_{ij} is the value of feature j for datapoint i

Examples of Applications of Regression

Application Area	Target y	Features x
Finance	Stock market value tomorrow	Stock market + economic data from today + earlier
Health	Time until cancer recurrence	Medical tests, physiological information about patient, etc
Real Estate	Selling price of a house	Size/#bedrooms/etc, neighborhood characteristics, etc
Ecommerce	Future spending by a customer	Income, browsing habits, past spending, etc

Regression

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

Caps and bold font for a matrix

n rows: each one is a feature vector for a different individual or object

d columns = d features

We also have a vector of real-valued **targets** y:

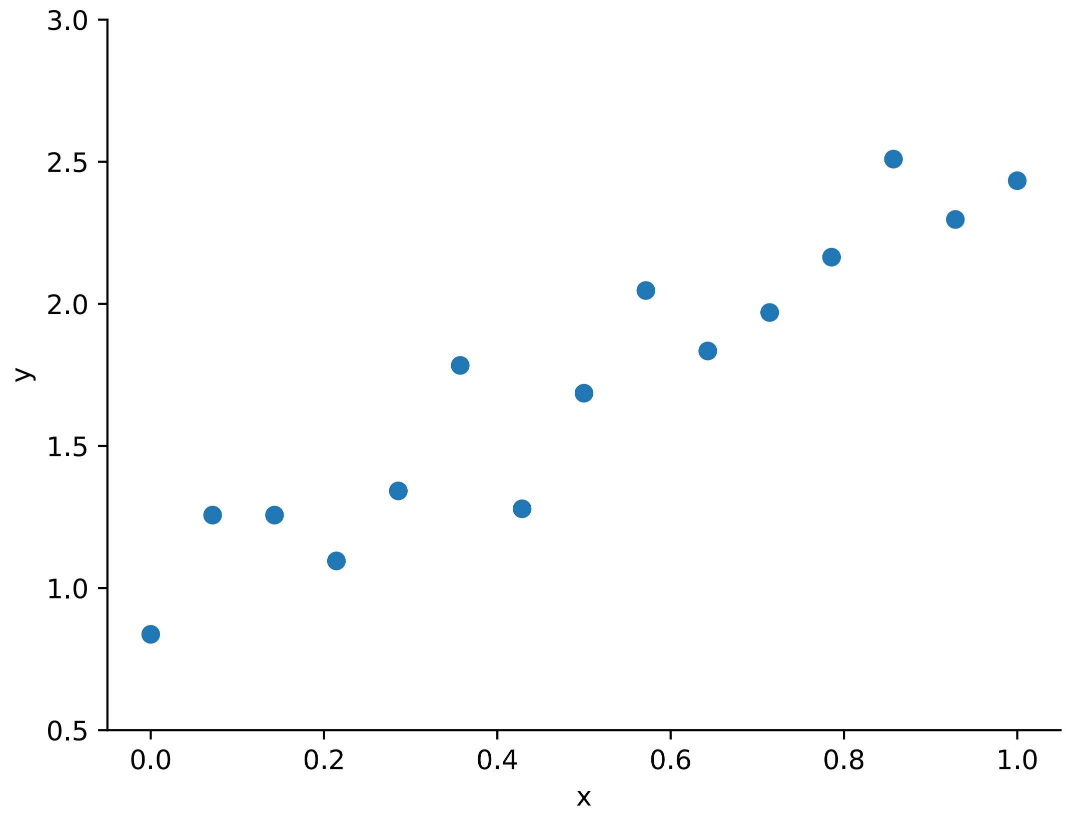
$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Bold font for a vector

n rows: one target for each datapoint

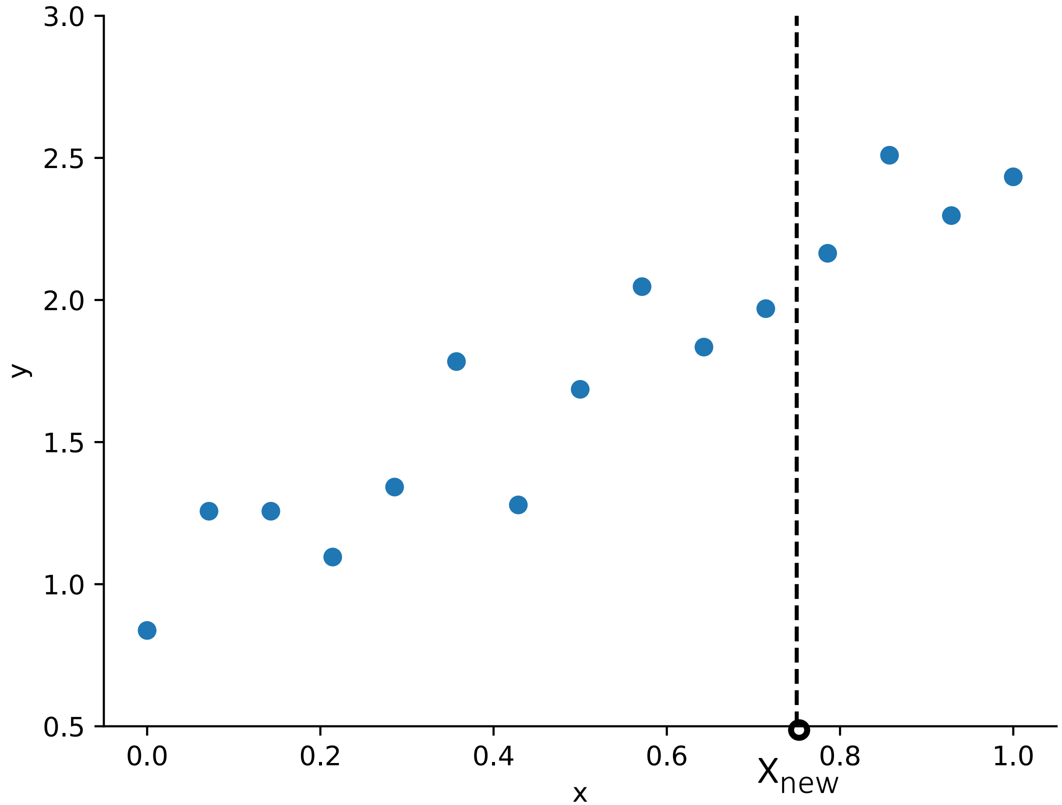
Regression in 1-dimension

Blue points indicate training data, set of (x_i,y_i) pairs



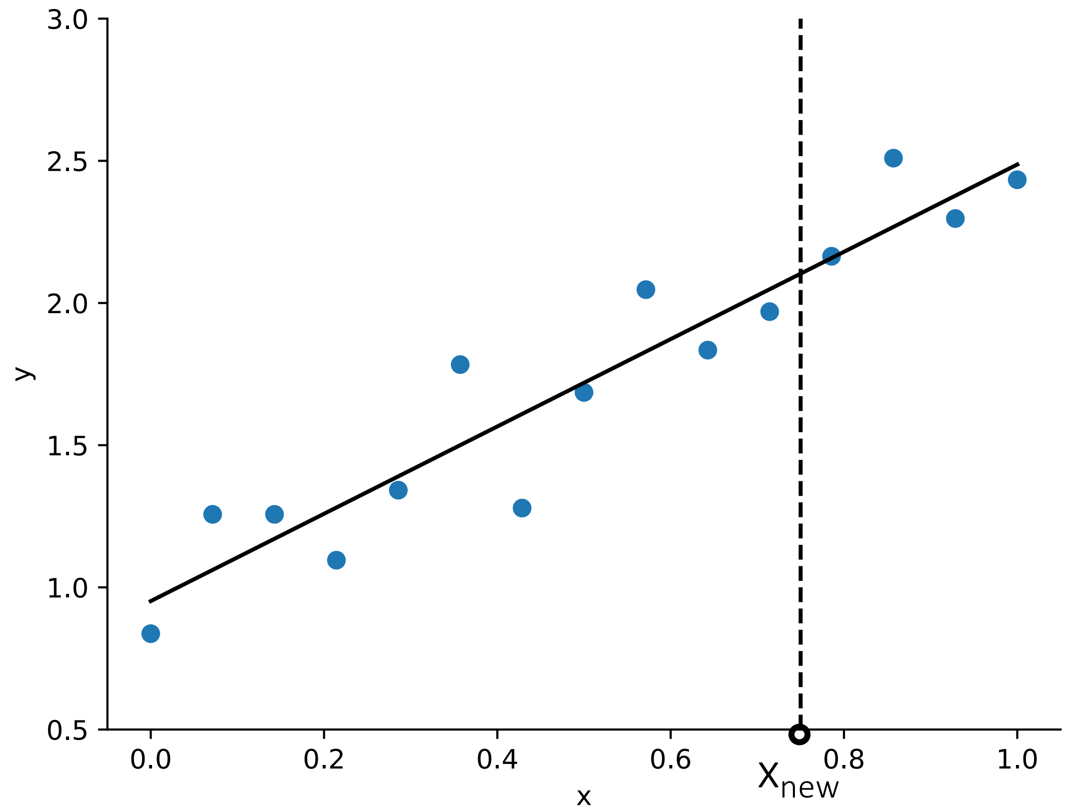
Regression in 1-dimension

How can we make predictions for inputs (x_{new}) that we didn't see in our data?



