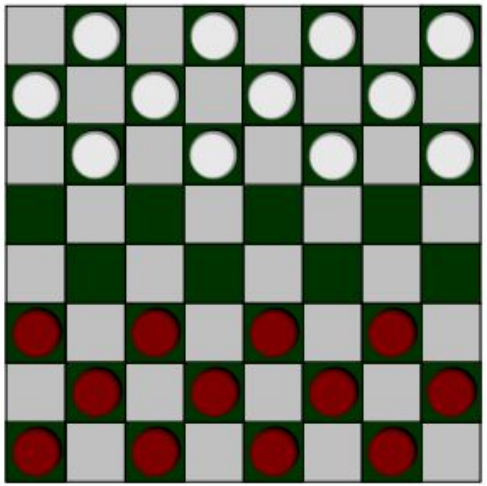
Machine Learning definition

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.



Machine Learning definition

- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
- Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T in this setting?
  - Classifying emails as spam or not spam.
  - Watching you label emails as spam or not spam.
  - The number (or fraction) of emails correctly classified as spam/not spam.
  - None of the above—this is not a machine learning problem.

Examples

- Database mining, Large datasets from growth of automation/web.
  - E.g., Web click data, medical records, biology, engineering
- Applications can't program by hand.
  - E.g., Autonomous helicopter, handwriting recognition, most of
  - Natural Language Processing (NLP), Computer Vision.

Machine learning Types

- Machine learning algorithms:
  - Supervised learning
  - Unsupervised learning
- Others: Reinforcement learning, recommender systems.

Supervised Learning

Supervised machine learning algorithms are designed to learn by example. The name "supervised" learning originates from the idea that training this type of algorithm is like having a teacher supervise the whole process.

Supervised Learning

When training a supervised learning algorithm, the training data will consist of inputs paired with the correct outputs. During training, the algorithm will search for patterns in the data that correlate with the desired outputs. After training, a supervised learning algorithm will take in new unseen inputs and will determine which label the new inputs will be classified as based on prior training data. The objective of a supervised learning model is to predict the correct label for newly presented input data. At its most basic form, a supervised learning algorithm can be written simply as:

$$Y=f(x)$$

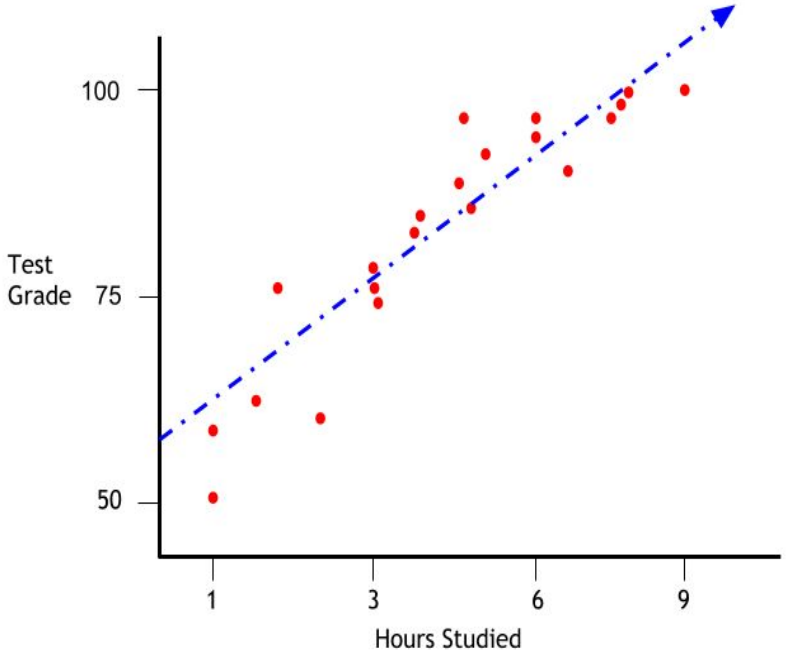
Where Y is the predicted output that is determined by a mapping function that assigns a class to an input value x. The function used to connect input features to a predicted output is created by the machine learning model during training.

Supervised learning can be split into two subcategories:

- **Regression**
    - Linear Regression
    - Logistic Regression
    - Polynomial Regression

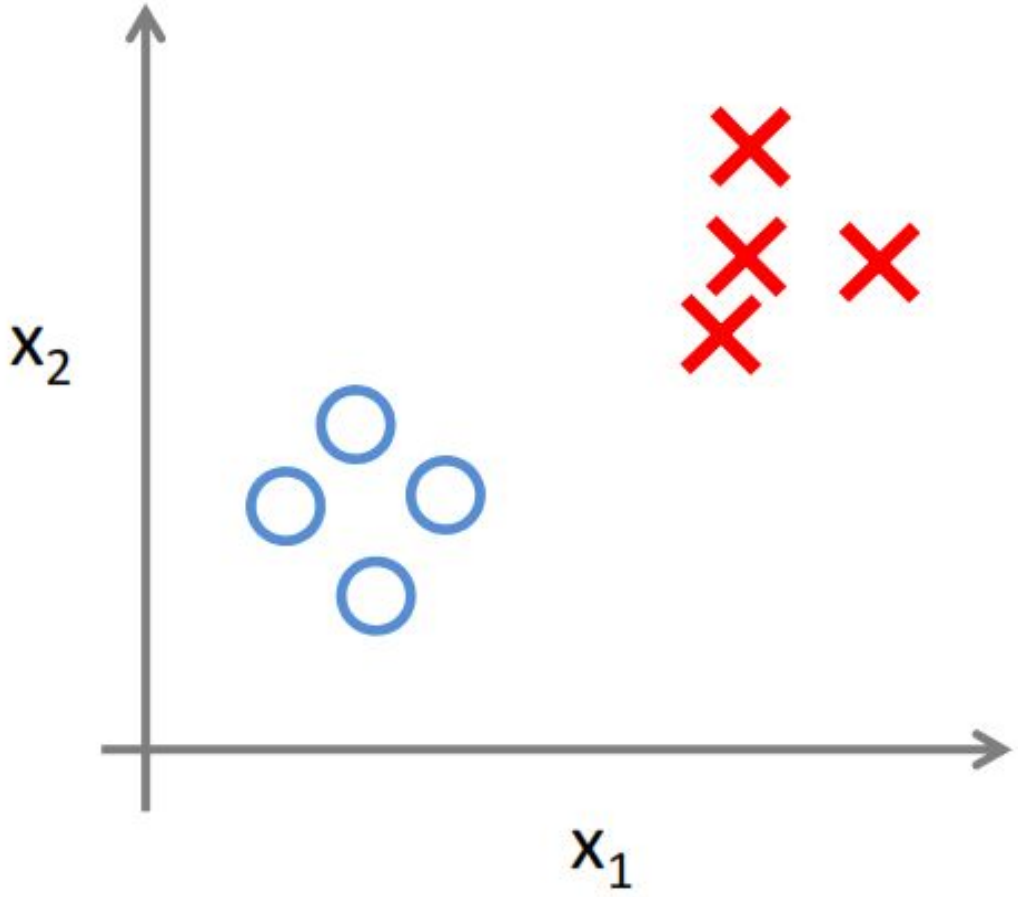
- **Classification**
    - Linear Classifiers
    - Support Vector Machines
    - Decision Trees
    - K-Nearest Neighbor
    - Random Forest

Regression : example



Classification

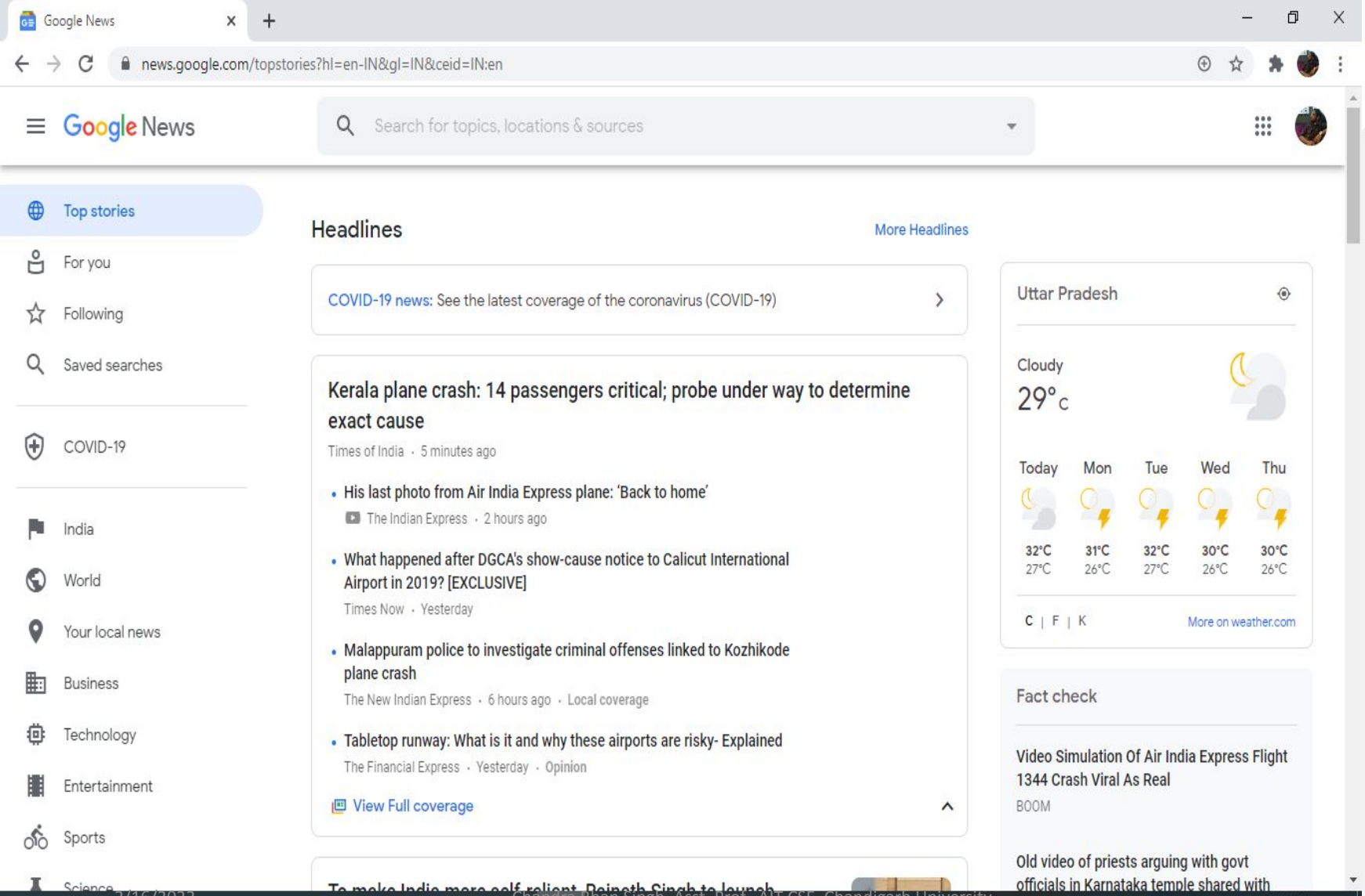
A classification algorithm will be given data points with an assigned category. The job of a classification algorithm is to then take an input value and assign it a class, or category, that it fits into based on the training data provided.


x_2
x_1

Unsupervised learning

Unsupervised learning occurs when an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms. As a kind of learning, it resembles the methods humans use to figure out that certain objects or events are from the same class, such as by observing the degree of similarity between objects.

- K-Means
- SVD
- PCA
- HMM
- Neural Networks
- Fuzzy C-Means


Google News
news.google.com/topstories?hl=en-IN&gl=IN&ceid=IN:en
Google News
Search for topics, locations & sources
Top stories
For you
Following
Saved searches
COVID-19
India
World
Your local news
Business
Technology
Entertainment
Sports
Science
Headlines
More Headlines
COVID-19 news: See the latest coverage of the coronavirus (COVID-19)
Kerala plane crash: 14 passengers critical; probe under way to determine exact cause
Times of India · 5 minutes ago
His last photo from Air India Express plane: 'Back to home'
The Indian Express · 2 hours ago
What happened after DGCA's show-cause notice to Calicut International Airport in 2019? [EXCLUSIVE]
Times Now · Yesterday
Malappuram police to investigate criminal offenses linked to Kozhikode plane crash
The New Indian Express · 6 hours ago · Local coverage
Tabletop runway: What is it and why these airports are risky- Explained
The Financial Express · Yesterday · Opinion
View Full coverage
Uttar Pradesh
Cloudy
29°C
Today Mon Tue Wed Thu
32°C 31°C 32°C 30°C 30°C
27°C 26°C 27°C 26°C 26°C
C | F | K
More on weather.com
Fact check
Video Simulation Of Air India Express Flight 1344 Crash Viral As Real
BOOM
Old video of priests arguing with govt officials in Karnataka temple shared with

Reinforcement learning

Reinforcement learning occurs when you present the algorithm with examples that lack labels, as in unsupervised learning. However, you can accompany an example with positive or negative feedback according to the solution the algorithm proposes. Reinforcement learning is connected to applications for which the algorithm Must make decisions (so the product is prescriptive, not just descriptive, as in unsupervised learning), and the decisions bear consequences. In the human world, it is just like learning by trial and error. Errors help you learn because they have a penalty added (cost, loss of time, regret, pain, and so on), teaching you that a certain course of action is less likely to succeed than others. An interesting example of reinforcement learning occurs when computers learn to play video games by themselves. In this case, an application presents the algorithm with examples of specific situations, such as having the gamer stuck in a maze while avoiding an enemy. The application lets the algorithm know the outcome of actions it takes, and learning occurs while trying to avoid what it discovers to be dangerous and to pursue survival. You can have a look at how the company Google DeepMind has created a reinforcement learning program that plays old Atari videogames at https://www.youtube.com/watch?v=V1eYniJ0Rnk. When watching the video, notice how the program is initially clumsy and unskilled but steadily improves with training until it becomes a champion.

