Introdução à Aprendizagem Automática (IAA)

SUSANA BRÁS

SUSANA.BRAS@UA.PT

IAA – L3

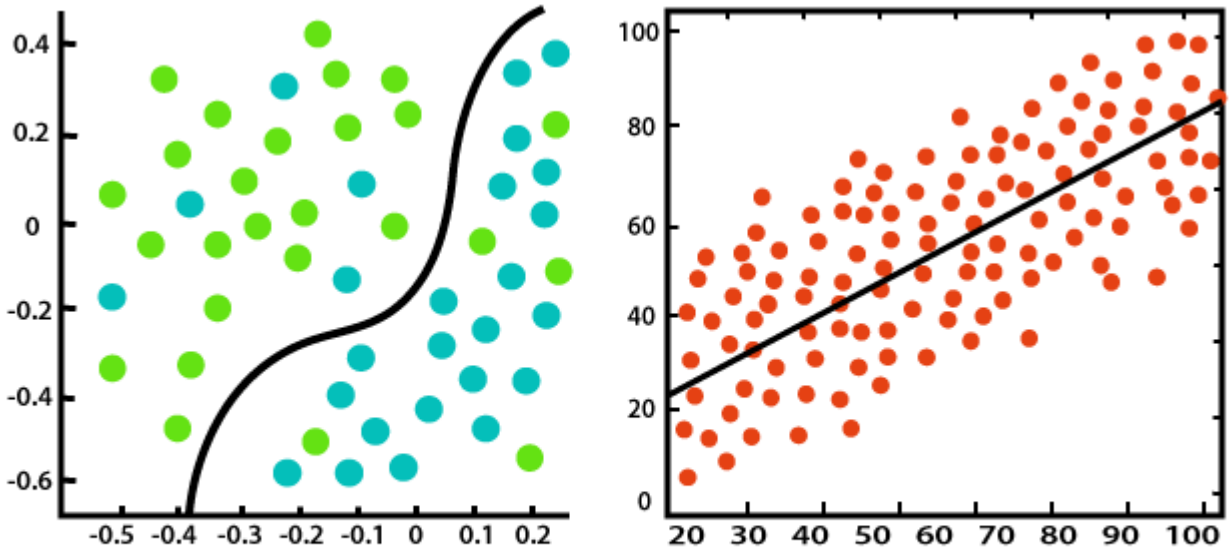
Linear regression – univariate and multivariate

- As a method to identify relations between variables
- A model for prediction
- Regression model
- Error
- Cost function
- Model optimization, parameter fit – gradient descent
- Optimization convergence – local vs global minimum
- Learning rate
- Overfit problem – how to identify, how to deal with it
- Identify overfit, underfit, balance/ good fit
- Regularization:
  - Ridge
  - Lasso
  - Elastic net
  - Characteristics and differences between them

Supervised Learning

Regression and Classification algorithms are Supervised Learning algorithms. Both algorithms are used for prediction in Machine learning and work with labeled datasets. However, the difference between the two lies in their application to various machine learning problems.

The main difference between Regression and Classification algorithms is that Regression algorithms are used to predict continuous values, such as price, salary, age, etc., and Classification algorithms are used to predict/classify the discrete values, such as Male or Female, True or False, Spam or Not Spam, etc.



Data Matrix

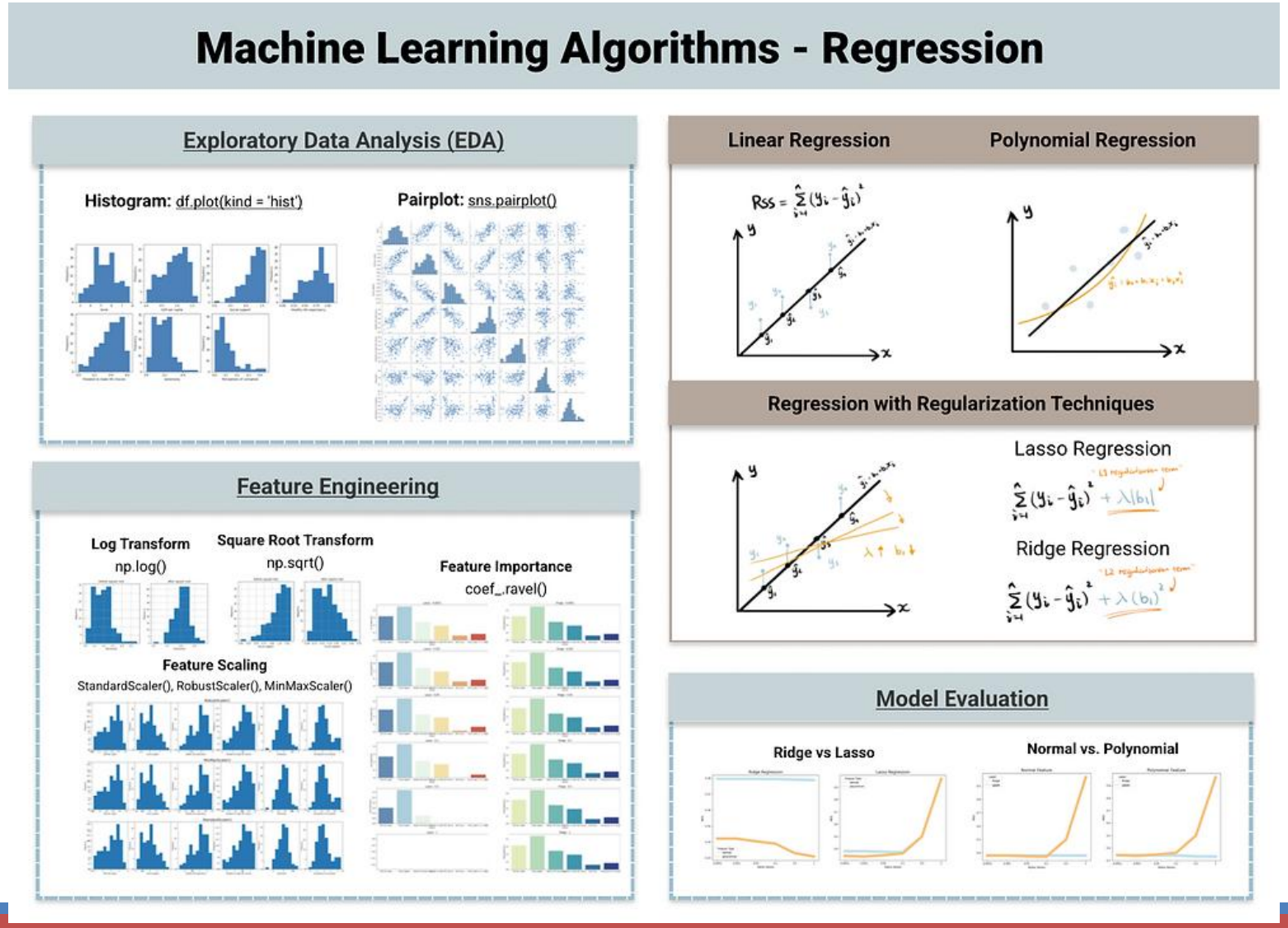
matrix X (mxn)	feature x_1	feature x_2	.....	feature x_n	Target Y
Example 1	$x_1^{(1)}$	$x_2^{(1)}$		$x_n^{(1)}$	$Y^{(1)}$
Example 2	$x_1^{(2)}$	$x_2^{(2)}$		$x_n^{(2)}$	$Y^{(2)}$
…					
Example i	$x_1^{(i)}$	$x_2^{(i)}$		$x_n^{(i)}$	$Y^{(i)}$
…					
…					
Example m	$x_1^{(m)}$	$x_2^{(m)}$		$x_n^{(m)}$	$Y^{(m)}$