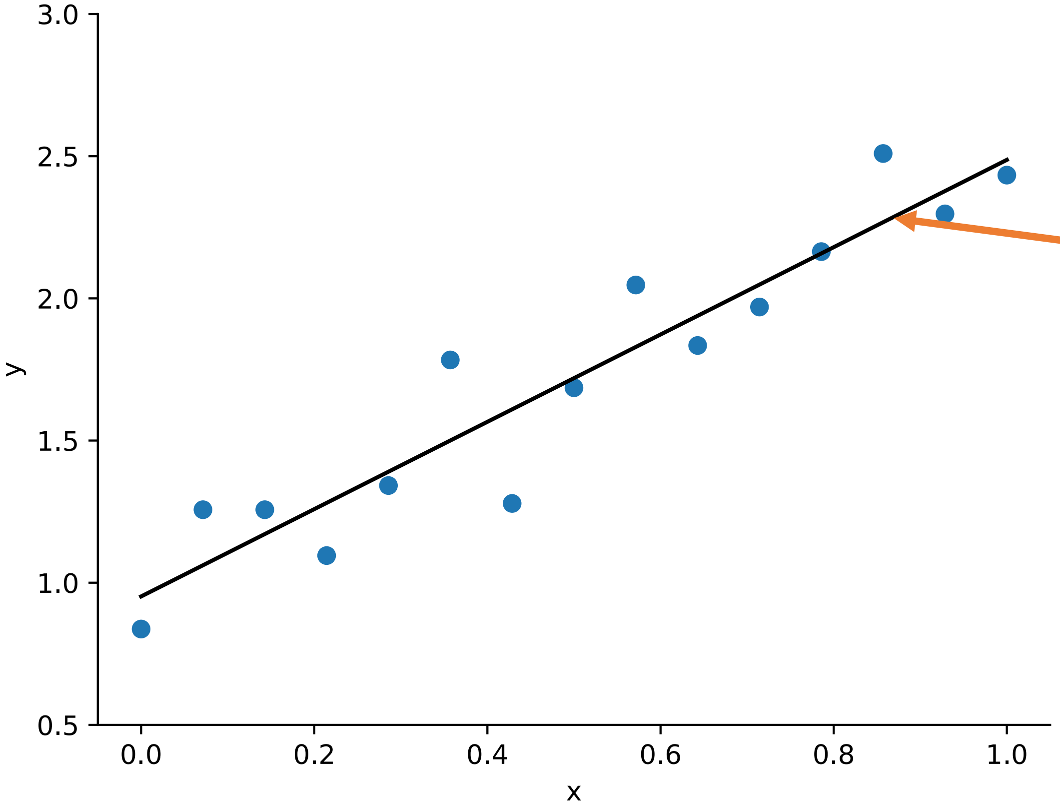
Regression in 1-dimension

In **linear regression**, we want to find the "line of best fit"



Regression in 1-dimension

In **linear regression**, we want to find the "line of best fit"



Equation of a line:

$$f(x \mid \theta) = \theta_0 + \theta_1 x$$

"bias"

The **parameters** of the model:

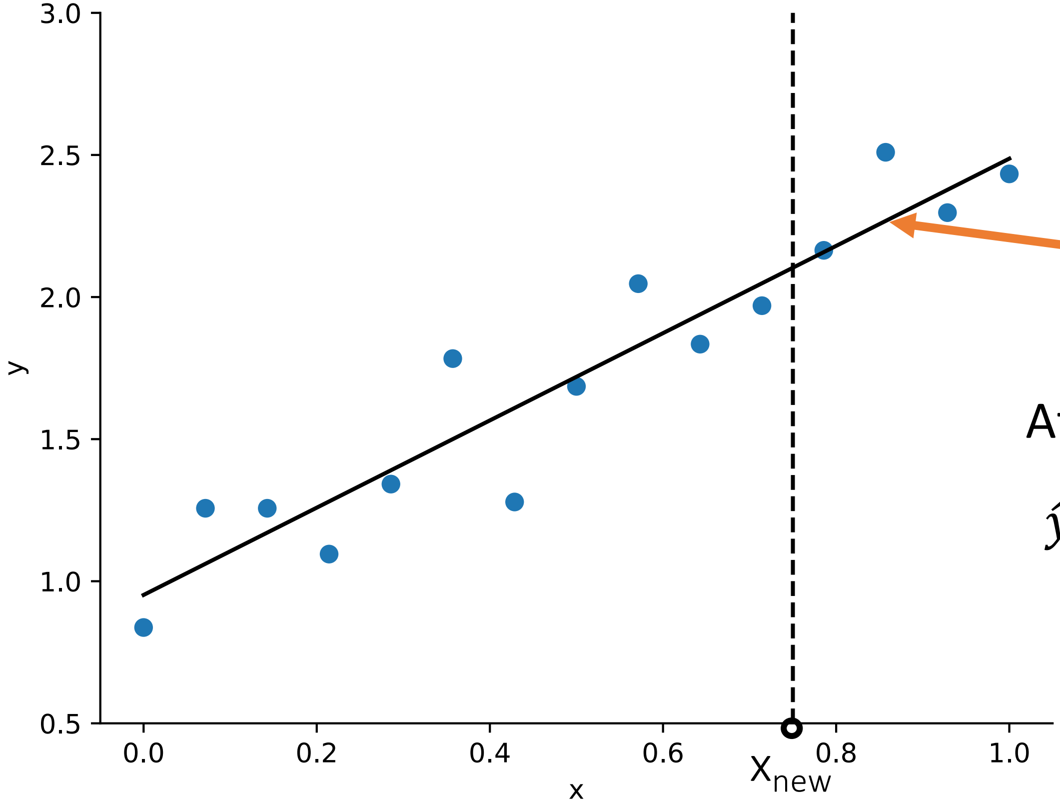
$$\theta = (\theta_0, \theta_1)^T$$

θ is unknown

- We will *learn* it from the data

Regression in 1-dimension

In **linear regression**, we want to find the "line of best fit"



Equation of a line:

$$f(x \mid \theta) = \theta_0 + \theta_1 x$$

After learning θ, you can *predict*

$$\hat{y} = f(x_{new} \mid \theta) = \theta_0 + \theta_1 x_{\text{new}}$$

Example of Linear Regression in 2 Dimensions

Linear Model = equation of a plane = f(**x** ; θ) = $\theta_0 + \theta_1 x_1 + \theta_2 x_2$

The **parameters** of the model:

$$\theta = (\theta_0, \theta_1, \theta_2)^T$$

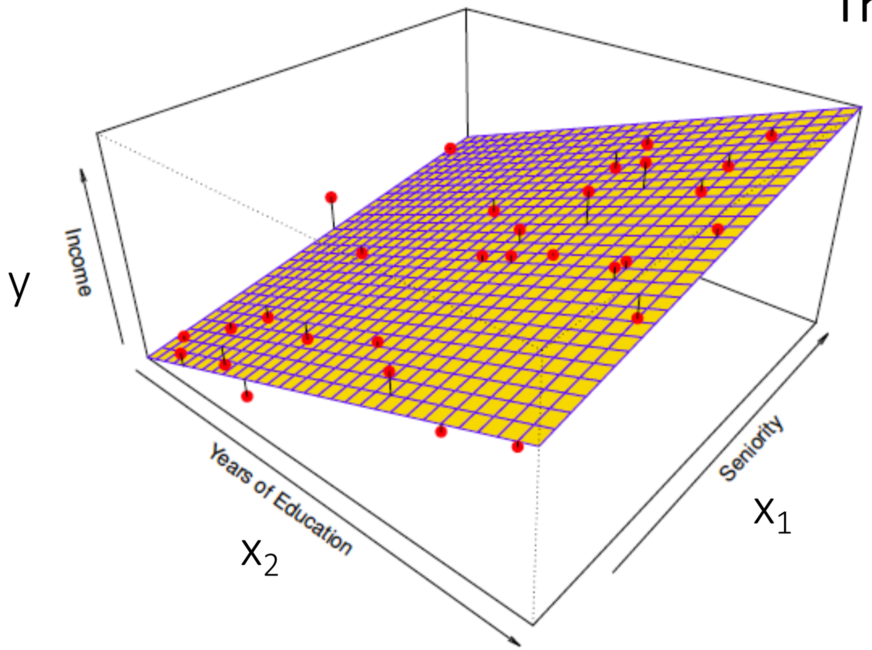
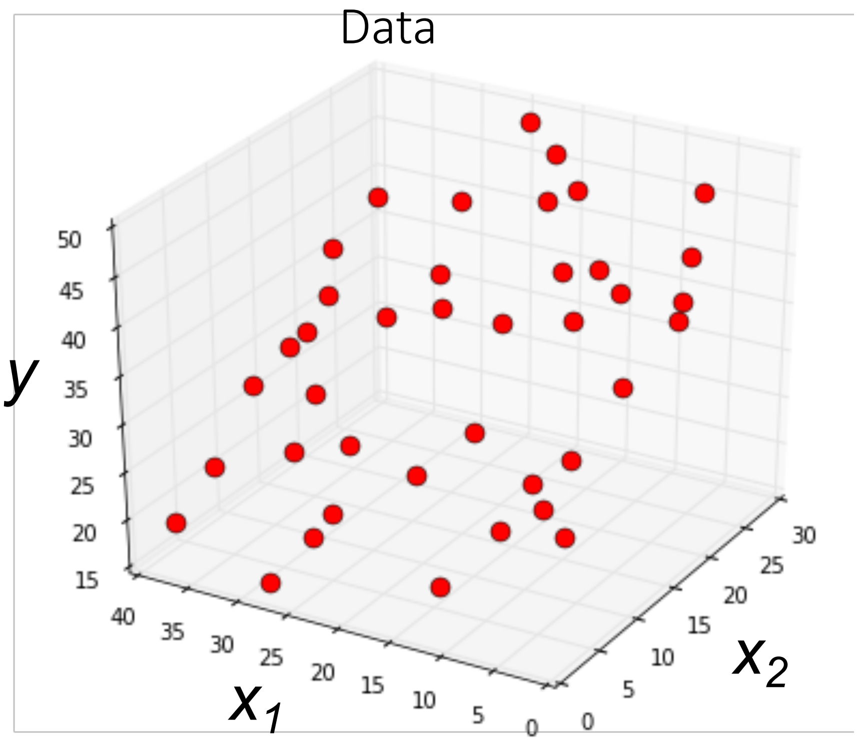