Regression

- Linear
- Logistic

Linear Regression

Regression models are used to predict target variables on a continuous scale, which makes them attractive for addressing many questions in science.

They also have applications in industry, such as understanding relationships between variables, evaluating trends, or making forecasts. One example is predicting the sales of a company in future months.

Introducing linear regression

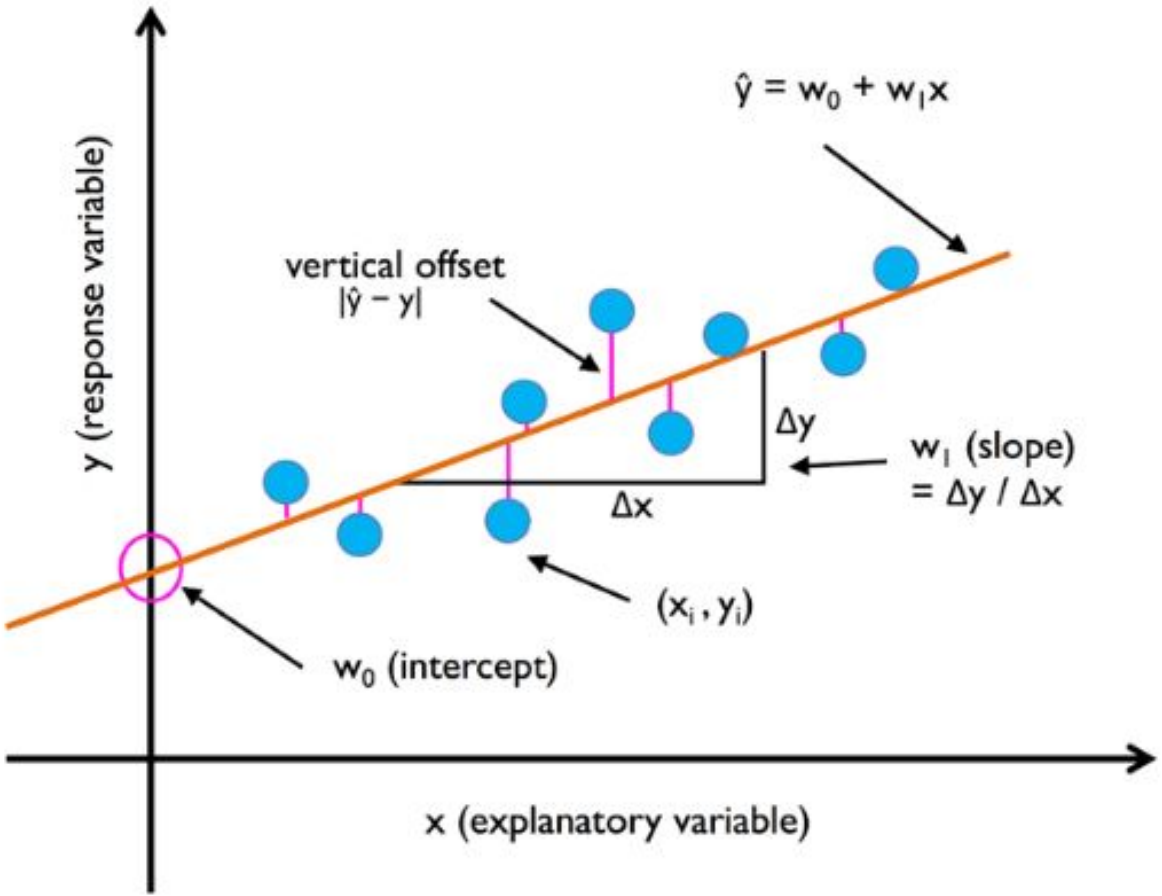
The goal of linear regression is to model the relationship between one or multiple features and a continuous target variable.

In contrast to classification—a different subcategory of supervised learning—regression analysis aims to predict outputs on a continuous scale rather than categorical class labels.

Simple linear regression

- The goal of simple (univariate) linear regression is to model the relationship between a single feature (explanatory variable, x) and a continuous-valued target (response variable, y). The equation of a linear model with one explanatory variable is defined as follows

$$y = w_0 + w_1 x$$


$\hat{y} = w_0 + w_1 x$
vertical offset
$|\hat{y} - y|$
y (response variable)
Δy
w_1 (slope)
$= \Delta y / \Delta x$
Δx
(x_i, y_i)
w_0 (intercept)
x (explanatory variable)

Linear Regression

The values w_0 and w_1 must be chosen so that they minimize the error. If sum of squared error is taken as a metric to evaluate the model, then goal to obtain a line that best reduces the error. If we don't square the error, then positive and negative point will cancel out each other

$$\text{Error} = \sum_{i=1}^{n}(actual\_output - predicted\_output) ** 2$$

$$(\text{Intercept Calculation}) w_0 = y - w_1 x$$

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Co-efficient Formula

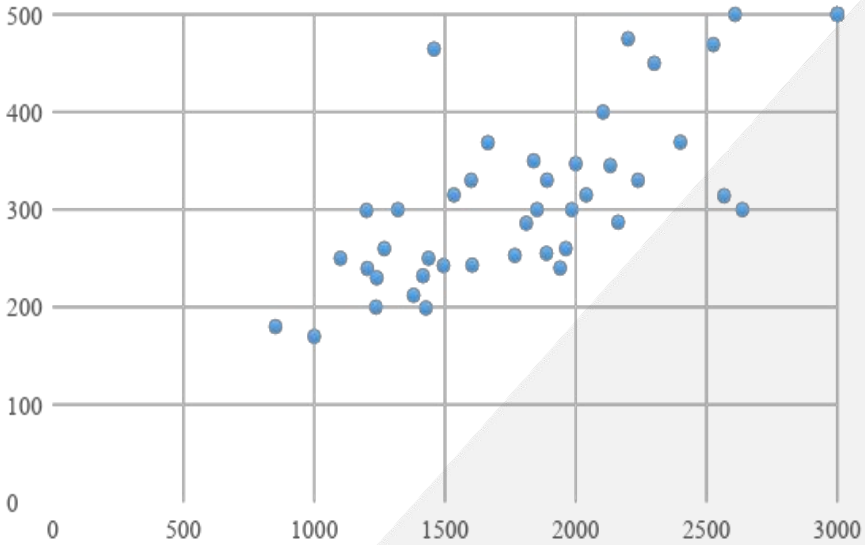
- Exploring 'w_1'
  - If $w_1 > 0$, then x(predictor) and y(target) have a positive relationship. That is increase in x will increase y.
  - If $w_1 < 0$, then x(predictor) and y(target) have a negative relationship. That is increase in x will decrease y.

Exploring w0

- If the model does not include x=0, then the prediction will become meaningless with only w_0. For example, we have a dataset that relates height(x) and weight(y). Taking x=0(that is height as 0), will make equation have only w_0 value which is completely meaningless as in real-time height and weight can never be zero. This resulted due to considering the model values beyond its scope.
- If the model includes value 0, then 'w_0' will be the average of all predicted values when x=0. But, setting zero for all the predictor variables is often impossible.
- The value of w_0 guarantee that residual have mean zero. If there is no 'w_0' term, then regression will be forced to pass over the origin. Both the regression co-efficient and prediction will be biased.

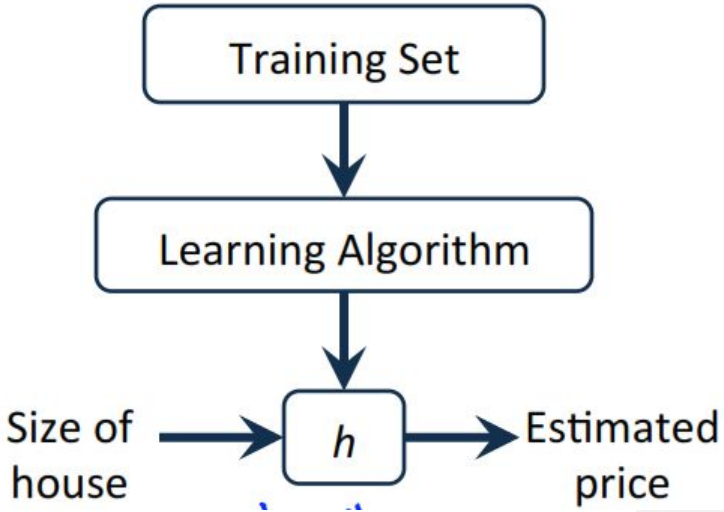
Housing Prices

Price
(in 1000s of
dollars)



Size (feet^2)

Notation:
- m = Number of training examples
- x's = "input" variable / features
- y's = "output" variable / "target" variable


Training Set
Learning Algorithm
Size of house
h
Estimated price

Training Set (m=47)

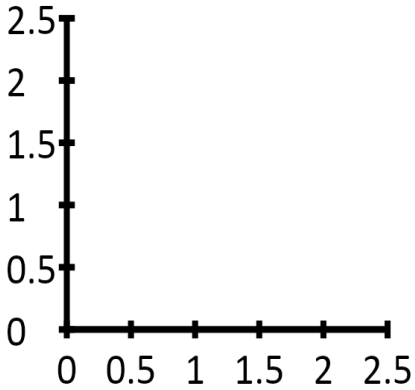
Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

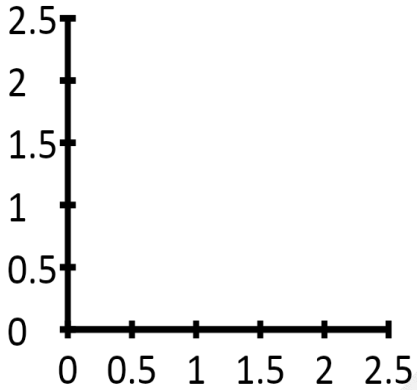
θ_i's: Parameters

How to choose θ_i's ?

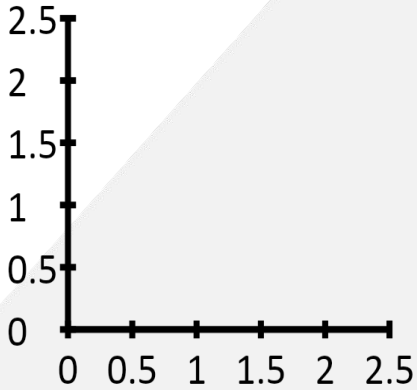
$$h_\theta(x) = \theta_0 + \theta_1 x$$



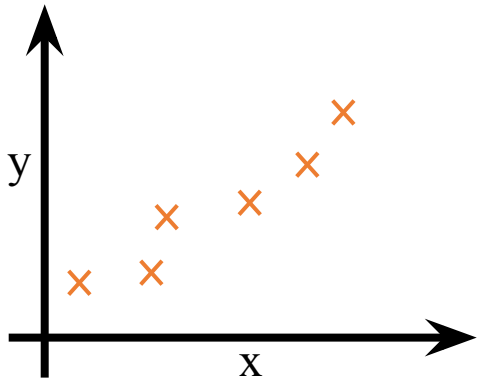
$\theta_0 = 1.5$
$\theta_1 = 0$



$\theta_0 = 0$
$\theta_1 = 0.5$



$\theta_0 = 1$
$\theta_1 = 0.5$



Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

Linear regression with one variable

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Simplified

$$h_\theta(x) = \theta_1 x$$

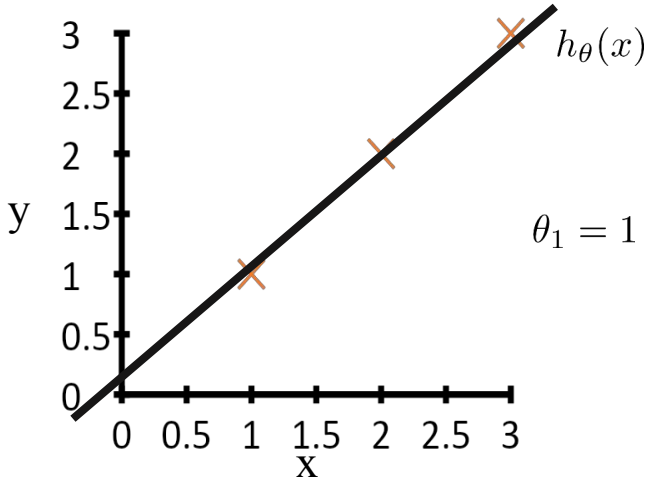
$$\theta_1$$

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$

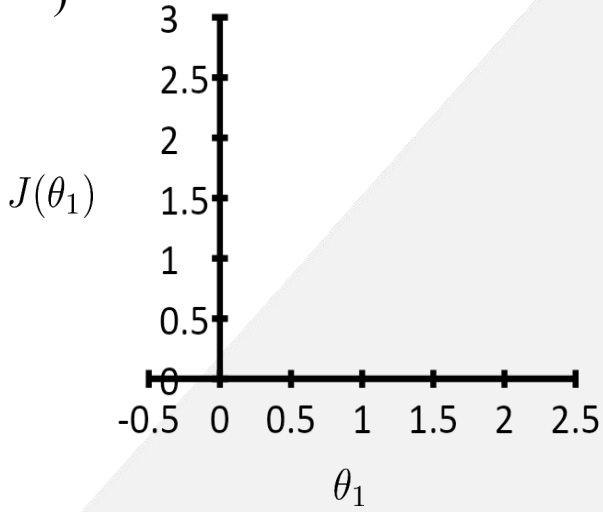
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