x – input vector of features, attributes
y – output vector of labels, ground truth, target
m - number of training examples
n – number of features
$h_\theta(x)$ - model (hypothesis)
θ - vector of model parameters

Training set: data matrix X (m rows, n columns)

Supervised Learning
Regression

Regression


Machine Learning Algorithms - Regression
Exploratory Data Analysis (EDA)
Histogram: df.plot(kind = 'hist')
Pairplot: sns.pairplot()
Feature Engineering
Log Transform
np.log()
Square Root Transform
np.sqrt()
Feature Importance
coef_.ravel()
Feature Scaling
StandardScaler(), RobustScaler(), MinMaxScaler()
Linear Regression
Polynomial Regression
Regression with Regularization Techniques
Lasso Regression
Ridge Regression
Model Evaluation
Ridge vs Lasso
Normal vs. Polynomial

Regression

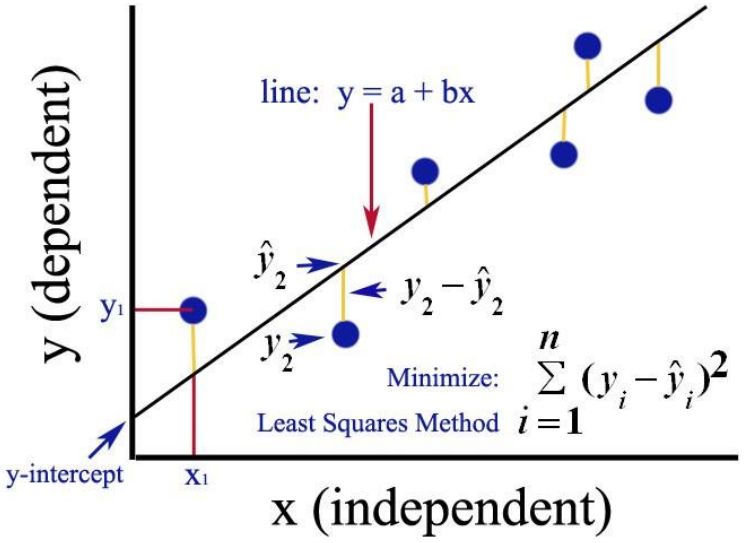
The objective of performing a regression is to build a model to express the relation between the response $\mathbf{y} \in \mathbb{R}^n$ and a combination of one or more (independent) variables $\mathbf{x}_i \in \mathbb{R}^n$. [1] The model allows us to predict the response $\mathbf{y}$ from the variables. The simplest model which can be considered is a *linear model*, where the response $\mathbf{y}$ depends linearly on the d variables $\mathbf{x}_i$:

$$\mathbf{y} = a_1\mathbf{x}_1 + \cdots + a_d\mathbf{x}_d. \tag{6.1}$$

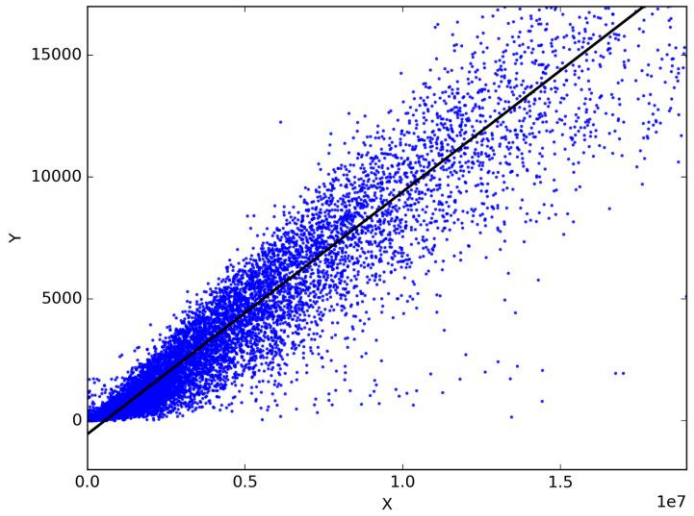
The variables a_i are termed the *parameters* or *coefficients* of the model. This equation can be rewritten in a more compact matrix form: $\mathbf{y} = \mathbf{X}\mathbf{w}$, where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \mathbf{X} = \begin{pmatrix} x_{11} \ldots x_{1d} \\ x_{21} \ldots x_{2d} \\ \vdots \\ x_{n1} \ldots x_{nd} \end{pmatrix}, \mathbf{w} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_d \end{pmatrix}.$$

Linear regression is the technique for creating these linear models.