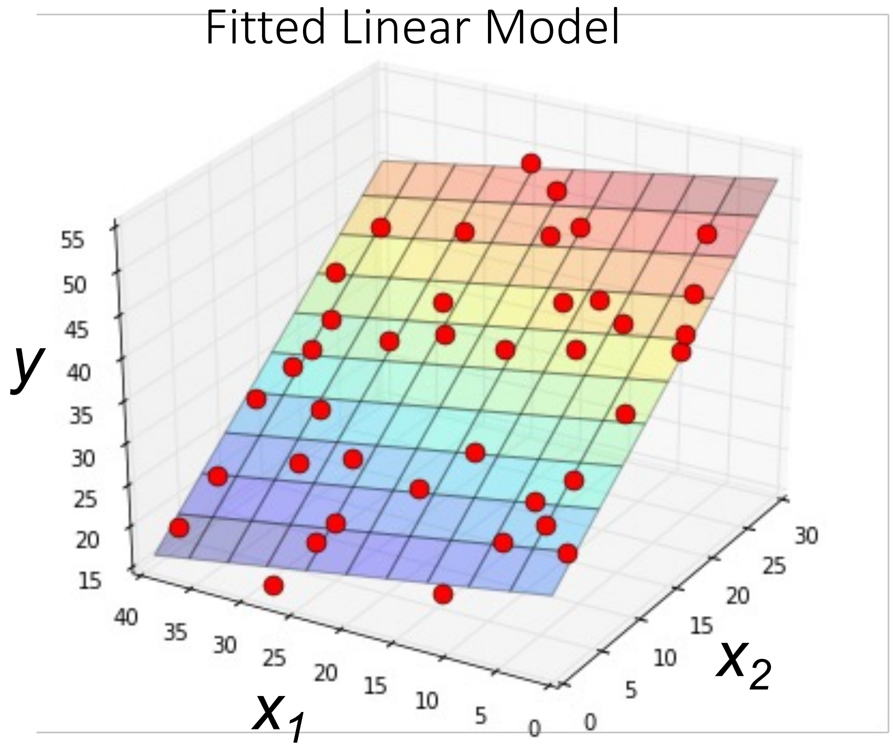
FIGURE 2.4. *A linear model fit by least squares to the* Income *data from Figure 2.3. The observations are shown in red, and the yellow plane indicates the least squares fit to the data.*

Figure from James et al, Introduction to Statistical Learning, Chapter 2

Example of Linear Regression in 2 Dimensions

Linear Model = equation of a plane = $f(\mathbf{x}\,;\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$





Linear Regression in Higher Dimensions

Feature vector $\boldsymbol{x} = (1, x_1, x_2, \dots, x_d)$ with d features

Add a "feature" $x_0 = 1$ (constant) to account for bias term θ_0

Linear regression model:

$$f(\boldsymbol{x} \mid \theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d$$

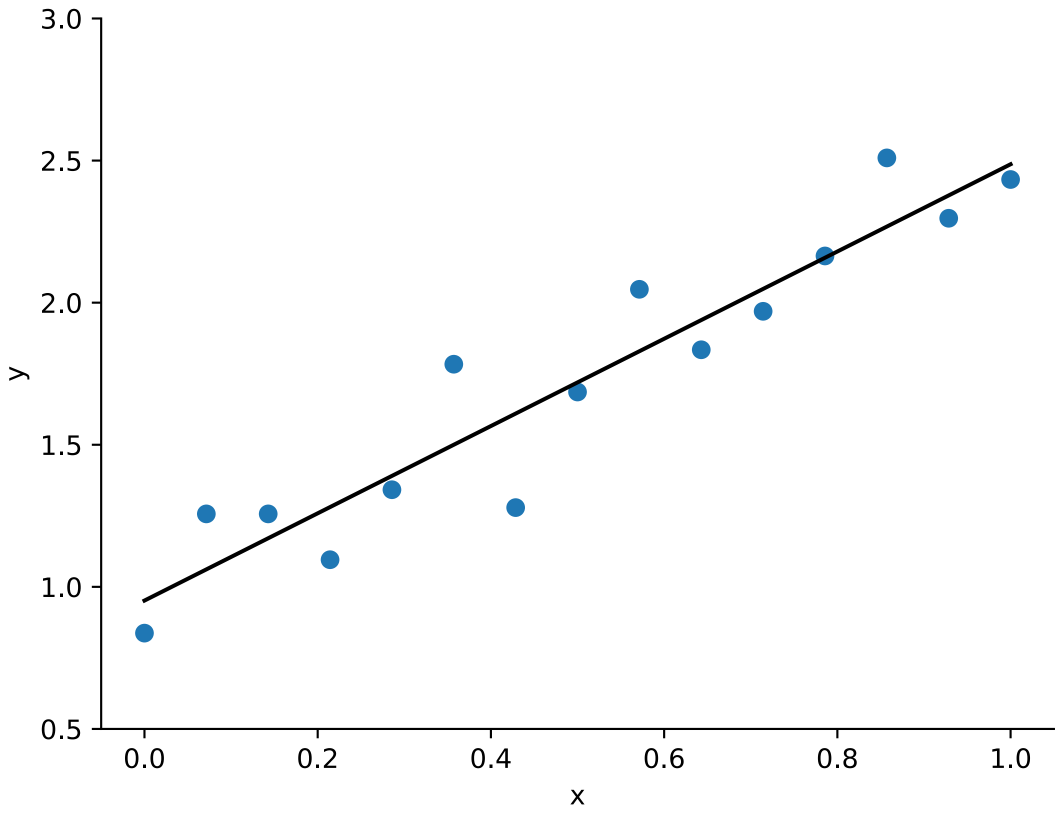
$$= \sum_{k=0}^{d} \theta_k x_k = \theta^T \boldsymbol{x}$$

$\theta_0 x_0$

Questions?

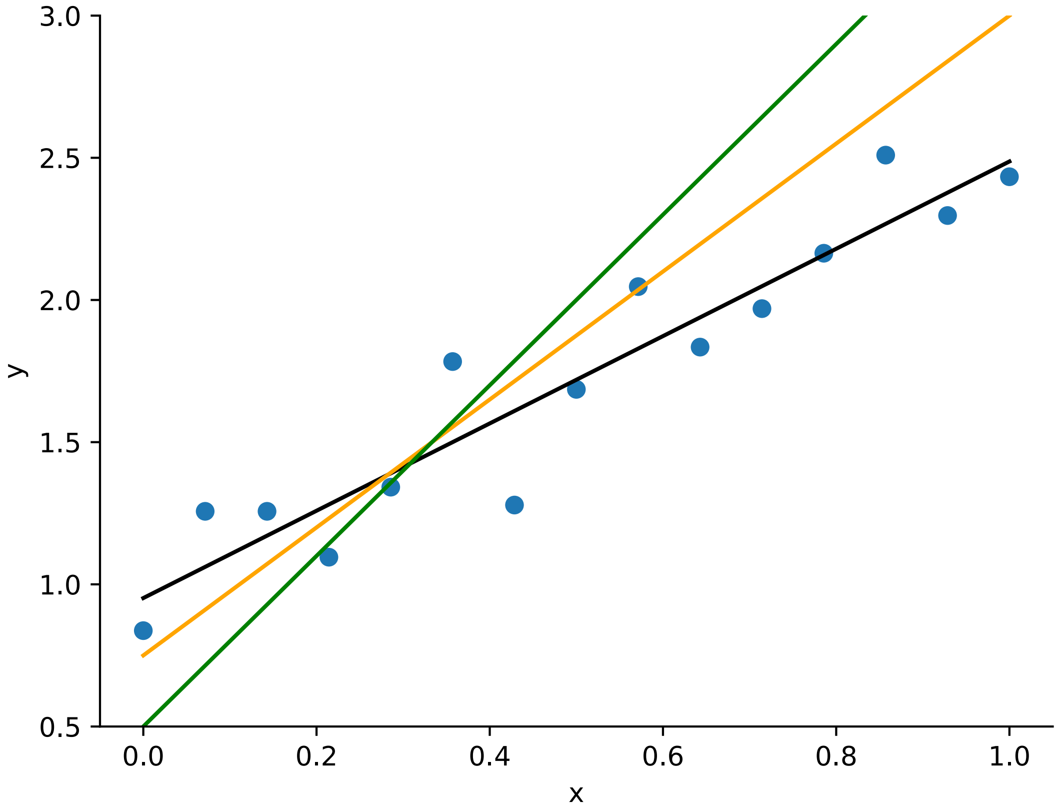
Learning a Linear Model

How can we determine the line of best fit?



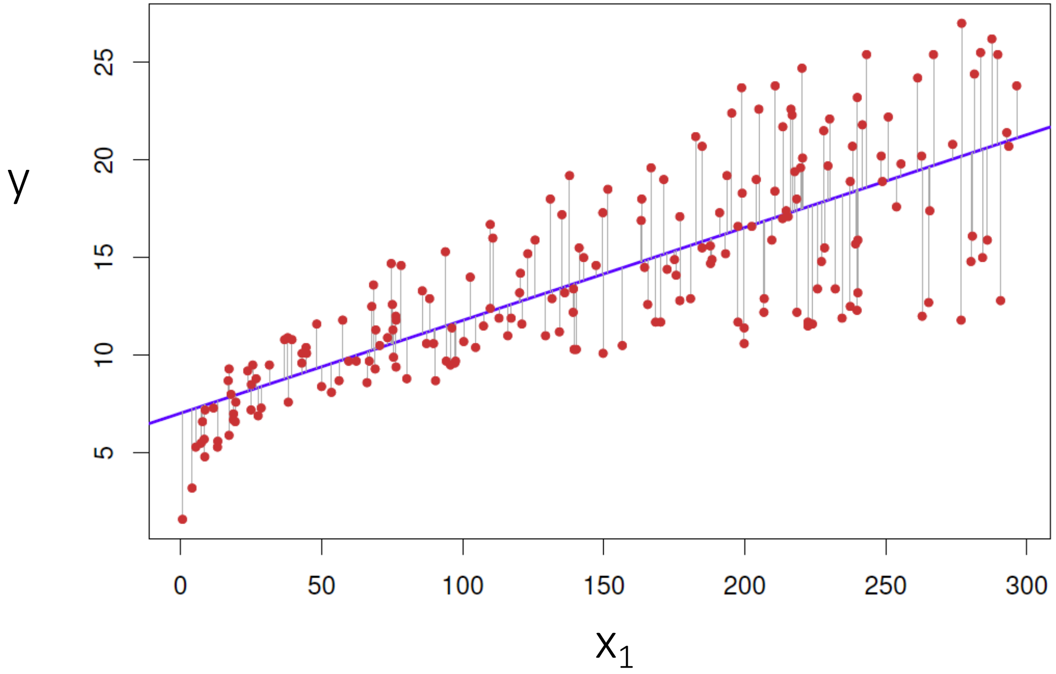
Learning a Linear Model

How can we determine the line of best fit?