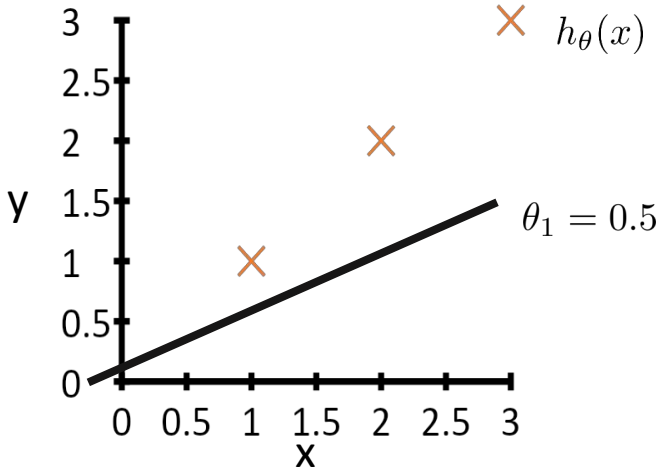
$J(\theta_1)$

(function of the parameter θ_1)



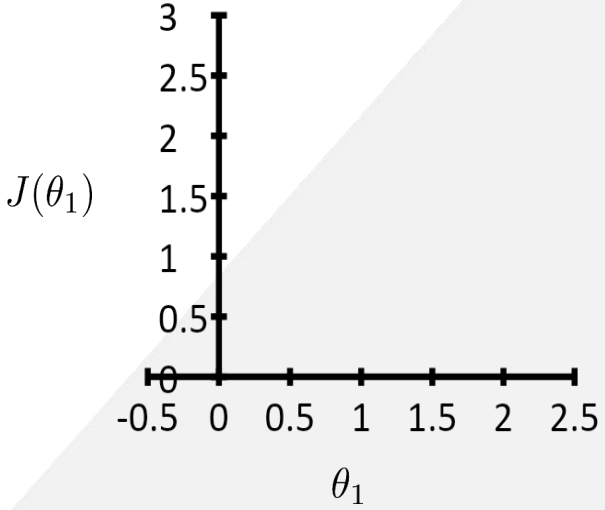
$h_\theta(x)$

(for fixed θ_1 , this is a function of x)



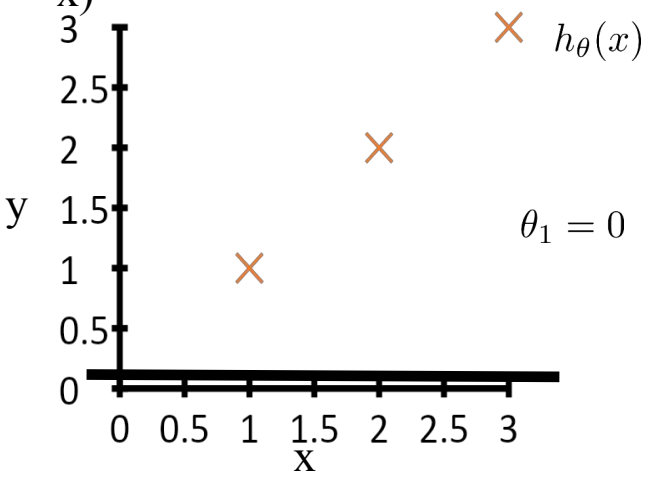
$J(\theta_1)$

(function of the parameter θ_1)



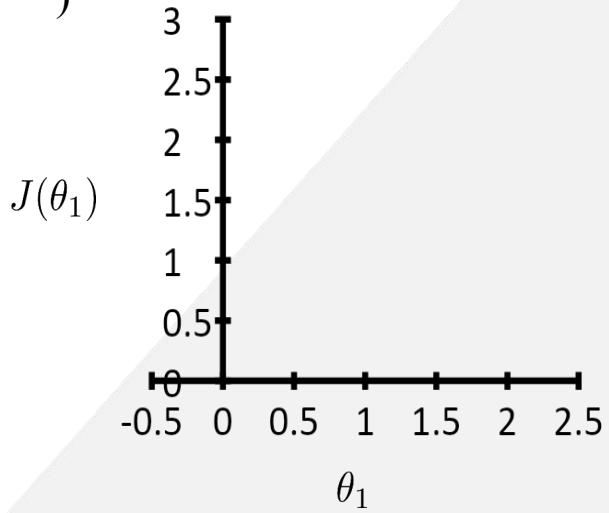
$h_\theta(x)$

(for fixed θ_1 , this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

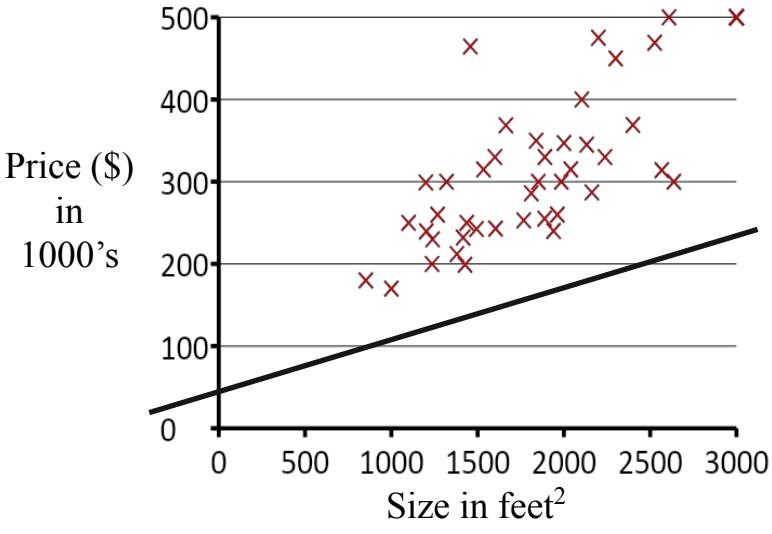
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Cost function with both parameters

$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)

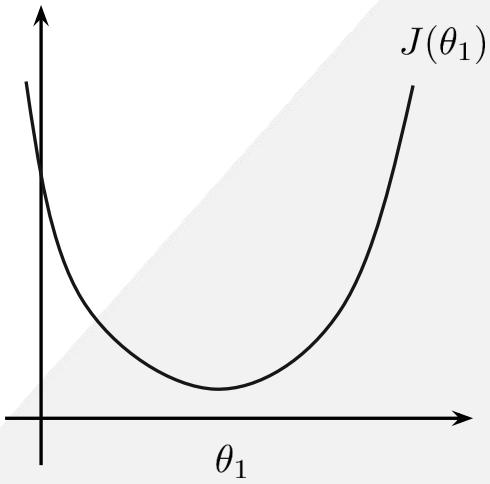


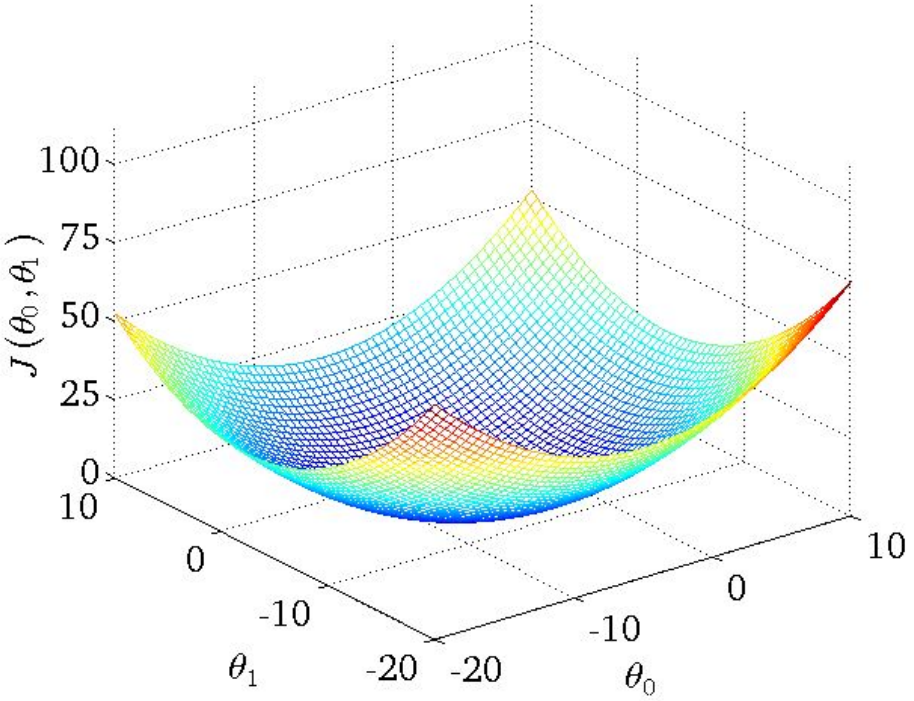
(x)

$h_\theta(x) = 50 + 0.06x$

$J(\theta_0, \theta_1)$

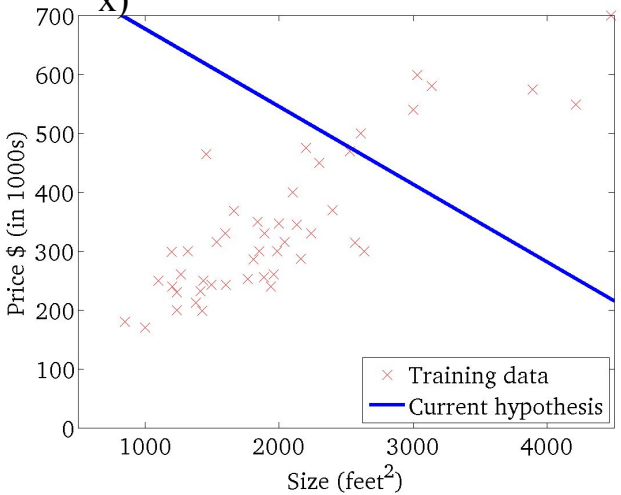
(function of the parameters θ_0, θ_1)