Regression



Regression is a supervised machine learning algorithm and statistical method used to model the relationship between a dependent variable (target) and one or more independent variables (predictors) by fitting a linear equation to observed data.

Applications:

Prediction: Forecasting trends, such as future sales or stock prices.
Inference: Determining the strength of the relationship between variables.
Examples: Predicting exam scores based on hours studied or fuel efficiency based on car weight.

Regression is an important tool for data science because:

1. Lets you identify relationships between variables
2. Make predictions based on those relationships
3. Assess how well a given model fits the data

Regression - Assumptions

1. Linearity: The relationship between inputs (X) and the output (Y) is a straight line.

2. Independence of Errors: The errors in predictions should not affect each other.

3. Constant Variance (Homoscedasticity): The errors should have equal spread across all values of the input. If the spread changes (like fans out or shrinks), it's called heteroscedasticity and it's a problem for the model.

4. Normality of Errors: The errors should follow a normal (bell-shaped) distribution.

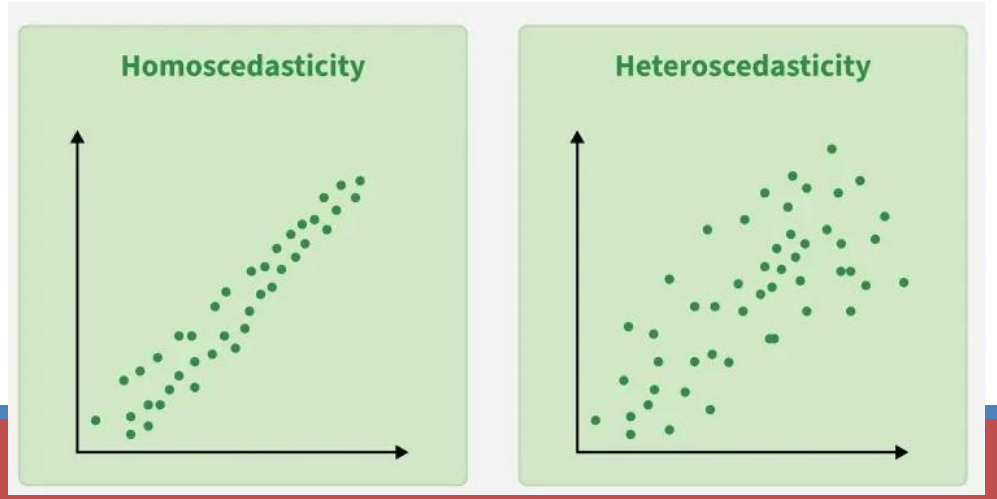
5. No Multicollinearity (for multiple regression): Input variables shouldn’t be too closely related to each other.

6. No Autocorrelation: Errors shouldn't show repeating patterns, especially in time-based data.

7. Additivity: The total effect on Y is just the sum of effects from each X, no mixing or interaction between them.'

In linear regression some hypothesis are made to ensure reliability of the model's results.

Assumes Linearity: The method assumes the relationship between the variables is linear. If the relationship is non-linear, linear regression might not work well.
Sensitivity to Outliers: Outliers can significantly affect the slope and intercept, skewing the best-fit line.

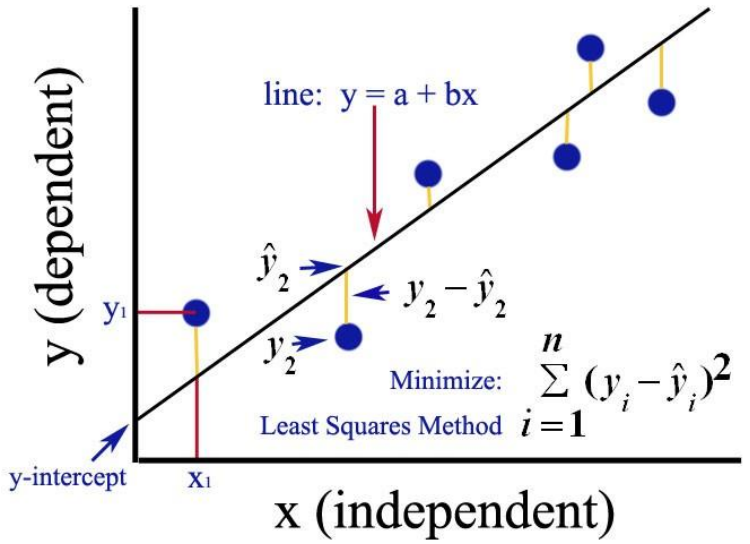


Univariate Regression

Problem: Learning to predict the housing prices (output, predicted variable) as a function of the living area (input, feature, predictor)

observation – prediction = Error (“residual”)

Living area ($feet^2$)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



Univariate Regression

$$h_\theta(x) = \theta_0 + \theta_1 x_1 = \begin{bmatrix} 1 & x_1 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \vec{x}^T \vec{\theta}$$

=> in Python => np.dot(X, Theta)

$$\vec{x} = \begin{bmatrix} x_0 = 1 \\ x_1 \end{bmatrix}$$

Components and Interpretation:

Dependent Variable (Y): The variable being predicted (e.g., house price).