Learning a Linear Model

<u>Key idea</u>: Find the model that *minimizes the prediction error* on our data



Red dots are (x_i, y_i) pairs

Blue line is the fitted model
$= f(x ; \theta) = \theta_0 + \theta_1 x_1$

Vertical black lines illustrate error between true y values (red) and predicted values (blue)

Figure from James et al, Introduction to Statistical Learning, Chapter 3

Loss Function: Mean Squared Error

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x}_i|\theta))^2$$

The loss function we want to minimize

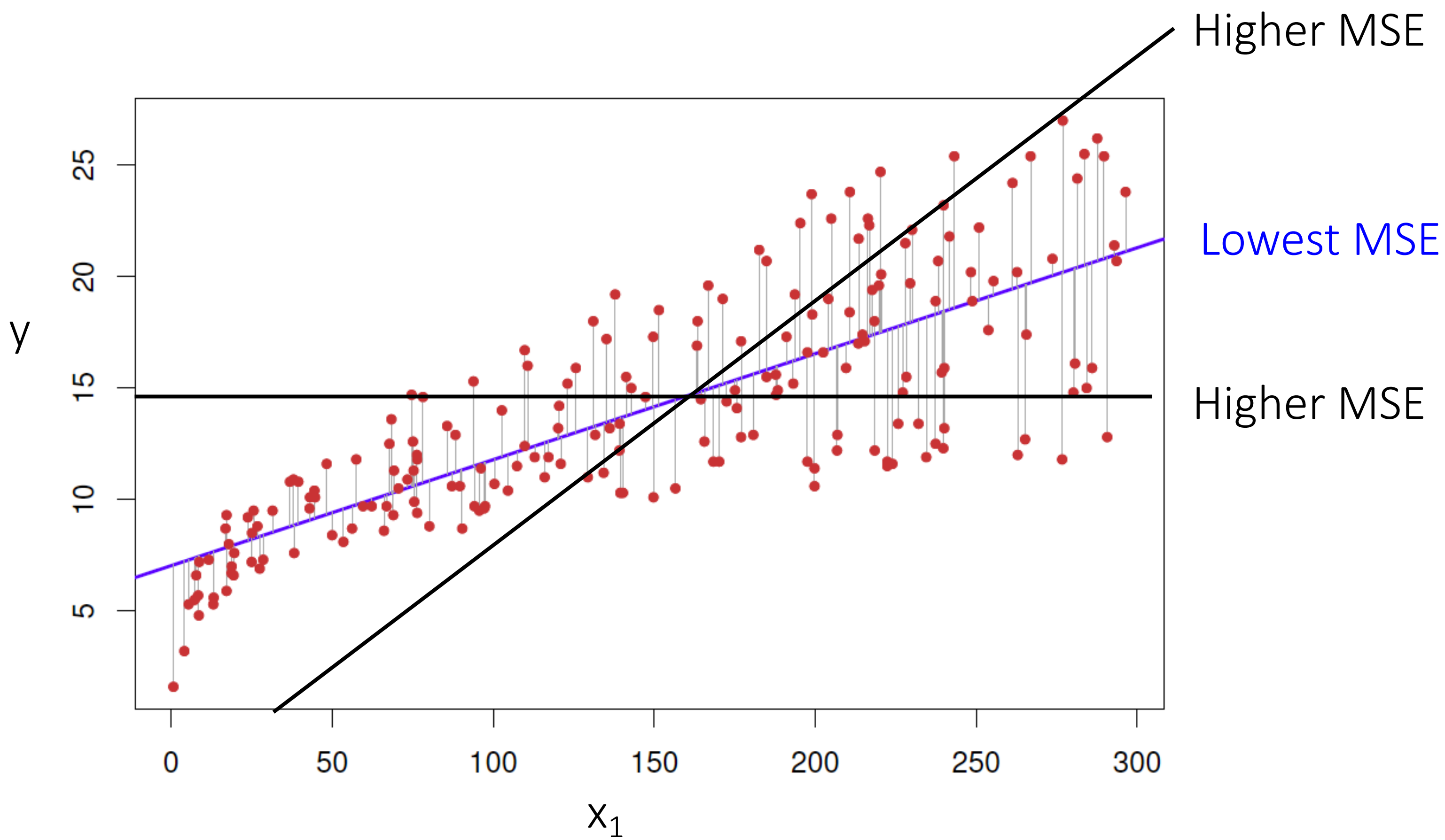
Average over all training data points

True target for ith datapoint

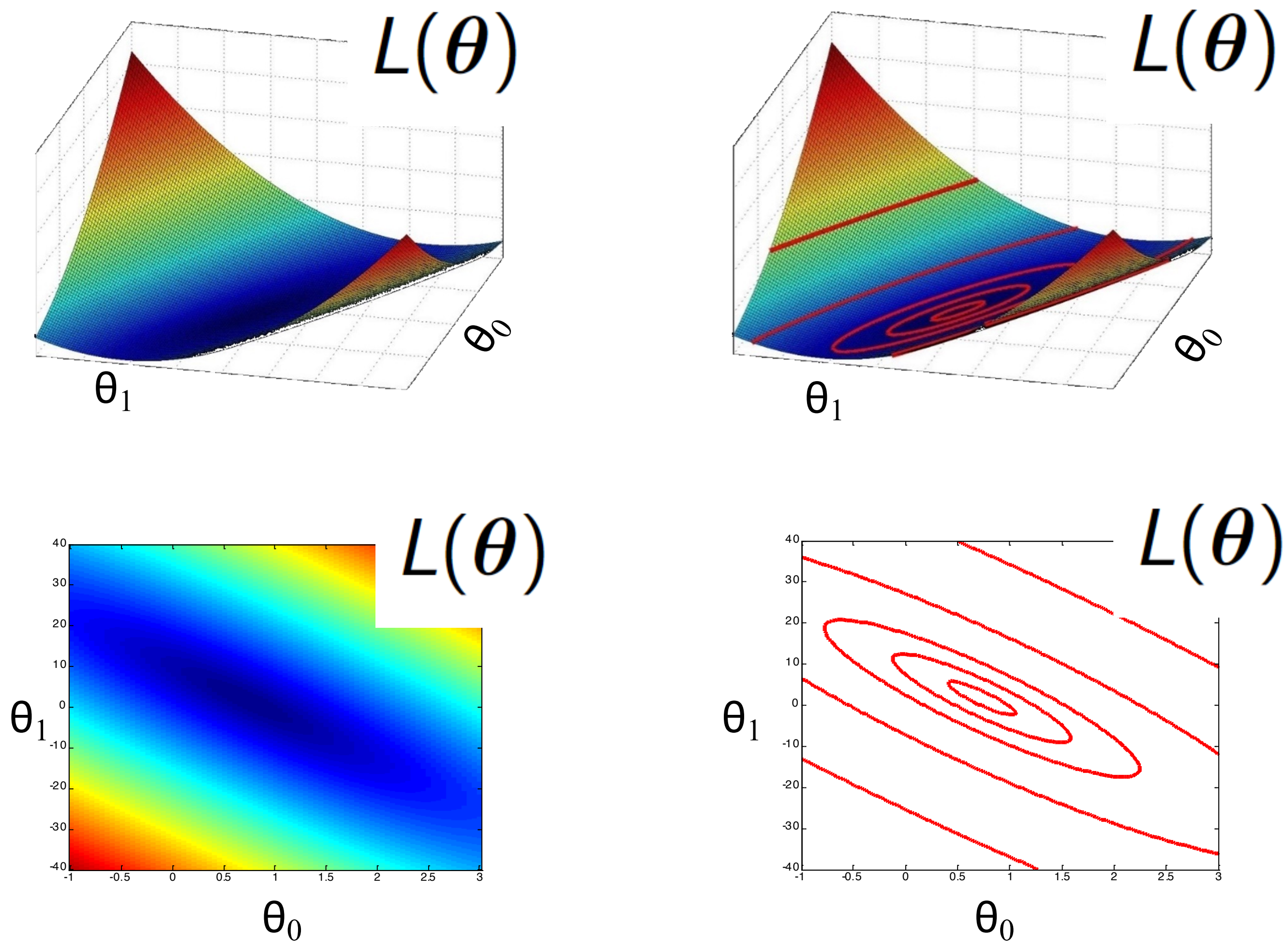
Model's prediction for ith datapoint

Idea: try to find the the parameters θ that minimize this loss, i.e., that make the predictions close to the true targets