100
75
50
25
0
$J(\theta_0, \theta_1)$
10
0
-10
-20
θ_1
-20
-10
0
10
θ_0

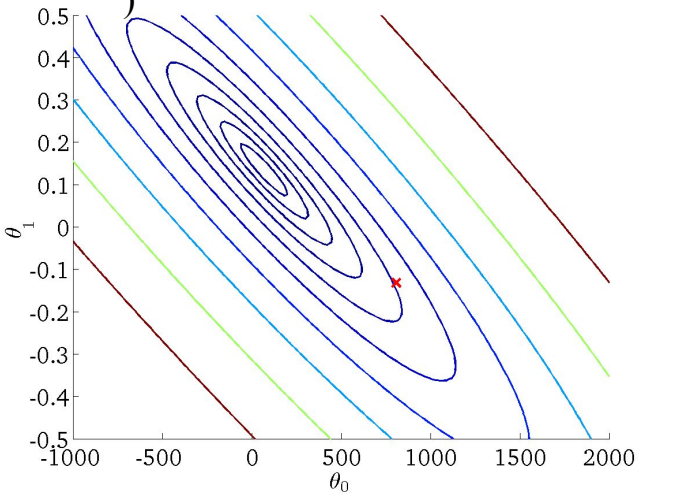
$h_\theta(x)$

(for fixed θ_0, θ_1 , this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



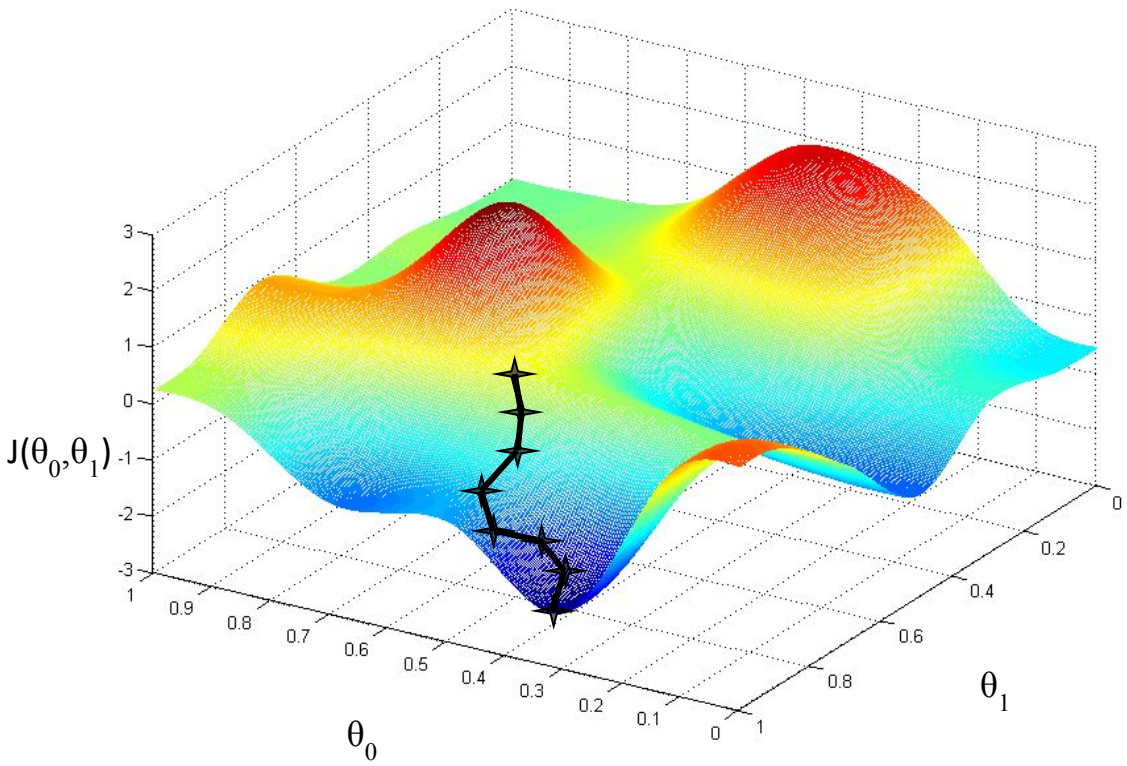
Gradient descent

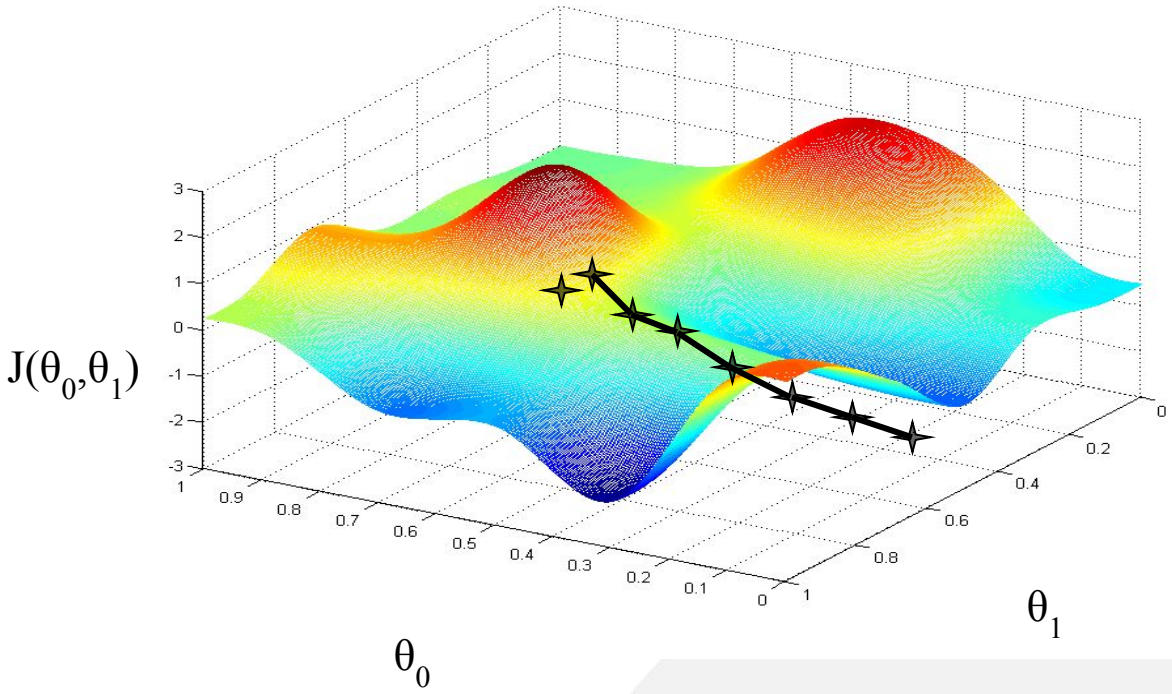
Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum


J(θ0,θ1)
θ0
θ1
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1


J(θ0,θ1)
θ0
θ1
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

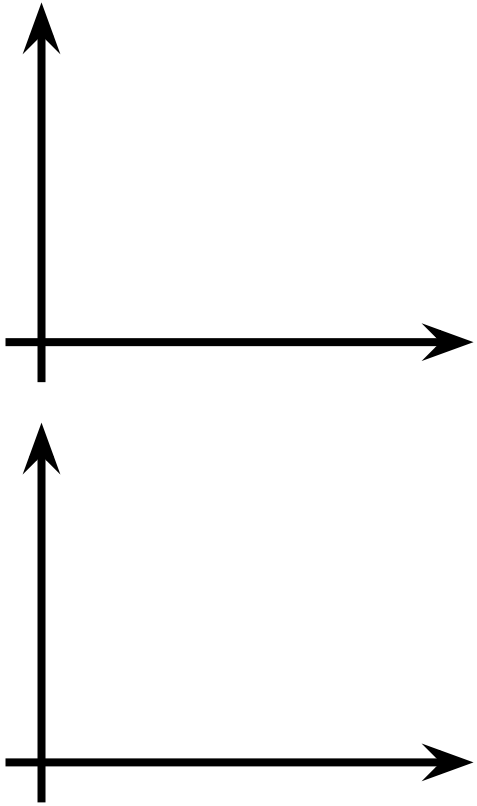
$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$ (simultaneously update $j = 0$ and $j = 1$)

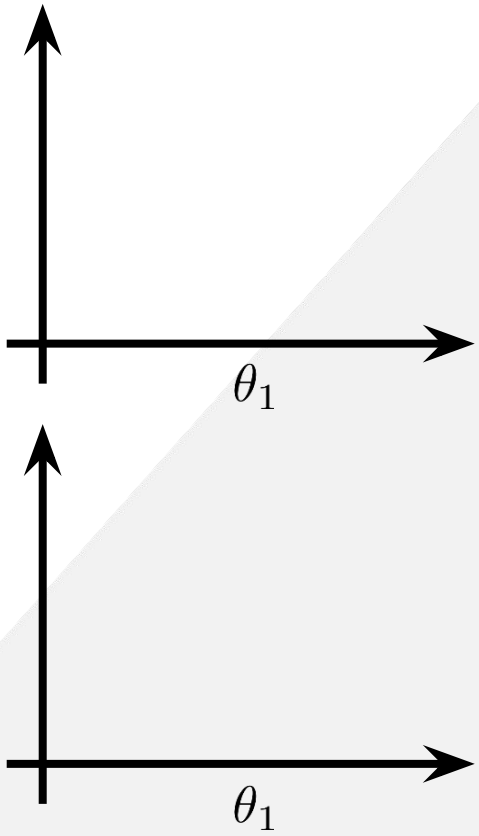
}

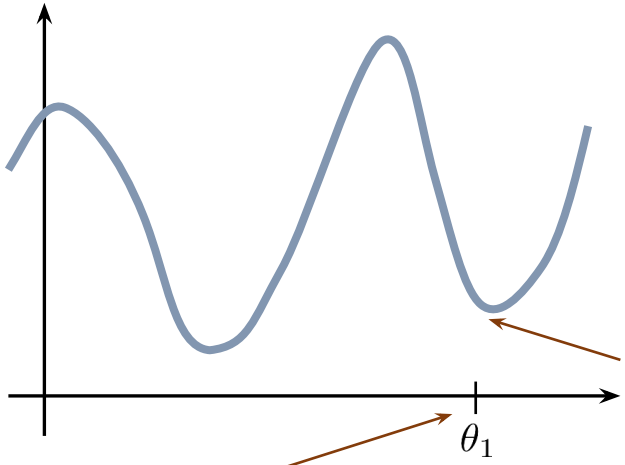


$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



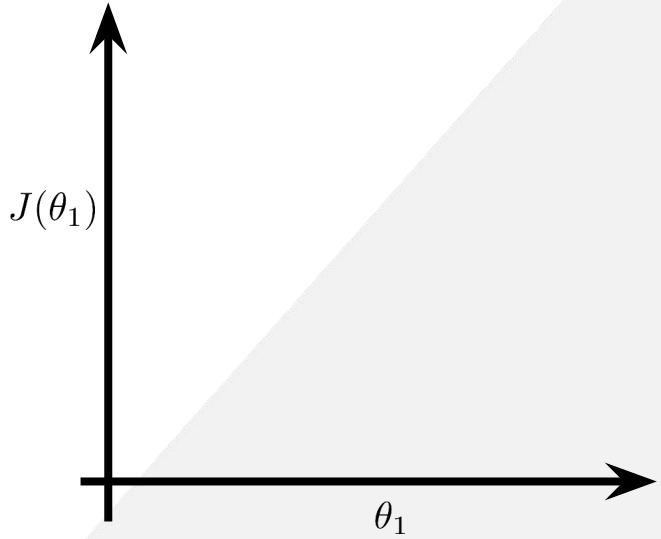


$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) =$$

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) =$$

Gradient descent algorithm

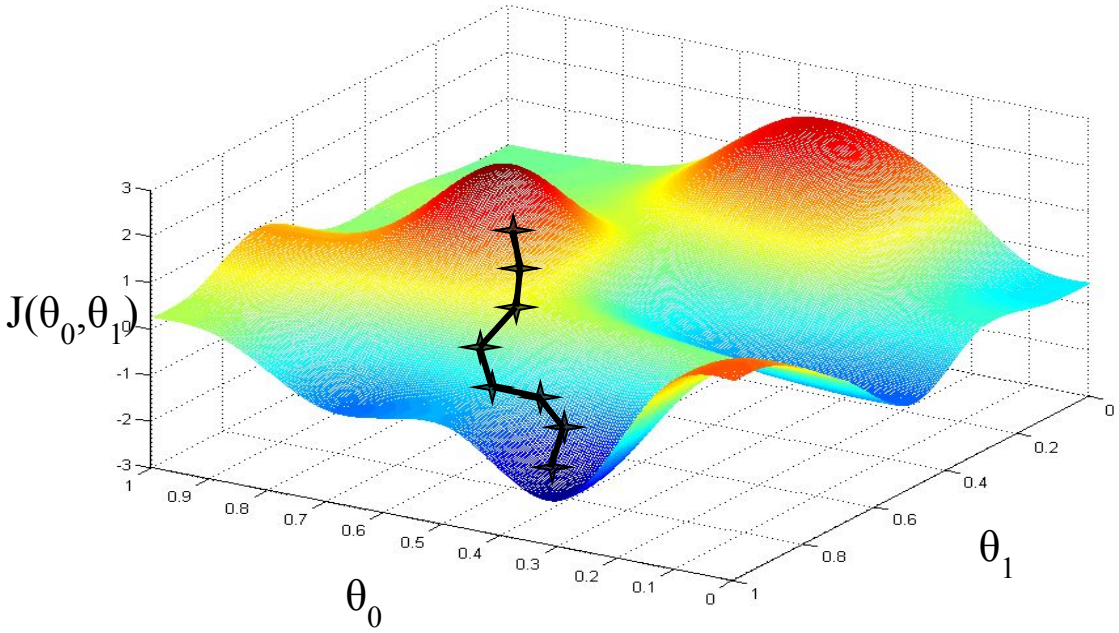
repeat until convergence {

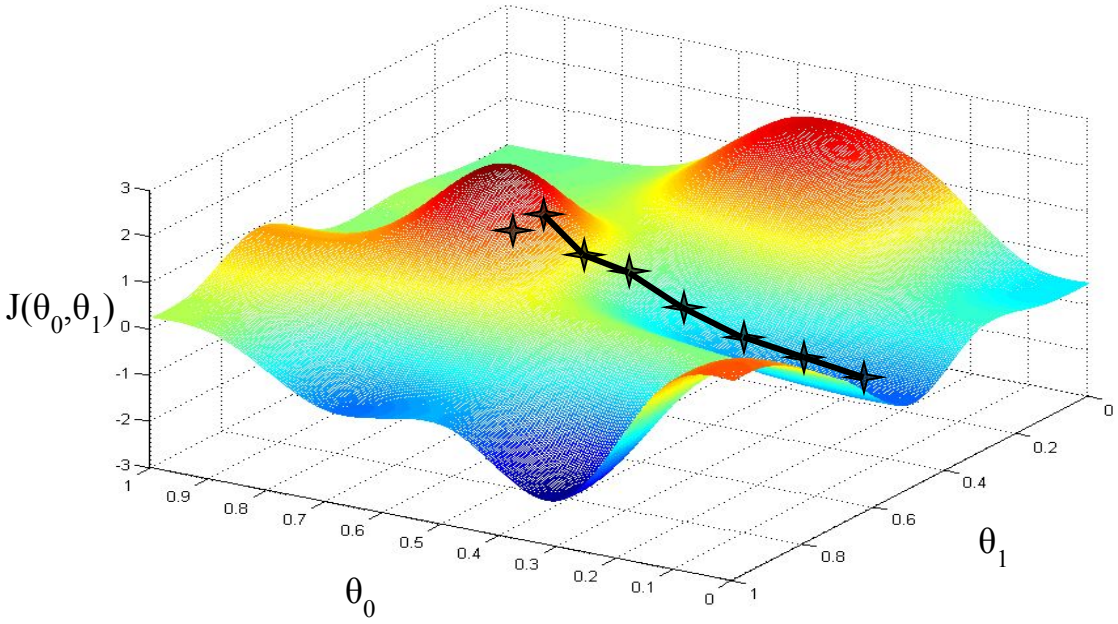
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