Independent Variable (X): The predictor variable (e.g., size of house).

Slope (θ_0): Represents the change in the dependent variable for a one-unit increase in the independent variable.

Intercept (θ_1): The value of (Y) when all (X) variables are zero.

Univariate Regression

$$h_\theta(x) = \theta_0 + \theta_1 x_1 = \begin{bmatrix} 1 & x_1 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \vec{x}^T \vec{\theta}$$

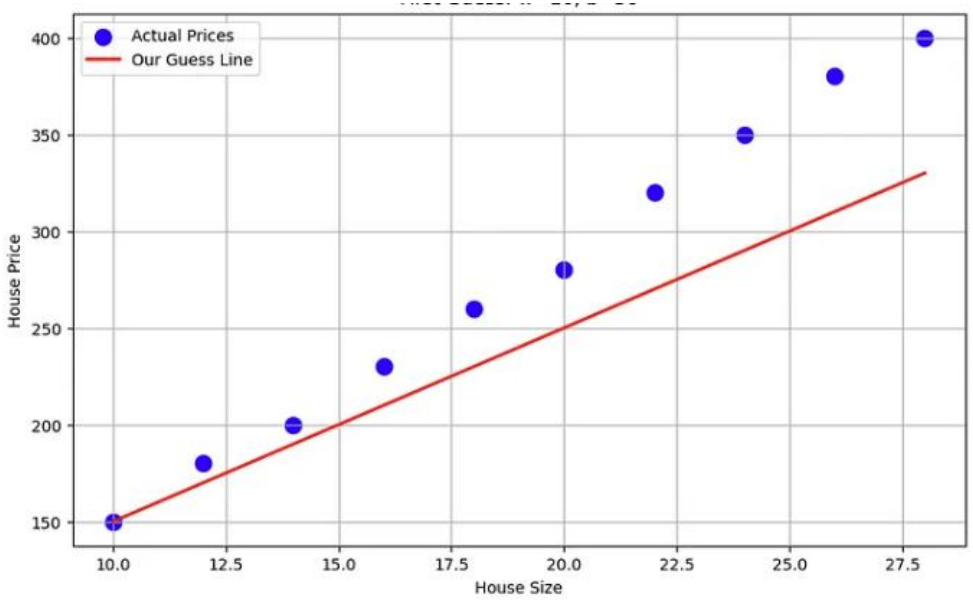
=> in Python => np.dot(X, Theta)

$$\vec{x} = \begin{bmatrix} x_0 = 1 \\ x_1 \end{bmatrix}$$

Why?

matrix X (mxn)	X0 (EXTRA)	feature x_1	feature x_2	.....	feature x_n	Target Y
Example 1	1	$x_1^{(1)}$	$x_2^{(1)}$		$x_n^{(1)}$	$Y^{(1)}$
Example 2	1	$x_1^{(2)}$	$x_2^{(2)}$		$x_n^{(2)}$	$Y^{(2)}$
...						
Example i	1	$x_1^{(i)}$	$x_2^{(i)}$		$x_n^{(i)}$	$Y^{(i)}$
...						
...						
Example m	1	$x_1^{(m)}$	$x_2^{(m)}$		$x_n^{(m)}$	$Y^{(m)}$

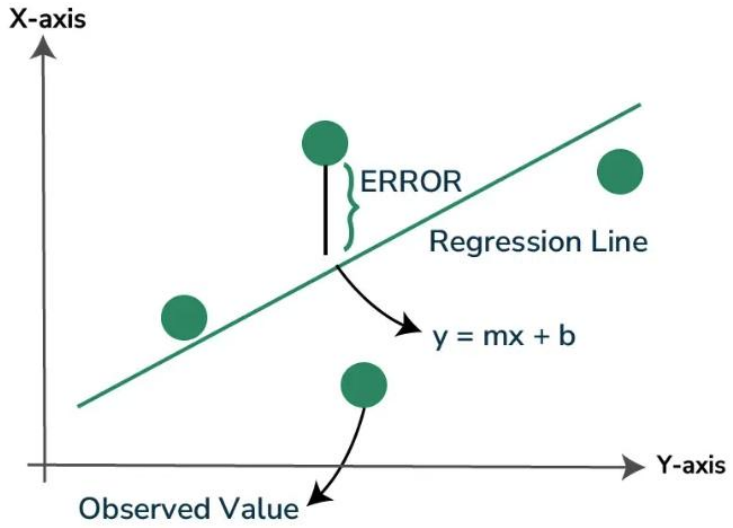
Univariate Regression



How good is this model?

How can I evaluate it?

Univariate Regression – Mean Squared Error



Linear Model (hypothesis)

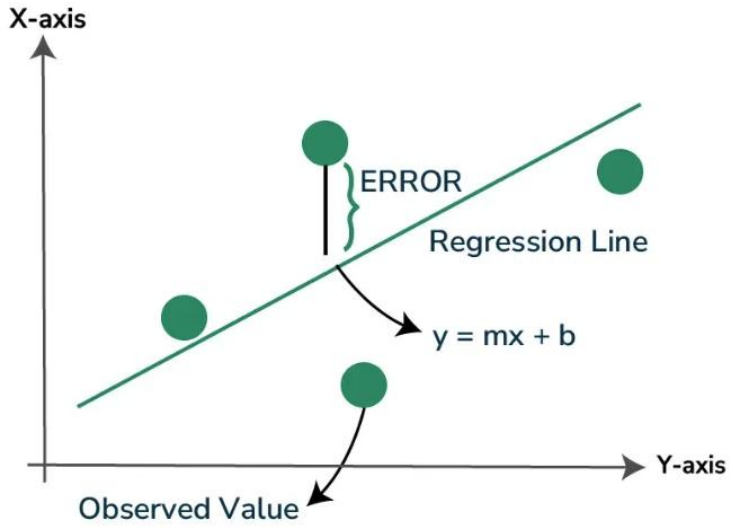
$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Cost (loss) function (Mean Squared Error)