MSE reflects the Goodness of Fit



Visualizing the MSE Loss Function



Today’s Lecture

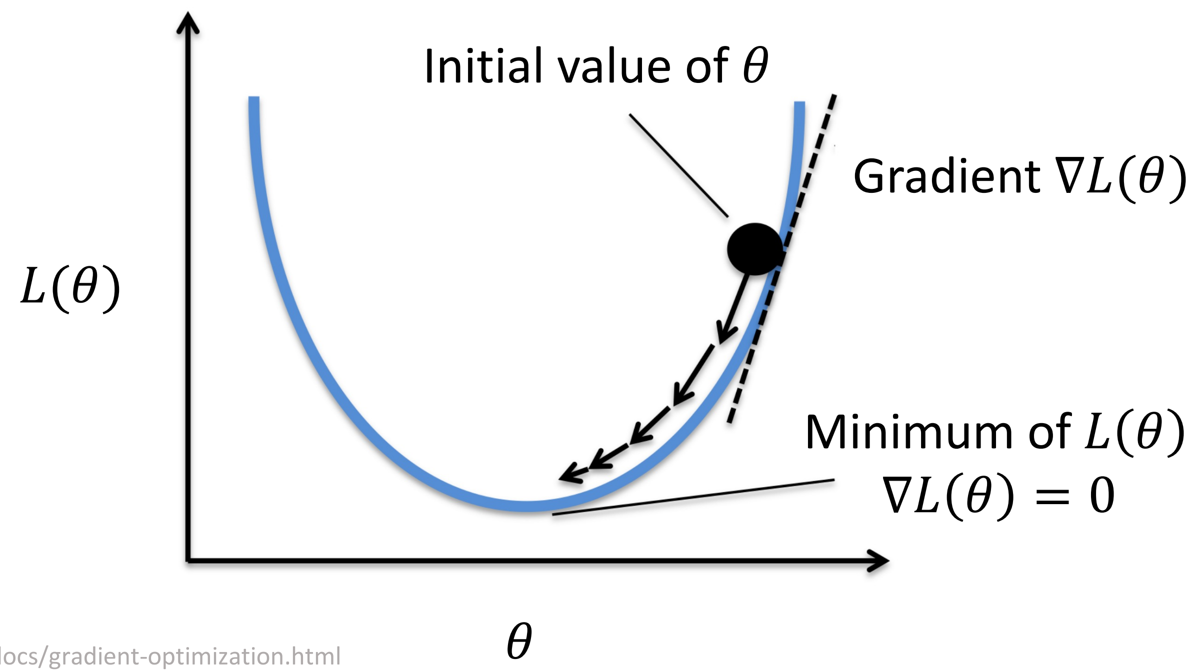
Linear Regression Models

Learning Linear Models with Gradient Descent

Gradient Descent Intuition

How can we find the value of $\theta = (\theta_0, \theta_1, \dots, \theta_d)^T$ that minimize the loss $L(\theta)$?

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i|\theta))^2$$



https://sebastianraschka.com/faq/docs/gradient-optimization.html

Minimizing a Function in Multiple Dimensions

In machine learning will want to do this type of "downhill move" in many dimensions (i.e., with many parameters), not just one dimension

In general we don't know much about the "shape" of the loss function

(Imagine being on a mountain in fog and wanting to take steps to the bottom)

Locally we can compute the "local downhill direction" and go that direction (in a multi-dimensional space)

The gradient is the multi-dimensional version of the one-dimensional derivative

And gradient descent is a heuristic local search algorithm that uses the gradient – widely used in machine learning

What is a Gradient?

The gradient of a function of multiple variables is a **vector of partial derivatives**, one partial derivative for each variable

The loss function is a function of multiple parameters, e.g.,

$$L(\boldsymbol{\theta}) = L(\theta_0, \theta_1, \ldots, \theta_d)$$

The gradient vector for the loss function is a vector of partial derivatives, one partial derivative per parameter

$$\nabla L(\boldsymbol{\theta}) = \left(\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_0}, \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_d} \right)$$

Property of the Gradient:

The gradient vector of $L(\boldsymbol{\theta})$ points in the steepest uphill direction of the $L(\boldsymbol{\theta})$ surface at point $\boldsymbol{\theta}$.

So the negative of the gradient, $-\nabla L(\boldsymbol{\theta})$, points in the steepest downhill direction

We can use this to "move" locally downhill in $\boldsymbol{\theta}$ space

i.e., if we are at a current parameter vector $\boldsymbol{\theta}$, we compute the gradient $\nabla L(\boldsymbol{\theta})$, and move in the negative (opposite) of this direction in $\boldsymbol{\theta}$ space.

At a local minimum of $L(\theta)$, the gradient is zero: $\nabla L(\theta) = 0$

Update Equation using Gradient

$$\theta^{new} = \theta^{old} - \lambda \cdot \nabla L(\theta^{old})$$

new parameter vector

old parameter vector

Learning rate (also known as stepsize)

gradient vector: steepest downhill direction at θ^{old}

Theory tells us that we will converge to at least a local minimum if λ is small

General Gradient Descent Algorithm

1. Initialize the parameter vector θ with random values

2. Compute the gradient at the current parameter vector

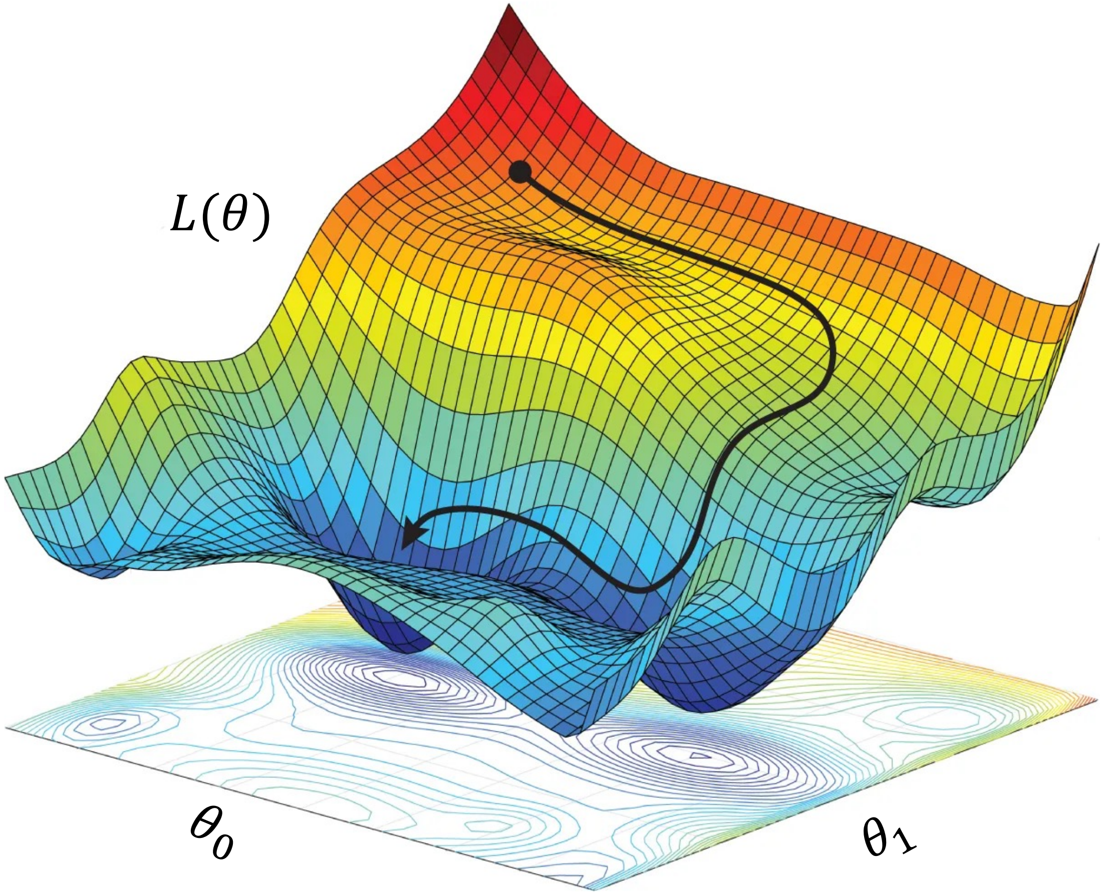
3. Check for convergence (e.g., if magnitude of gradient < epsilon)

4. If not converged:
    (a) update the parameters using the gradient (equation on last slide)
    (b) return to step 2

This is the basic iteration loop: keep moving downhill in parameter space

5. If converged, return the current parameter vector

General Gradient Descent Algorithm



Amini et al. 2019, Spatial Uncertainty Sampling for End to End Control

Notation for MSE and Linear Model

Lets generalize notation a little with $x_0 = 1$ and define

$$\mathbf{x} = (x_0, x_1, \ldots, x_d)$$

With this we can define

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sum_{j=0}^{d} \theta_j x_j$$

We can define the error for predicting each training datapoint as

$$e_i \;=\; y_i - f(\mathbf{x}_i; \boldsymbol{\theta}) \;=\; y_i - \sum_{j=0}^{d} \theta_j x_{ij}$$

The value of the jth feature for the ith feature vector $\mathbf{x}_i$

and rewrite the loss function as

$$L(\boldsymbol{\theta}) \;=\; \frac{1}{n} \sum_{i=1}^{n} (e_i)^2$$

Defining the Gradient for MSE Loss + Linear Model

MSE Loss Function:

$$L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(e_i)^2$$

The Gradient Vector:

$$\nabla L(\boldsymbol{\theta}) = \left(\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_0}, \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_d}\right)$$

We need to define each element (each partial derivative) of this vector:

$$\frac{\partial}{\partial \theta_k} L(\theta) \quad \text{for } k = 0, 1, \ldots, d$$

Defining the Gradient for MSE Loss + Linear Model

For each parameter θ_k we need to find the partial derivative:

$$
\begin{aligned}
\frac{\partial}{\partial \theta_k} L(\theta) = \quad & \frac{\partial}{\partial \theta_k}\left(\frac{1}{n}\sum_{i=1}^{n}(e_i)^2\right) \\
= \quad & \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \theta_k}(e_i)^2 \\
= \quad & \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial e_i}(e_i)^2 \quad \frac{\partial}{\partial \theta_k}(e_i) \qquad \text{(chain rule)} \\
= \quad & \frac{1}{n}\sum_{i=1}^{n}2e_i\frac{\partial}{\partial \theta_k}\left(y_i - \sum_{j=0}^{d}\theta_j x_{ij}\right)
\end{aligned}
$$

Defining the Gradient for MSE Loss + Linear Model

Continuing…….

$$\frac{\partial}{\partial \theta_k} L(\theta) = \frac{1}{n}\sum_{i=1}^{n} 2e_i \frac{\partial}{\partial \theta_k}\left(y_i - \sum_{j=0}^{d} \theta_j x_{ij}\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n} 2e_i(-x_{ik})$$

$$= -\frac{2}{n}\sum_{i=1}^{n} e_i x_{ik}$$

$$= -\frac{2}{n}\sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{d} \theta_j x_{ij}\right) x_{ik}$$

The Gradient Vector:

$$\nabla L(\boldsymbol{\theta}) = \left(\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_0}, \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_d}\right)$$

General Gradient Update Equation:

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta}^{old} - \lambda \cdot \nabla L(\boldsymbol{\theta})$$

Specific case of MSE Loss + Linear Model:
For each parameter:

$$\theta_k^{\text{new}} = \theta_k^{\text{old}} + \lambda \frac{2}{n} \sum_{i=1}^{n} e_i x_{ik} \qquad k = 0, 1, \ldots, d$$

This is simple to compute for each parameter at each iteration

MSE loss with a linear model is a convex problem: one global minimum!