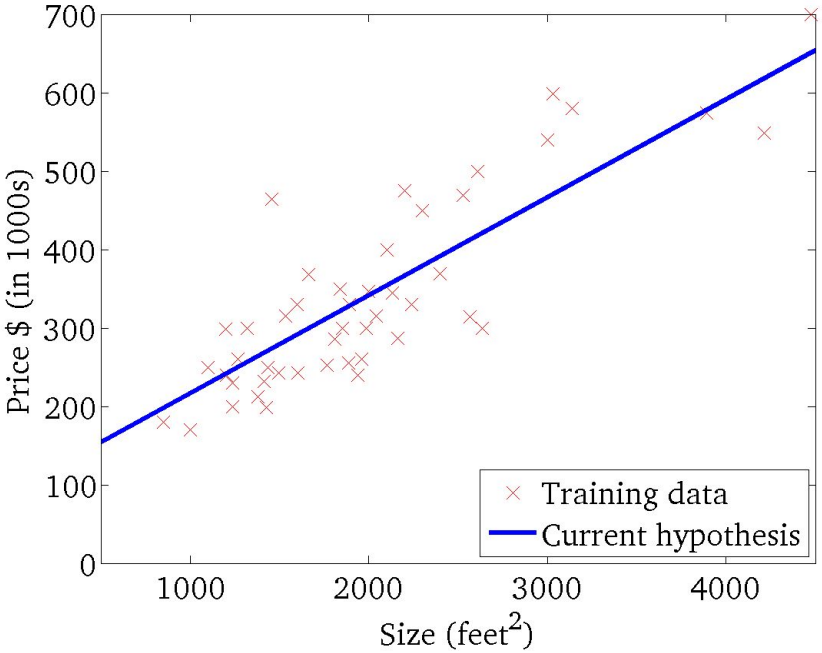
update θ_0 and θ_1 simultaneously


J(θ0,θ1)
θ0
θ1
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.2
0.4
0.6
0.8


J(θ0,θ1)
θ0
θ1
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

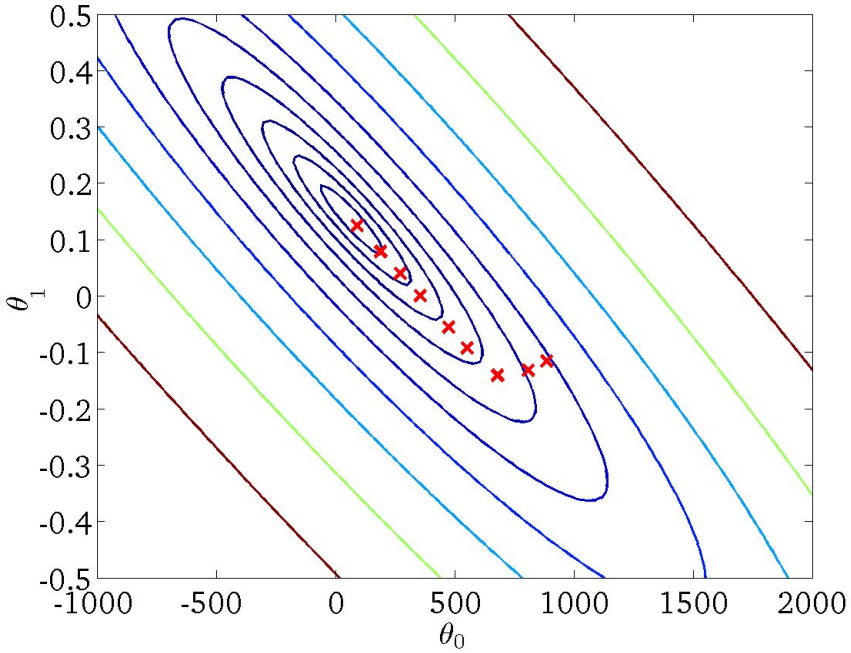
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



“Batch” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples.

Linear Regression with multiple variables

Multiple features

Multiple features (variables).

Size (feet^2) x	Price ($1000) y
2104	460
1416	232
1534	315
852	178
…	…

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Multiple features (variables).

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Multiple features (variables).

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

Hypothesis:

Previously: $h_\theta(x) = \theta_0 + \theta_1 x$

Now: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

For convenience of notation, define $x_0 = 1$

Multivariate linear regression.

Linear Regression with multiple variables

Gradient descent for multiple variables

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function: $J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial\theta_0}J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(simultaneously update for θ_j)

$$j = 0, \ldots, n$$

$\}$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

...

Linear Regression with multiple variables

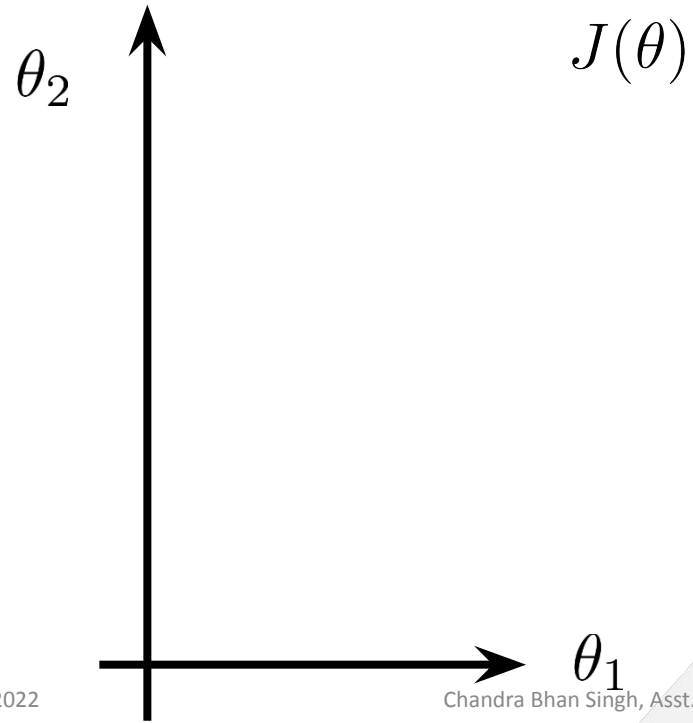
Gradient descent in practice I: Feature Scaling

Feature Scaling

Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet^2)

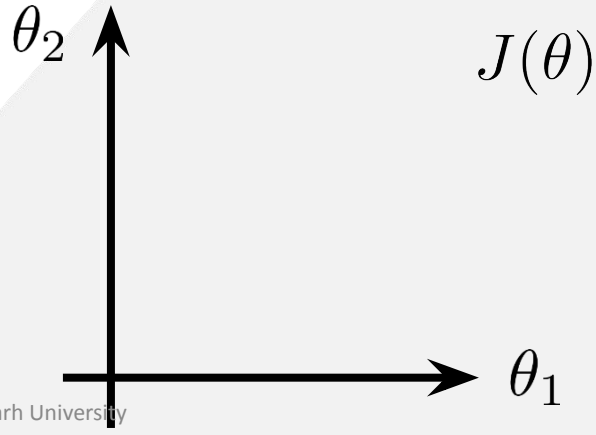
x_2 = number of bedrooms (1-5)



size (feet^2)

$$x_1 = \frac{size(feet^2)}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Feature Scaling

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

E.g. $x_1 = \frac{size - 1000}{2000}$

$x_2 = \frac{\#bedrooms - 2}{5}$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

Linear Regression with multiple variables

Gradient descent in practice II: Learning rate

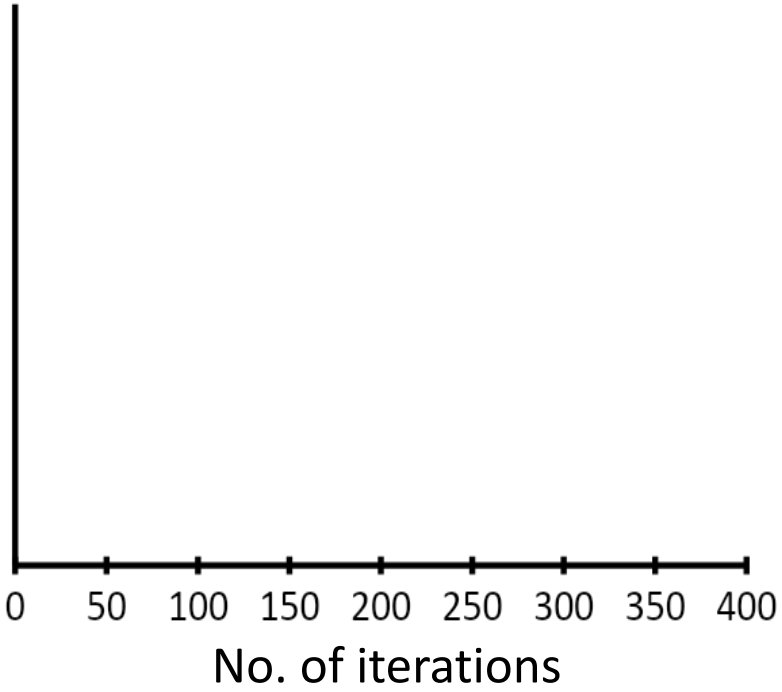
Gradient descent

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- "Debugging": How to make sure gradient descent is working correctly.
- How to choose learning rate α.

Making sure gradient descent is working correctly.

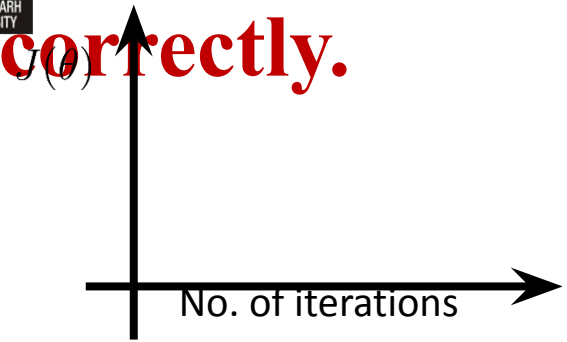
$\min_{\theta} J(\theta)$



Example automatic convergence test:

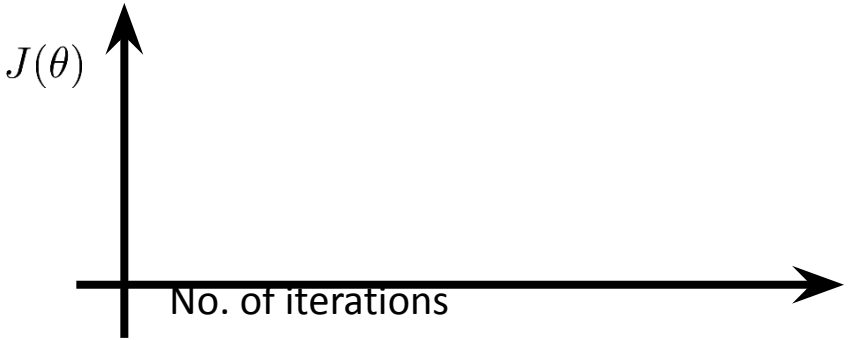
Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

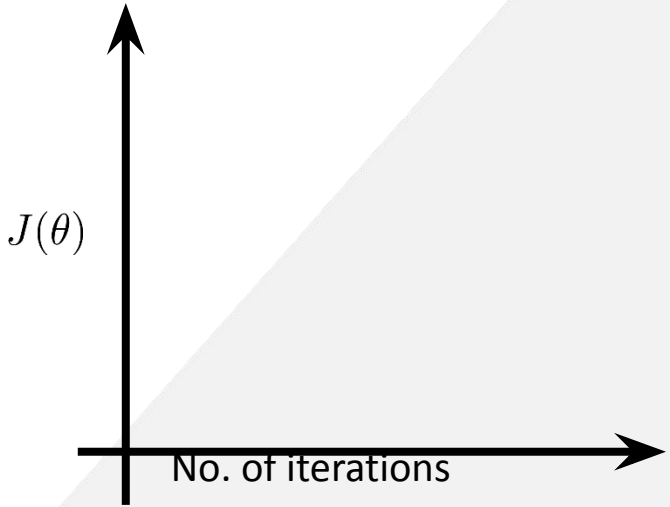
Making sure gradient descent is working correctly.