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

m – number of training examples

Univariate Regression – Cost function



Linear Model (hypothesis)

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Cost (loss) function (Mean Squared Error)

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

m – number of training examples

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

Gradient Descent

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

Cost function

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

Cost function gradients
(parcial derivatives of *J* with respect to θ)

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y \right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \left(h_\theta(x) - y \right) x_j
\end{aligned}$$

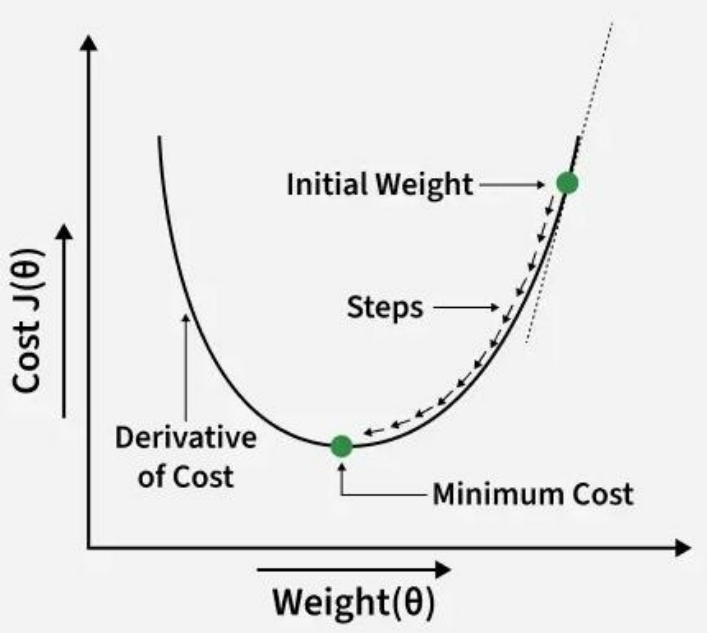
Cost function gradients
(generalization)

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

Gradient Descent

Gradient descent is an optimization technique used to train a linear regression model by minimizing the prediction error.

It works by starting with random model parameters and repeatedly adjusting them to reduce the difference between predicted and actual values.



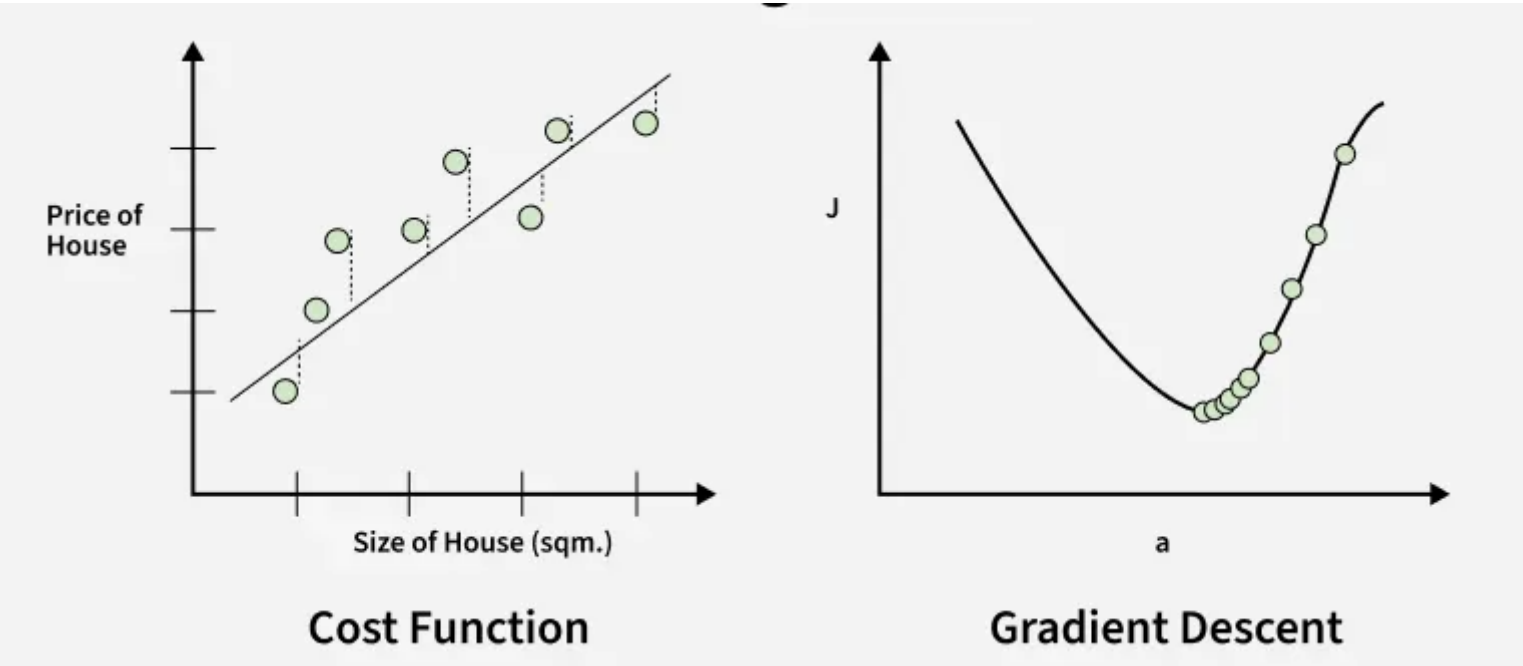
How it works:

1. Start with random values for slope and intercept.
2. Calculate the error between predicted and actual values.
3. Find how much each parameter contributes to the error (gradient).
4. Update the parameters in the direction that reduces the error.
5. Repeat until the error is as small as possible.