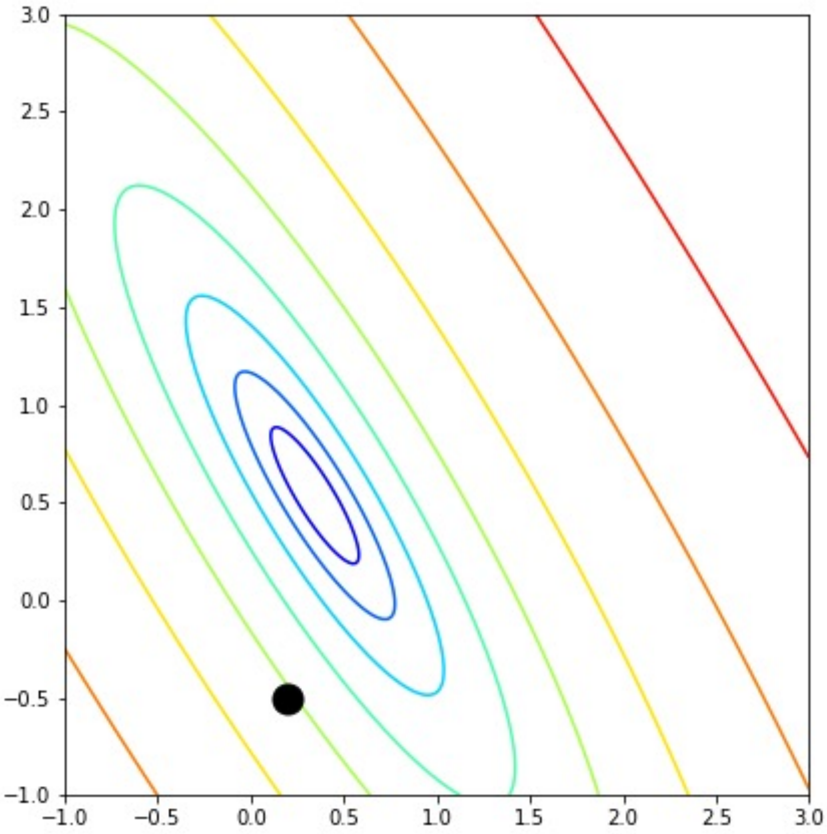
Example of Gradient Descent, MSE Loss + Linear Model