Gradient descent not working.

Use smaller α





- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Summary:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose α, try

$$\ldots, 0.001, \quad , 0.01, \quad , 0.1, \quad , 1, \ldots$$

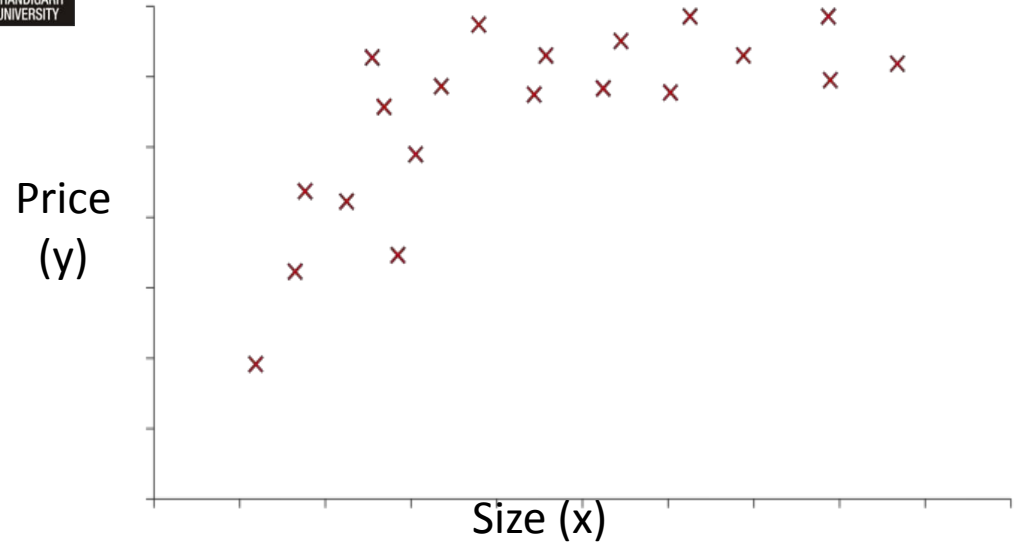
Linear Regression with multiple variables

Features and polynomial regression

Housing prices prediction

$$h_\theta(x) = \theta_0 + \theta_1 \times frontage + \theta_2 \times depth$$

Polynomial regression



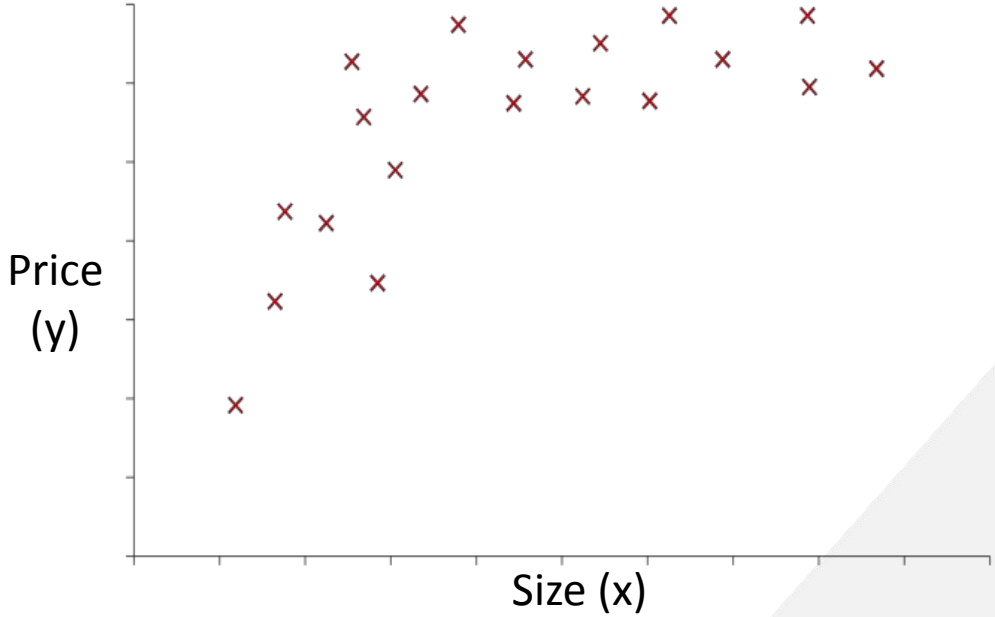
$$\theta_0 + \theta_1 x + \theta_2 x^2$$

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$

Choice of features



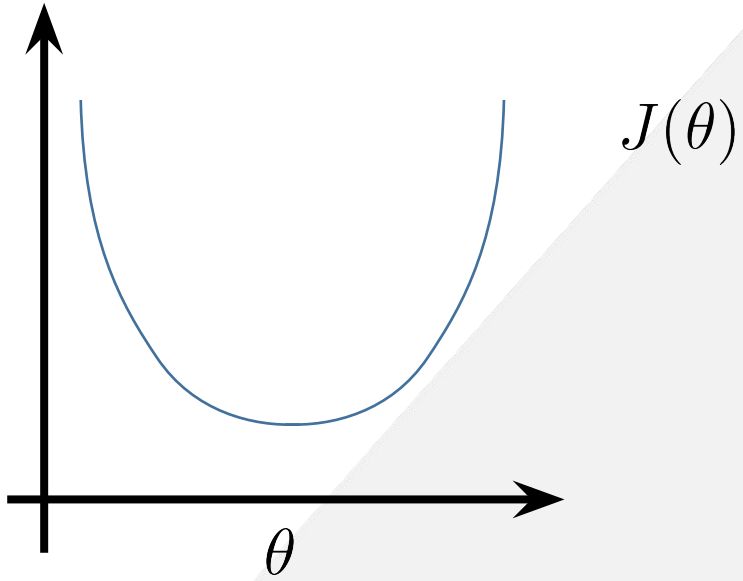
$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$

$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

Linear Regression with multiple variables

Normal equation

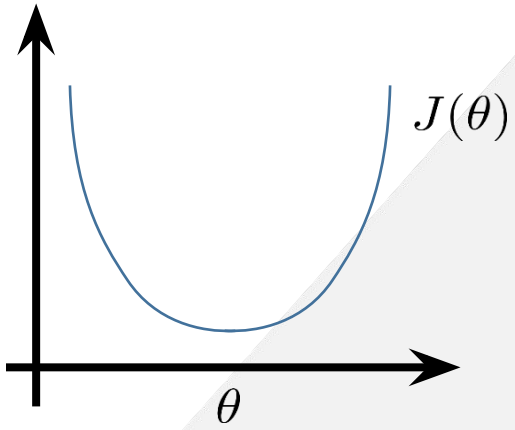
Gradient Descent



Normal equation: Method to solve for θ analytically.

Intuition: If 1D $(\theta \in \mathbb{R})$

$$J(\theta) = a\theta^2 + b\theta + c$$



$$\theta \in \mathbb{R}^{n+1} \qquad J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0 \quad \text{(for every } j\text{)}$$

Solve for $\theta_0, \theta_1, \ldots, \theta_n$

Examples: $m = 4$.

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

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

Examples: $m = 5.$

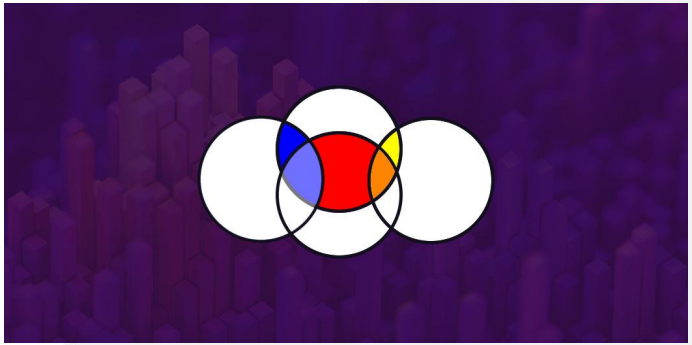
x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178
1	3000	4	1	38	540

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ 1 & 3000 & 4 & 1 & 38 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \\ 540 \end{bmatrix}$$

$$\Theta = (X^T X)^{-1} X^T y$$

Multicollinearity

- Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model.



The Problem with having Multicollinearity

- Multicollinearity can be a problem in a regression model because we would not be able to distinguish between the individual effects of the independent variables on the dependent variable. For example, let's assume that in the following linear equation:

$$Y = W_0 + W_1 * X_1 + W_2 * X_2$$

- Coefficient W_1 is the increase in Y for a unit increase in X_1 while keeping X_2 constant. But since X_1 and X_2 are highly correlated, changes in X_1 would also cause changes in X_2 and we would not be able to see their individual effect on Y.

" This makes the effects of X_1 on Y difficult to distinguish from the effects of X_2 on Y. "

What causes Multicollinearity?

- Multicollinearity could exist because of the problems in the dataset at the time of creation. These problems could be because of poorly designed experiments, highly observational data, or the inability to manipulate the data.
- Multicollinearity could also occur when new variables are created which are dependent on other variables.
- Including identical variables in the dataset.
- Inaccurate use of dummy variables can also cause a multicollinearity problem. This is called the **Dummy variable trap.**
- Insufficient data in some cases can also cause multicollinearity problems.

Detecting Multicollinearity using VIF(Variable Inflation Factors)

- VIF determines the strength of the correlation between the independent variables. It is predicted by taking a variable and regressing it against every other variable. “
- VIF score of an independent variable represents how well the variable is explained by other independent variables

Recursive Feature Elimination (RFE) for Feature Selection

- RFE is popular because it is easy to configure and use and because it is effective at selecting those features (columns) in a training dataset that are more or most relevant in predicting the target variable.
- There are two important configuration options when using RFE: the choice in the number of features to select and the choice of the algorithm used to help choose features. Both of these hyperparameters can be explored, although the performance of the method is not strongly dependent on these hyperparameters being configured well.

Recursive Feature Elimination

Recursive Feature Elimination, or RFE for short, is a feature selection algorithm. A machine learning dataset for classification or regression is comprised of rows and columns, like an excel spreadsheet. Rows are often referred to as samples and columns are referred to as features, e.g. features of an observation in a problem domain. Feature selection refers to techniques that select a subset of the most relevant features (columns) for a dataset. Fewer features can allow machine learning algorithms to run more efficiently (less space or time complexity) and be more effective. Some machine learning algorithms can be misled by irrelevant input features, resulting in worse predictive performance.