Gradient Descent

Gradient descent is an optimization technique used to train a linear regression model by minimizing the prediction error.

It works by starting with random model parameters and repeatedly adjusting them to reduce the difference between predicted and actual values.



Gradient Descent

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

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

Cost function gradients (generalization) $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

Update model parameters /weights $\theta_j := \theta_j - \alpha\frac{\partial}{\partial \theta_j}J(\theta)$

Learning rate

Learning Rate

Update model parameters /weights

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

Learning rate

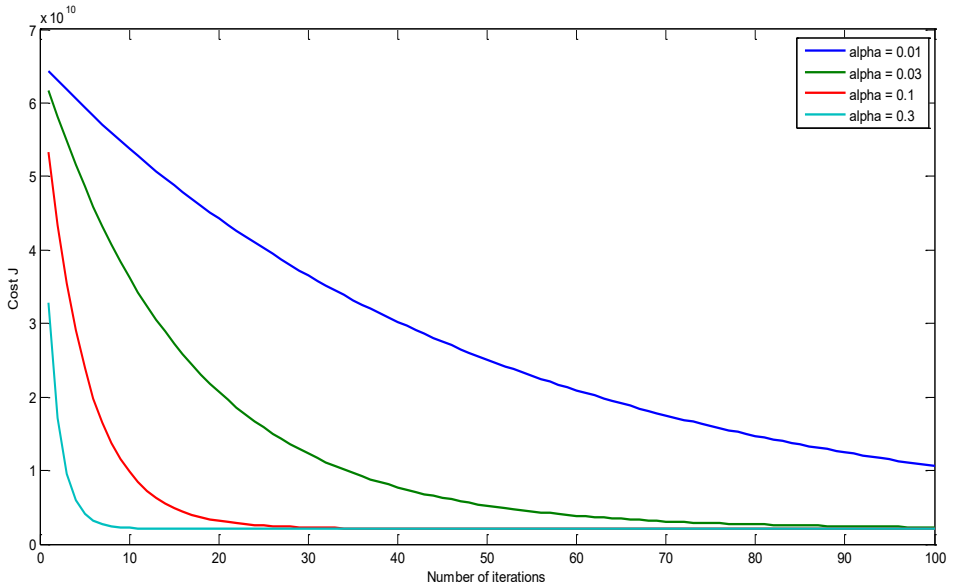
The **learning rate** is a **scalar** that multiplies the gradient of the objective function concerning the model parameters. It **influences** the **size** of the **steps** taken during optimization.

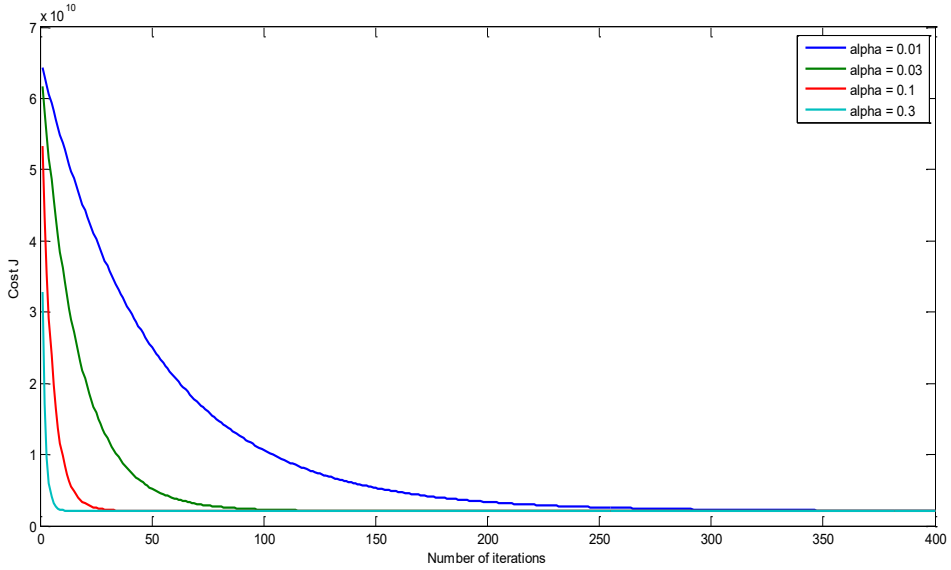
a. Too High Learning Rate: the algorithm may overshoot the minimum and fail to converge.

b. Too Low Learning Rate : the algorithm may take tiny steps and might converge very slowly, or it might get stuck in a local minimum.