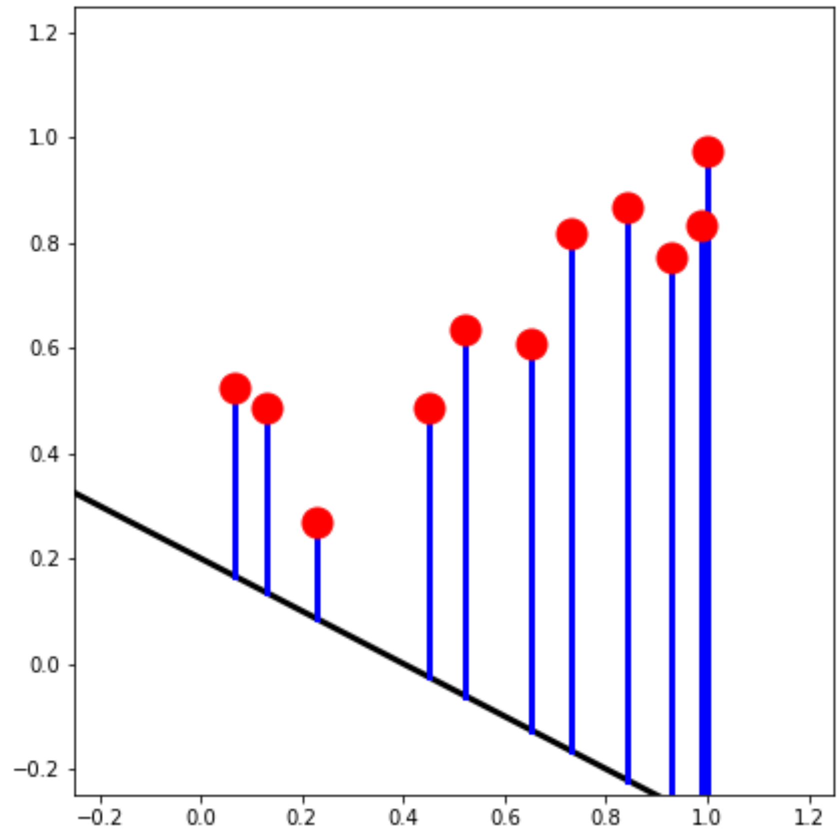
Red dots = true y values

Black line = current model

Blue lines = prediction errors e_i

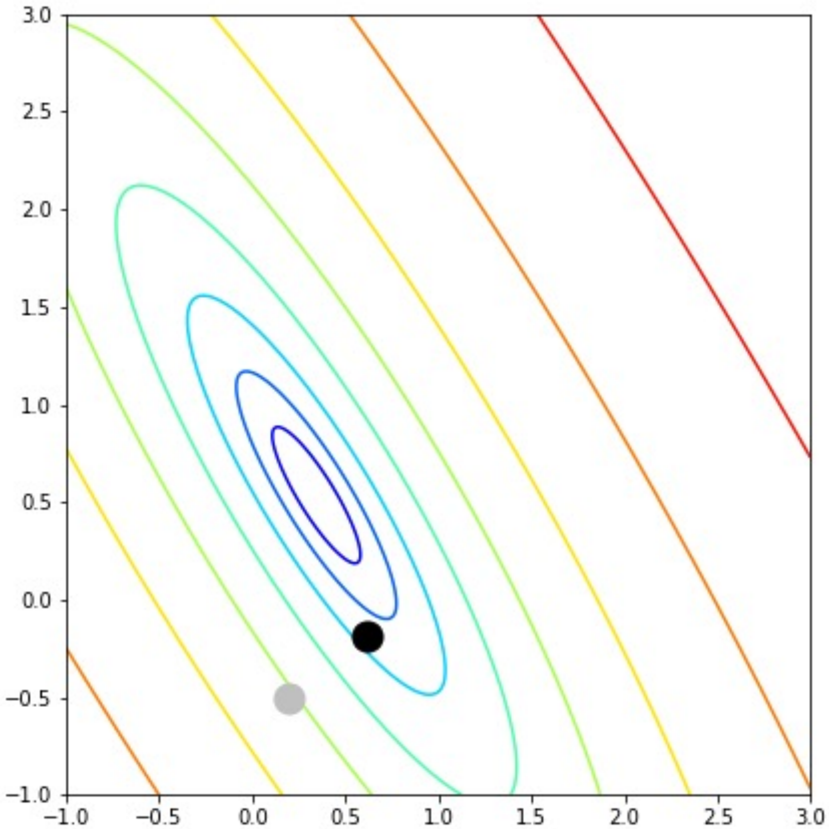
Initialization

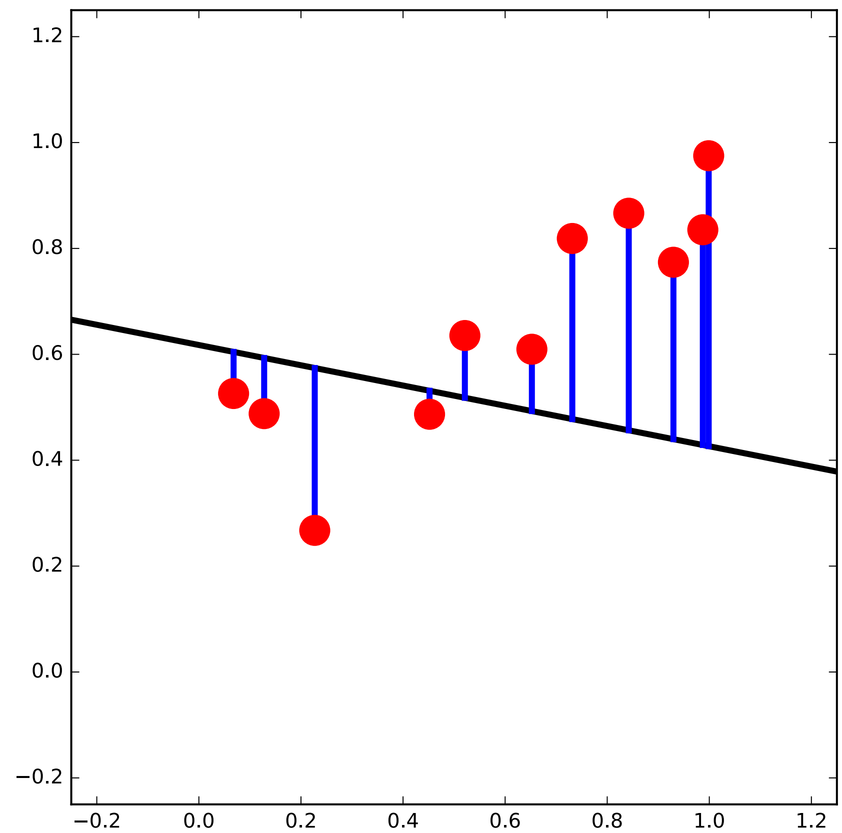




Example of Gradient Descent, MSE Loss + Linear Model

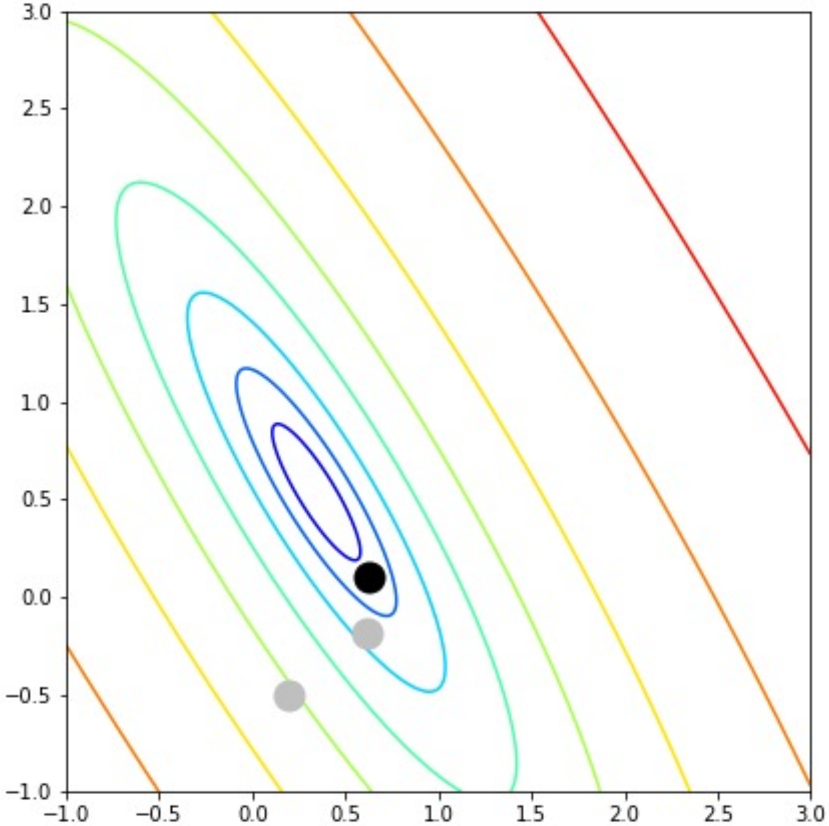
Iteration 1

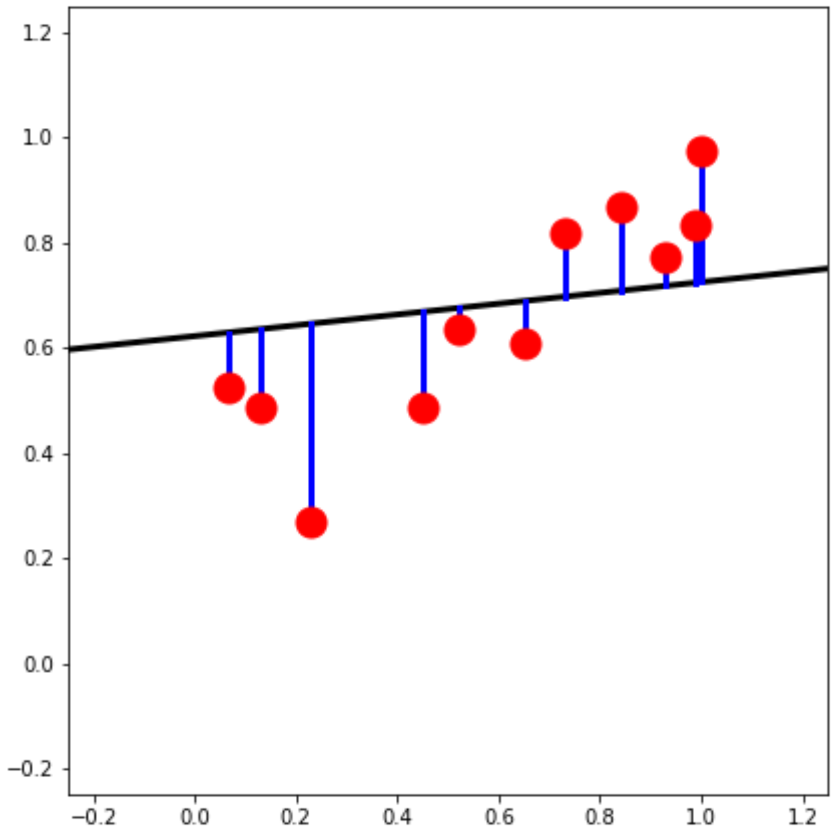




Example of Gradient Descent, MSE Loss + Linear Model

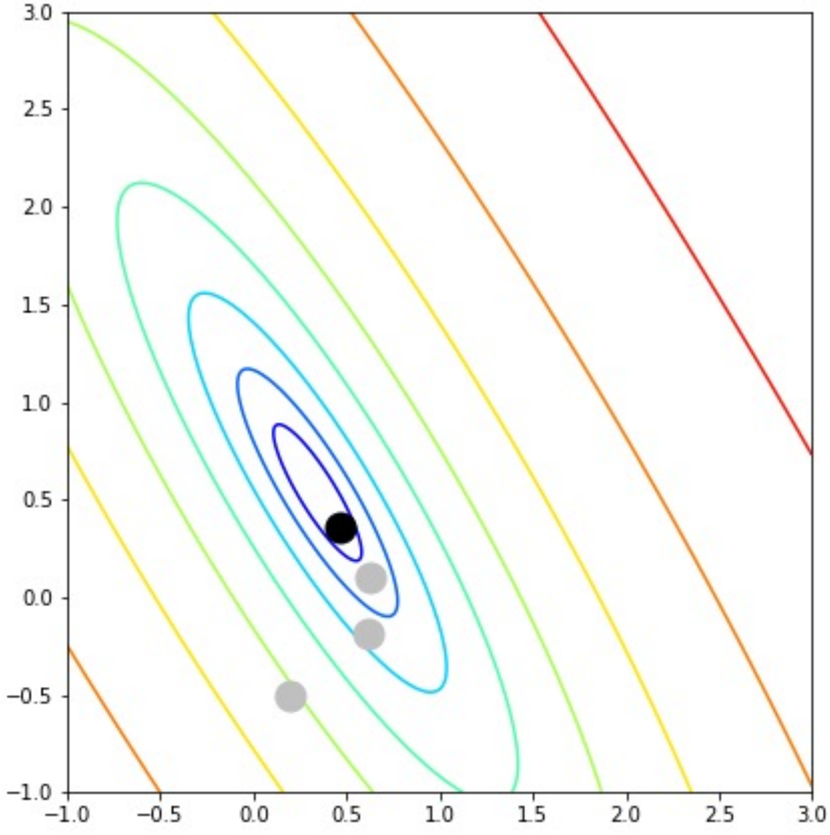
Iteration 10

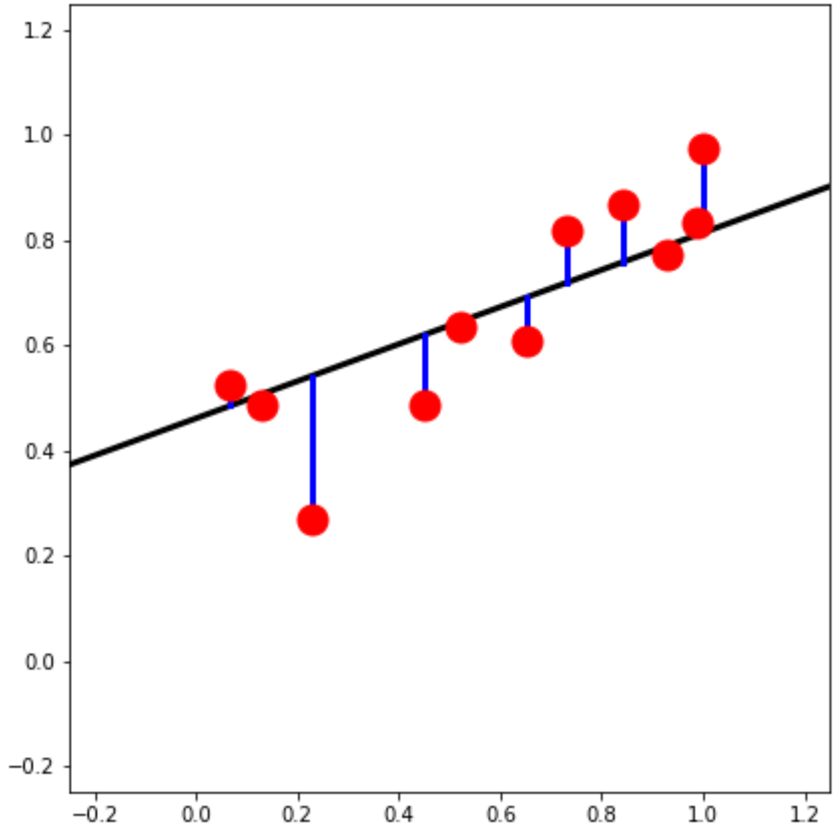




Example of Gradient Descent, MSE Loss + Linear Model

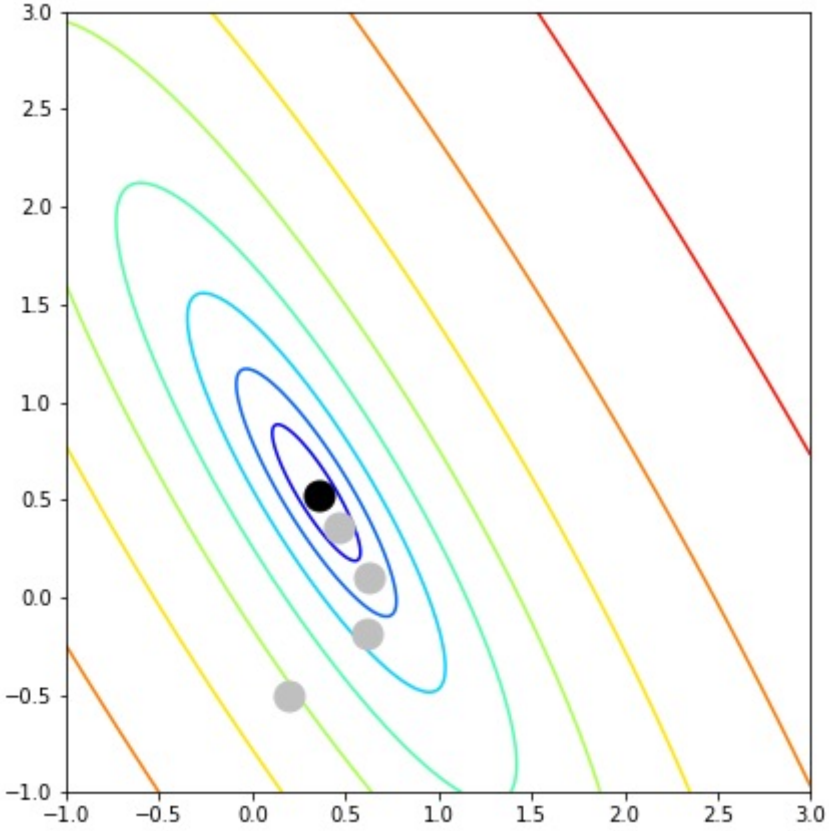
Iteration 30

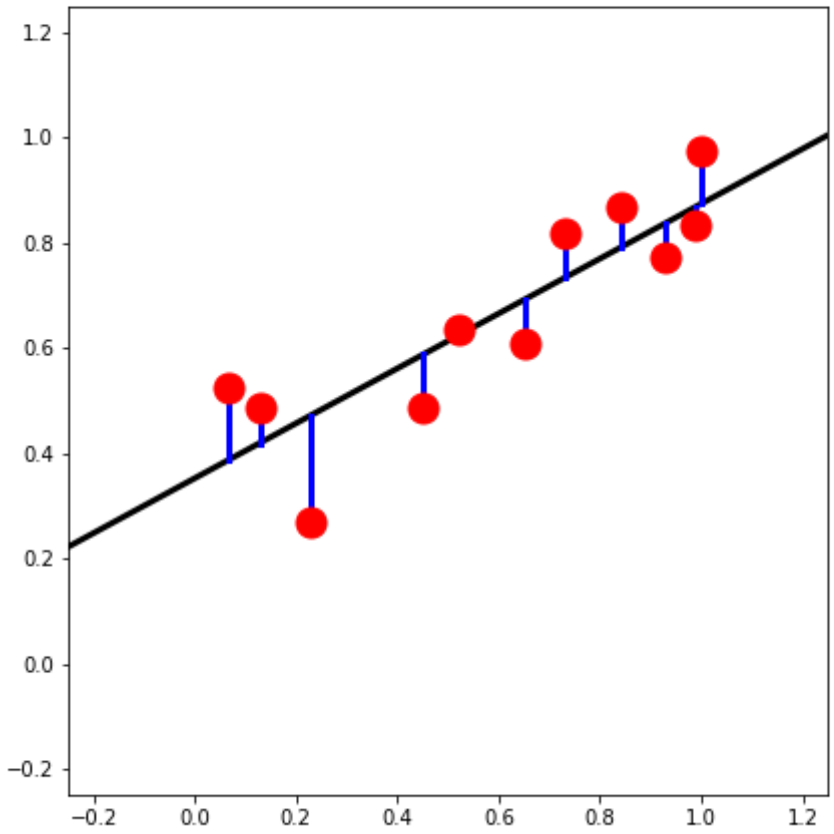




Example of Gradient Descent, MSE Loss + Linear Model

Iteration 90





Questions?

Alternative to Gradient: Analytical Solution

Using some matrix algebra, we could write the solution directly as

$$\theta = \mathbf{y}^T \cdot \mathbf{X} \cdot \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1}$$

This is the vector d+1 values of θ that minimize the MSE loss

where:

$\mathbf{y}$ is a $n \times 1$ column vector of the training targets $y_1, \ldots, y_n$

$\mathbf{X}$ is a $n \times (d+1)$ matrix of training features with $\mathbf{x}_i$ as rows

$\mathbf{X}^T \cdot \mathbf{X}$ is a $(d+1) \times (d+1)$ matrix

and where T indicates transpose and -1 indicates matrix inverse

Alternative to Gradient: Analytical Solution

Using some matrix algebra, we could write the solution directly as

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

This equation is the standard method used in statistics for finding the coefficients for linear models with MSE loss (known as "ordinary least squares" or OLS)

However, if d is large (high-dimensional) then computing the inverse $(\mathbf{X}^T \cdot \mathbf{X})^{-1}$ can be numerically unstable.

Also: time complexity is $O(nd^2 + d^3)$ versus $O(nd)$ for gradient method

So, in machine learning, gradient methods are generally preferred (since d is often large)

Example: Diabetes Dataset

7.1.3. Diabetes dataset

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

Data Set Characteristics:

Number of Instances::	442
Number of Attributes::	First 10 columns are numeric predictive values