Recursive Feature Elimination

RFE is a wrapper-type feature selection algorithm. This means that a different machine learning algorithm is given and used in the core of the method, is wrapped by RFE, and used to help select features. This is in contrast to filter-based feature selections that score each feature and select those features with the largest (or smallest) score. RFE works by searching for a subset of features by starting with all features in the training dataset and successfully removing features until the desired number remains.

```
# evaluate RFE for regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeRegressor
from sklearn.pipeline import Pipeline
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5,
random_state=1)
# create pipeline
rfe = RFE(estimator=DecisionTreeRegressor(), n_features_to_select=5)
model = DecisionTreeRegressor()
pipeline = Pipeline(steps=[('s',rfe),('m',model)])
# evaluate model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(pipeline, X, y, scoring='neg_mean_absolute_error',
cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

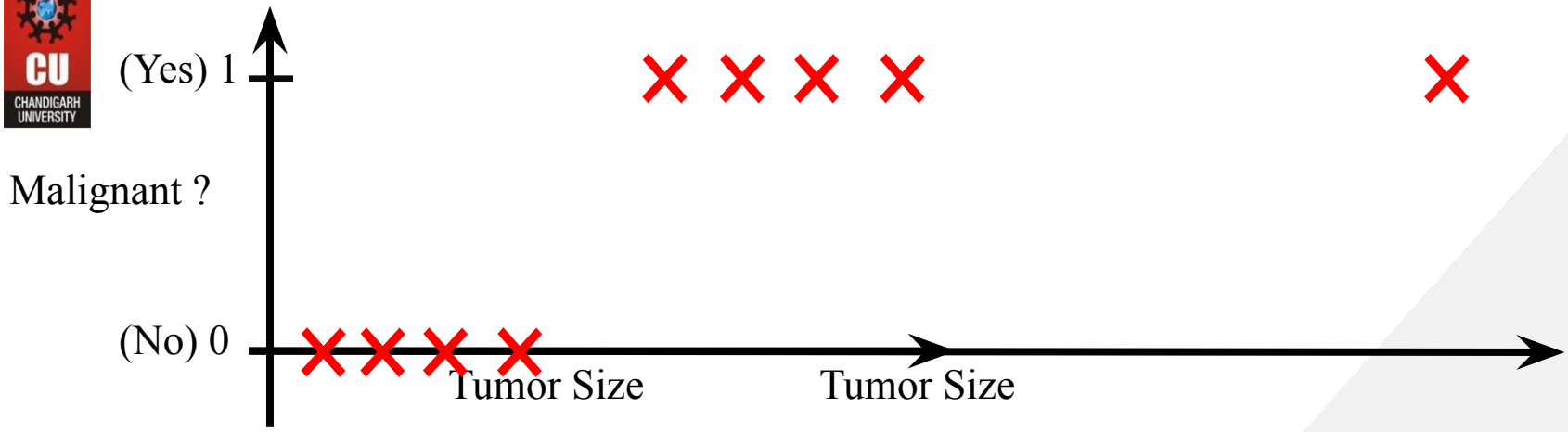
Classification

Email: Spam / Not Spam?
Online Transactions: Fraudulent (Yes / No)?
Tumor: Malignant / Benign ?

$y \in \{0, 1\}$

0: “Negative Class” (e.g., benign tumor)

1: “Positive Class” (e.g., malignant tumor)



Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

Classification: y = 0 or 1

$h_\theta(x)$ can be > 1 or < 0

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

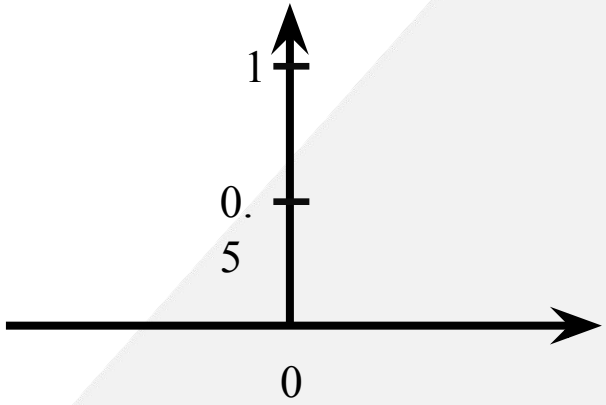
Logistic Regression

As demonstrated, linear regression is a useful technique to quantify relationships between continuous variables. Now, Predicting discrete variables plays a major part in data analysis and machine learning. For instance, is something “A” or “B?” Is it “positive” or “negative?” Is this person a “new customer” or a “returning customer?” Unlike linear regression, the dependent variable (y) is no longer a continuous variable (such as price) but rather a discrete categorical variable. The independent variables used as input to predict the dependent variable can be either categorical or continuous.

Logistic Regression Model

Want $0 \leq h_\theta(x) \leq 1$

$h_\theta(x) = \quad \theta^T x$



Sigmoid function
Logistic function

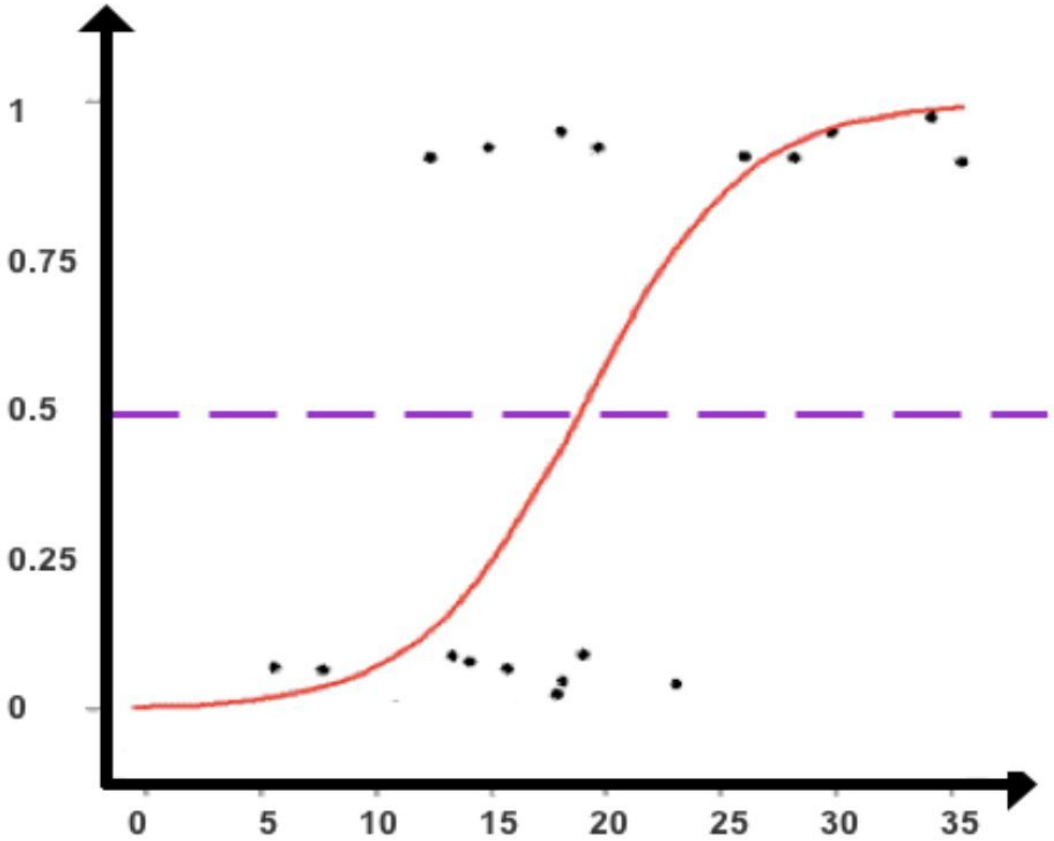


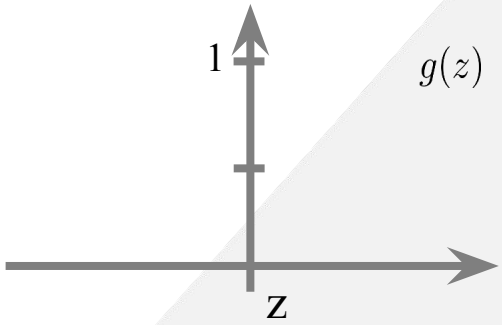
Figure : A sigmoid function used to classify data points

Decision boundary

Logistic regression

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$



Suppose predict "$y = 1$ " if$h_\theta(x) \geq 0.5$

predict " $y =$"0 if $h_\theta(x) < 0.5$

Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that y = 1 on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$$h_\theta(x) = 0.7$$

Tell patient that 70% chance of tumor being malignant

"probability that y = 1, given x, parameterized by θ"

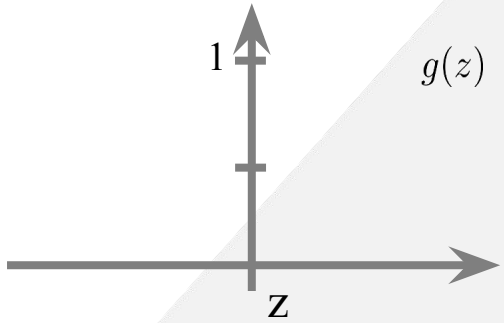
$$P(y = 0|x;\theta) + P(y = 1|x;\theta) = 1$$
$$P(y = 0|x;\theta) = 1 - P(y = 1|x;\theta)$$

Decision boundary

Logistic regression

$$h_\theta(x) = g(\theta^T x)$$

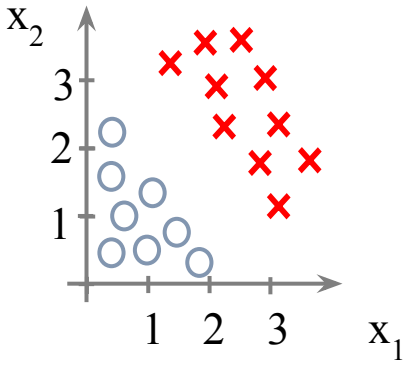
$$g(z) = \frac{1}{1+e^{-z}}$$



Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

predict "$y = 0$" if $h_\theta(x) < 0.5$

Decision Boundary

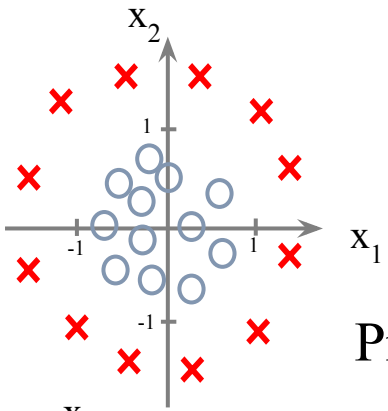


$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

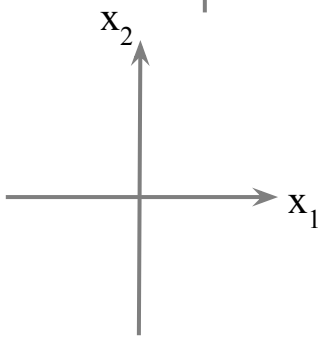
How to fit (find) Parameter $\underline{\theta}$: Parameter $\underline{\theta}$ ($\underline{\theta}_0$, $\underline{\theta}_1$, $\underline{\theta}_2$) defines the decision boundary not the training set. Training set may be used to find the Parameter $\underline{\theta}$

Non-linear decision boundaries



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$

Cost function

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix}$ $x_0 = 1, y \in \{0, 1\}$

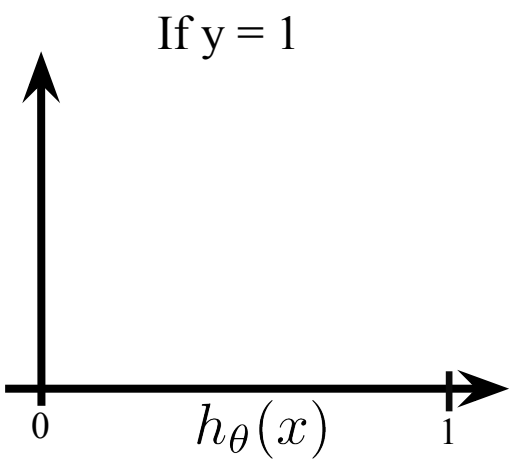
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters θ?

Logistic regression cost function

Different Cost Function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Logistic regression cost function

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

[redacted]

To fit parameters θ:

$$\min_\theta J(\theta)$$

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Gradient Descent

[redacted]

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

(simultaneously update all θ_j)

$\}$

Gradient Descent

[redacted]

Want $\min_\theta J(\theta)$:

Repeat $\{$

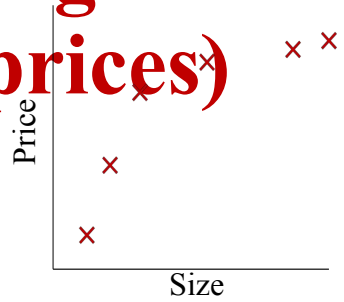
$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

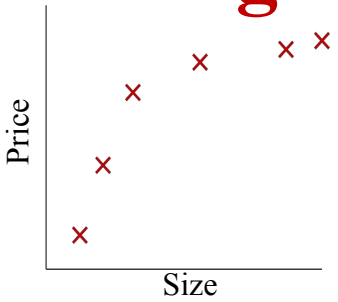
$\}$

Algorithm looks identical to linear regression!

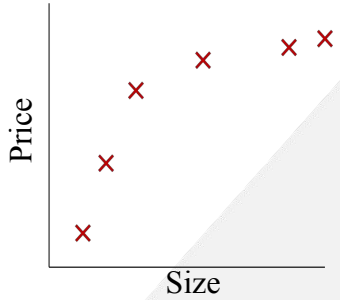
Example: Linear regression (housing prices)



$\theta_0 + \theta_1 x$



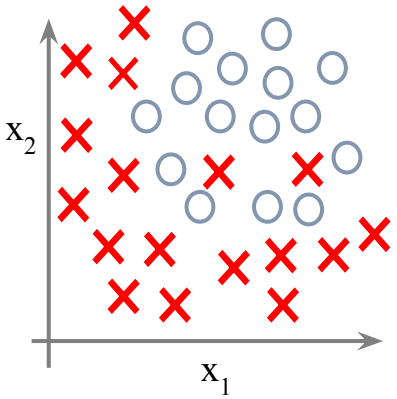
$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

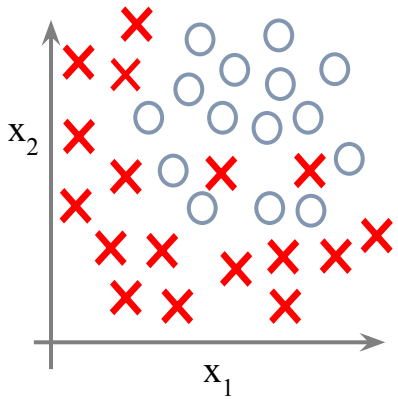
Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Example: Logistic regression

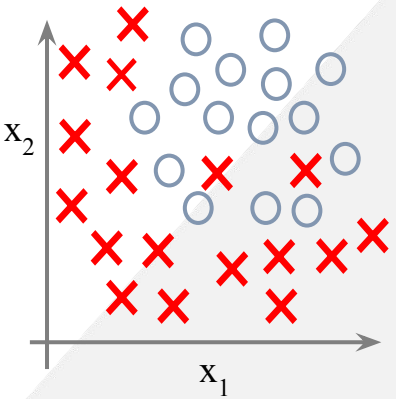


$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

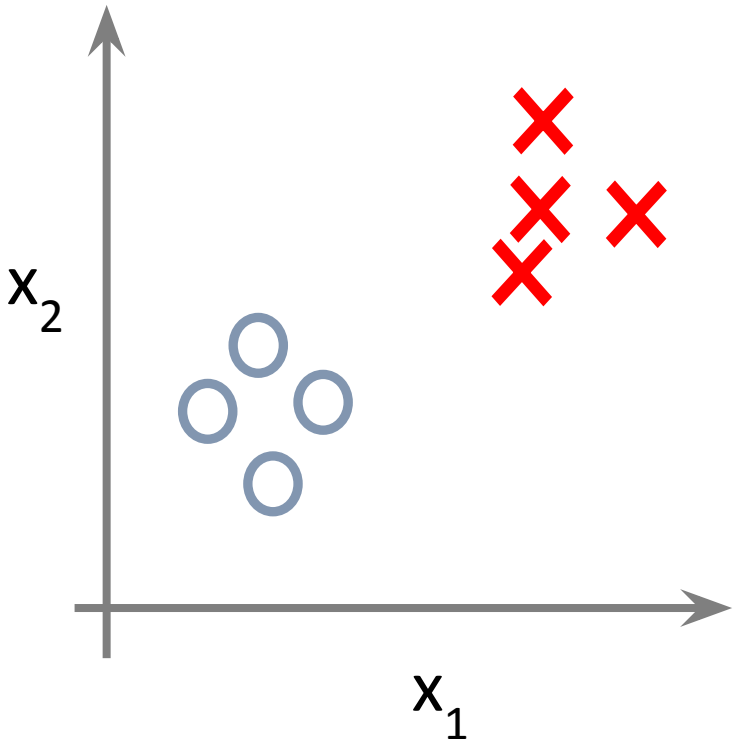
Logistic Regression

Multi-class classification: One-Vs-all

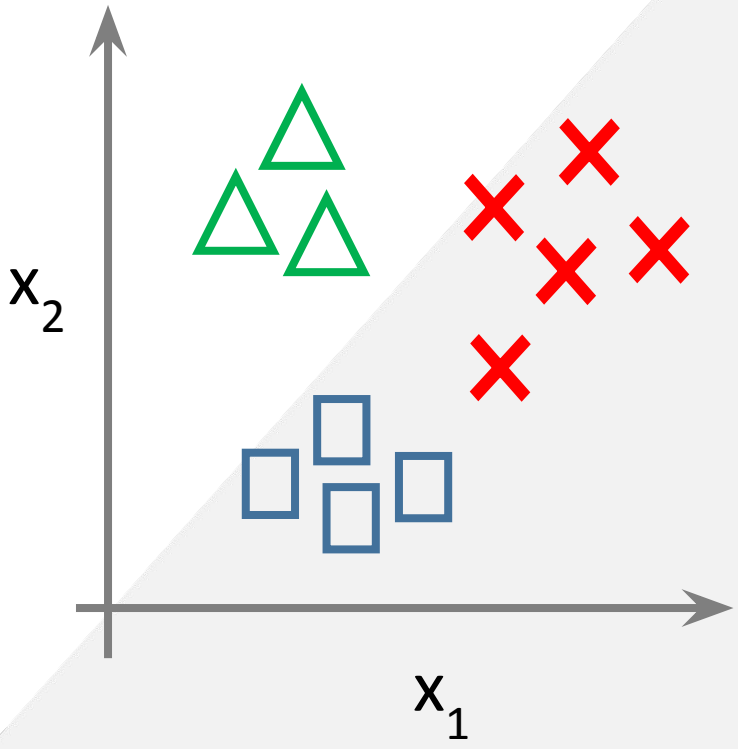
Multiclass classification

- Email folder /tagging: Work, Friends, Family, Hobby
- Medical diagrams: Not ill, Cold, Flu
- Weather: Sunny, Cloudy, Rain, Snow

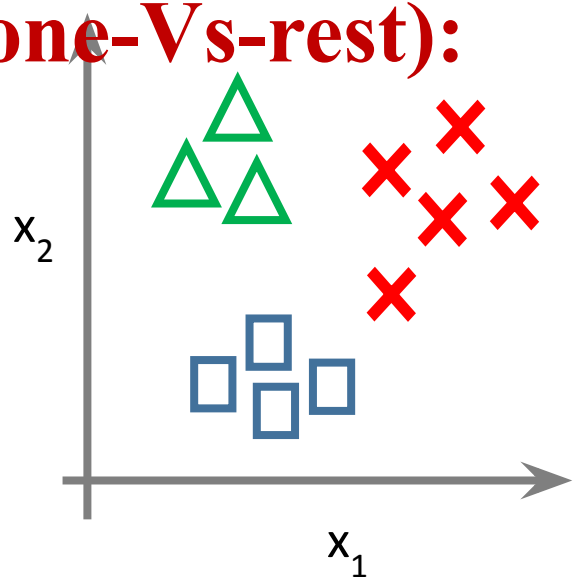
Binary classification:



Multi-class classification:



One-Vs-all (one-Vs-rest):

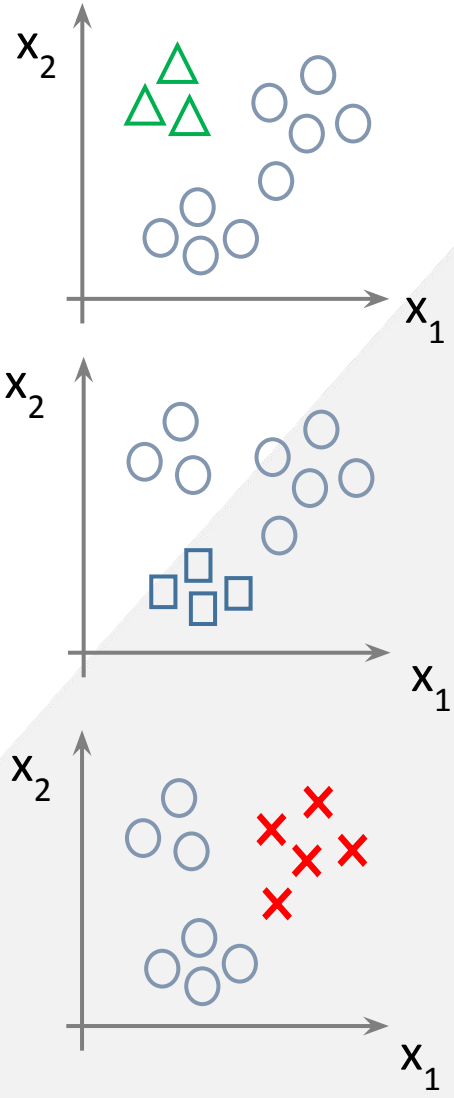


Class 1: △

Class 2: □

Class 3: ×

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$



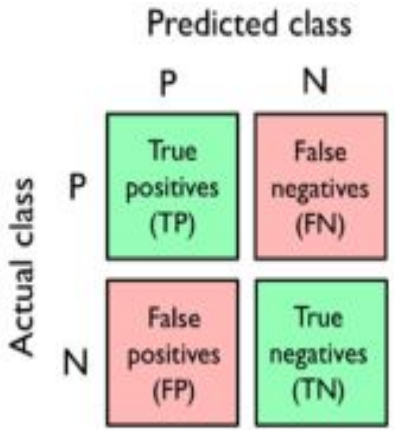
One-Vs-all

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes $\max_i h_\theta^{(i)}(x)$

Confusion Matrix

A confusion matrix is simply a square matrix that reports the counts of the **true positive (TP), true negative (TN), false positive (FP), and false negative (FN)** predictions of a classifier, as shown in the following figure:



Confusion Matrix

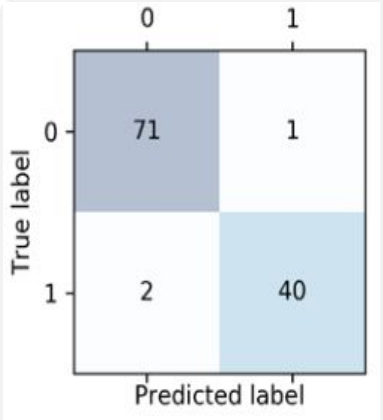
Although these metrics can be easily computed manually by comparing the true and predicted class labels, scikit-learn provides a convenient confusion_matrix function that we can use, as follows:

```
from sklearn.metrics import confusion_matrix
pipe_svc.fit(X_train, y_train)
y_pred = pipe_svc.predict(X_test)
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(confmat)
[[71  1]
[ 2 40]]
```

Confusion Matrix

The array that was returned after executing the code provides us with information about the different types of error the classifier made on the test dataset. We can map this information onto the confusion matrix illustration in the previous figure using Matplotlib's matshow function

```
fig, ax = plt.subplots(figsize=(2.5, 2.5))
>>> ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
>>> for i in range(confmat.shape[0]):
...     for j in range(confmat.shape[1]):
...         ax.text(x=j, y=i,
...                 s=confmat[i, j],
...                 va='center', ha='center')
>>> plt.xlabel('Predicted label')
>>> plt.ylabel('True label')
>>> plt.show()
```



Model Evaluation

Advanced Regression

- Generalized Linear Regression
- Regularized Regression
- Ridge
- Lasso Regression
- Feature Selection

Regularized Regression

- Regularization is a very important concept that is used to avoid overfitting of the data especially when the trained and tested data are much varying.
- Regularization is implemented by adding a “penalty” term to the best fit derived from the trained data, in order to achieve a *lesser variance* with the tested data and also restricts the influence of predictor variables over the output variable by compressing their coefficients.

Ridge Regression

Ridge regression is a technique that is implemented by adding bias to a multilinear regression model to expect a much accurate regression with tested data at a cost of losing accuracy for the training data.

The general equation of a best-fit line for multilinear regression is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots \beta_k x_k$$

where y is the output variable and x1,x2…xk are predictor variables.

The penalty term for ridge regression is λ(slope) ², where lambda denotes the degree of deflection from the original curve by restricting the coefficients of predictor variables but never makes them zero. Therefore the equation for ridge regression is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots \beta_k x_k + \lambda(\text{slope})^2$$

Ridge Regression

Courtesy: https://towardsdatascience.com/regression-with-regularization-techniques-7bbc1a26d9ba

Lasso Regression

Lasso regression is much similar to ridge regression but only differs in the penalty term. The penalty for lasso regression is λ|slope|. Lesso regression can even eliminate the variables by making their coefficients to zero thus removing the variables that have high covariance with other predictor variables.

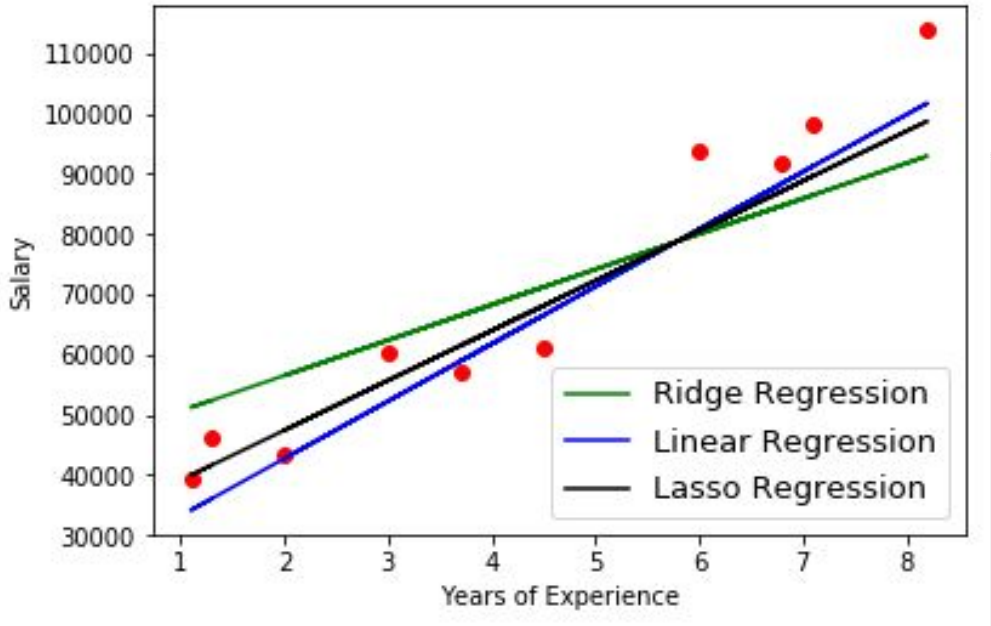
The equation for lasso regression is

$$y = \beta 0 + \beta 1x1 + \beta 2x2 + \cdots \beta kxk + \lambda|slope|$$

Lasso Regression

Courtesy: https://towardsdatascience.com/regression-with-regularization-techniques-7bbc1a26d9ba

Comparison of lasso and ridge with the linear regression model



Courtesy: https://towardsdatascience.com/regression-with-regularization-techniques-7bbc1a26d9ba