Learning Rate

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

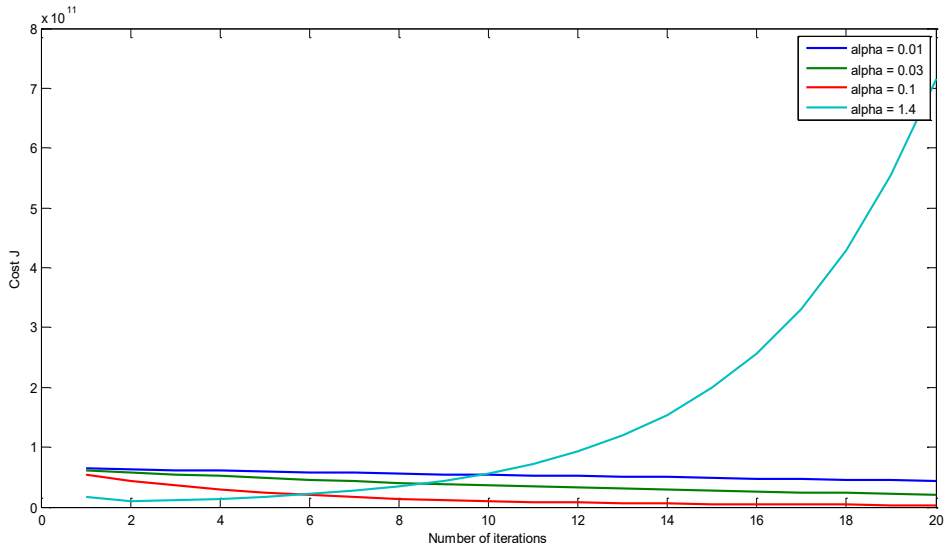




If a too **small** : **slow** convergence of the cost function J
(Gradient Descent optimization is slow)

Learning Rate

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



If a too **large**: the cost function J may not converge.
It may **diverge** !

Parameter Update

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

Batch learning (classical approach):

- update parameters after all training examples have been processed, repeat several iterations until convergence
- take a single step/update, we must go through the whole training set
- is expensive for large data set

Mini batch learning (if big training data):

- divide training data into small batches
- update parameters after each mini batch has been processed
- repeat until convergence

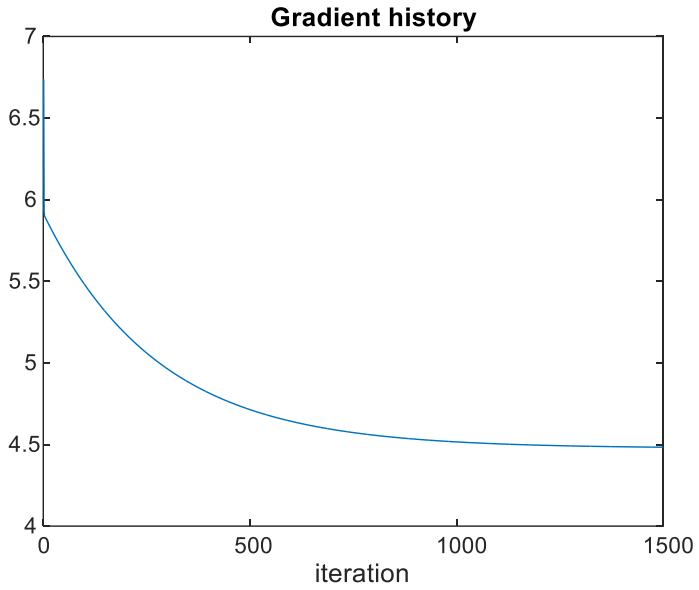
Stochastic (incremental) **learning** (large-scale ML problems):

- update parameters after every single training example has been processed

Stochastic Gradient Descent (SGD):

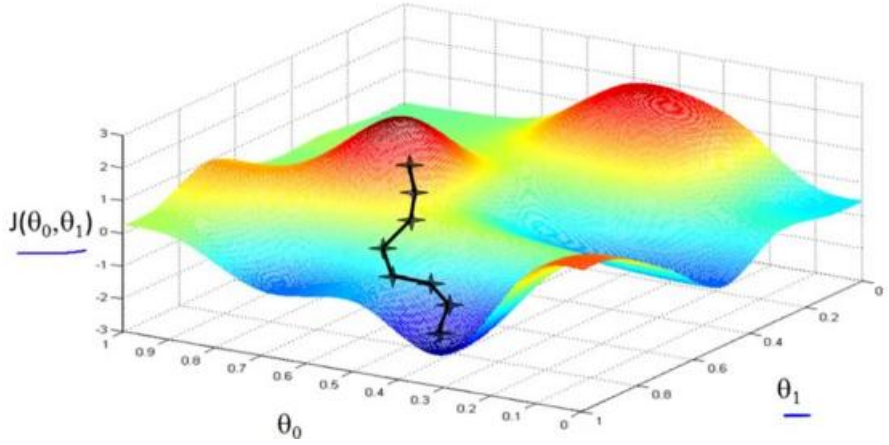
- for every training example, we take a single step
- the true gradient is approximated by a gradient at a single example. Adaptive learning rate.
- Implies a slightly noisy/random path towards global minima

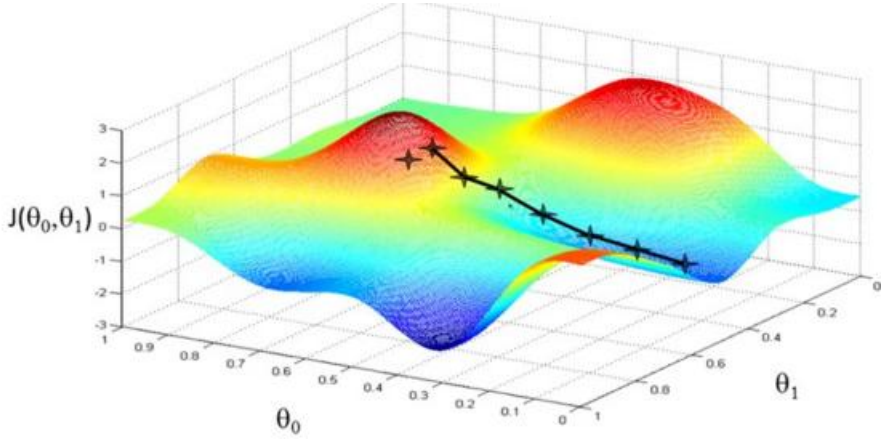
Convergence



Linear Regression (LinReg):

starting from different initial values of ϑ the cost function J should always converge (<u>maybe to a local minimum !!!</u>) if LinReg works properly.