Target::	Column 11 is a quantitative measure of disease progression one year afte
Attribute Information::	• age age in years • sex • bmi body mass index • bp average blood pressure • s1 tc, total serum cholesterol • s2 ldl, low-density lipoproteins • s3 hdl, high-density lipoproteins • s4 tch, total cholesterol / HDL • s5 ltg, possibly log of serum triglycerides level • s6 glu, blood sugar level

442 examples

10 features

1 real-valued target y

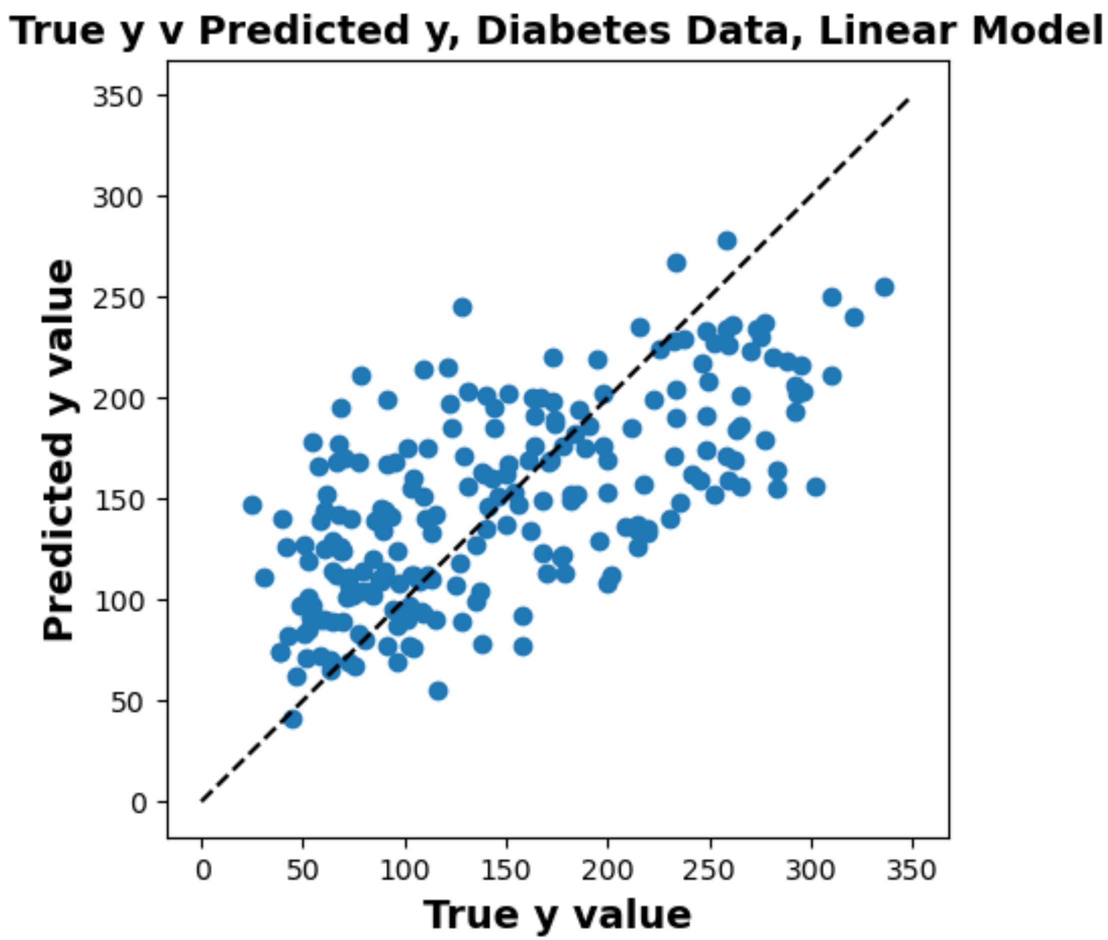
Features have been scaled to all have same standard deviation

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of `n_samples` (i.e. the sum of squares of each column totals 1).

Source URL: https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see: Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion), 407-499. (https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)

Fitting a Model to the Diabetes Data



Fitted Coefficients for Diabetes Data

From a linear model fitted on 50% of data (221 patients)

Feature Name
Age
Gender
Body Mass Index
Average Blood Pressure
Total Serum Cholesterol
Low Density Lipid Proteins
High Density Lipid Proteins
Total Cholestorol
Serum Triglycerides
Blood Sugar Level

Fitted Coefficients for Diabetes Data

From a linear model fitted on 50% of data (221 patients)

Feature Name	**Coefficient Value/100**
Age	-0.7
Gender	-2.8
Body Mass Index	6.3
Average Blood Pressure	2.2
Total Serum Cholesterol	-9.5
Low Density Lipid Proteins	5.2
High Density Lipid Proteins	2.1
Total Cholestorol	4.0
Serum Triglycerides	5.6
Blood Sugar Level	1.3

Summary and Wrapup

- **Linear regression** models are models of the form

$$f(x \mid \theta) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$$

With unknown parameter vector

$$\theta = (\theta_0, \theta_1, \ldots, \theta_d)^T$$

- To learn a linear regression model, we want to find parameters which minimize the MSE

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i|\theta))^2$$

- MSE can be minimized via **gradient descent**

Next Lecture

- More on details on gradient descent
    - How do we choose the step size?
    - Methods for improving gradient descent

- Nonlinear Regression
    - How can we model nonlinear relationships in data?

Questions?