Local minimum

Suppose θ_1 is at a local min. as shown in the figure.

What will do Gradient Descent algorithm at the next step ?

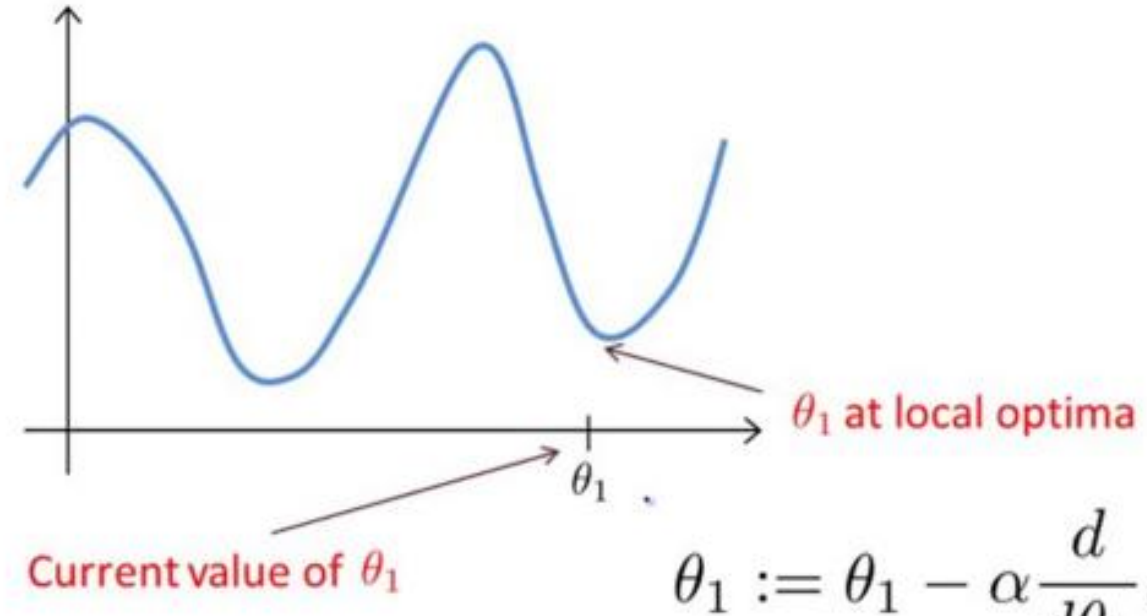
1) Leave θ_1 unchanged
2) Change θ_1 in a random direction
3) Decrease θ_1
4) Move θ_1 in direction to the global minimum of J

Some statistical facts about local minima:

•Local minima are common: In most cases, there are many local minima in the loss landscape.

•Local minima can be very different: A model that is in a local minimum may not be very accurate.

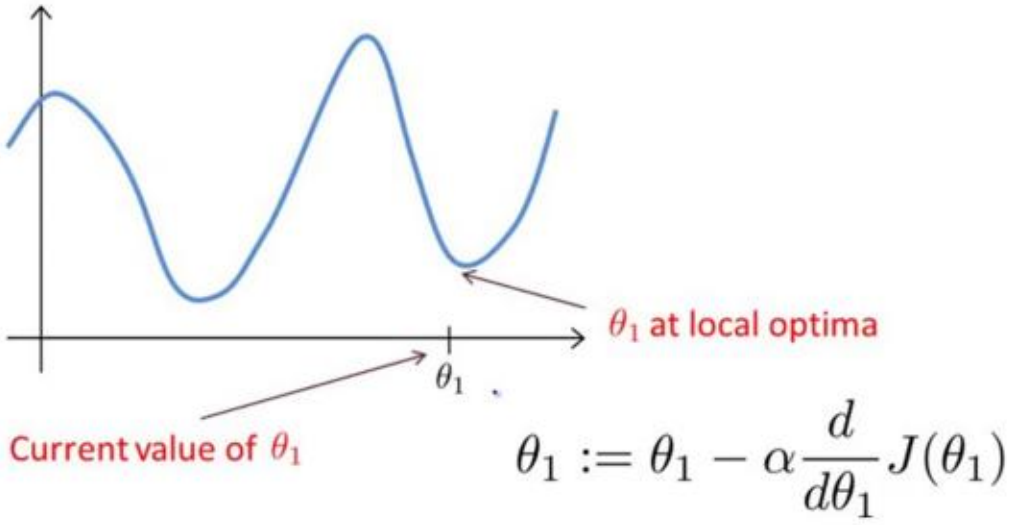
•It is difficult to find the global minimum: Finding the global minimum is a very difficult problem.



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

Local minimum

How does gradient descent deal with local minima?
Gradient descent is driven by the gradient, which will be zero at the base of any minima. Local minimum are called so since the value of the loss function is minimum at that point in a local region.



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

How to know if a model is stuck in a local minimum?

- ✓ If the loss function is not decreasing after a certain number of iterations.
- ✓ If the model parameters are not changing after a certain number of iterations.

If you think that your model is stuck in a local minimum, you can try one of the following:

- **Change the learning rate:** A smaller learning rate may help the model to escape from the local minimum.
- **Use a different optimization algorithm:** A different optimization algorithm, such as SGD or momentum, may be more effective at avoiding local minima.
- **Add regularization:** Regularization can help to prevent the model from overfitting the training data and can make it less likely to get stuck in a local minimum.

Regression Model Evaluation Metrics

Root Mean Squared Error

Square root of MSE, in the same unit as the target variable. Highlights larger errors.

R-Squared

Explains the proportion of variance in the dependent variable captured by the model.

Mean Absolute Error

Measures average absolute differences between actual and predicted values. Less sensitive to outliers.

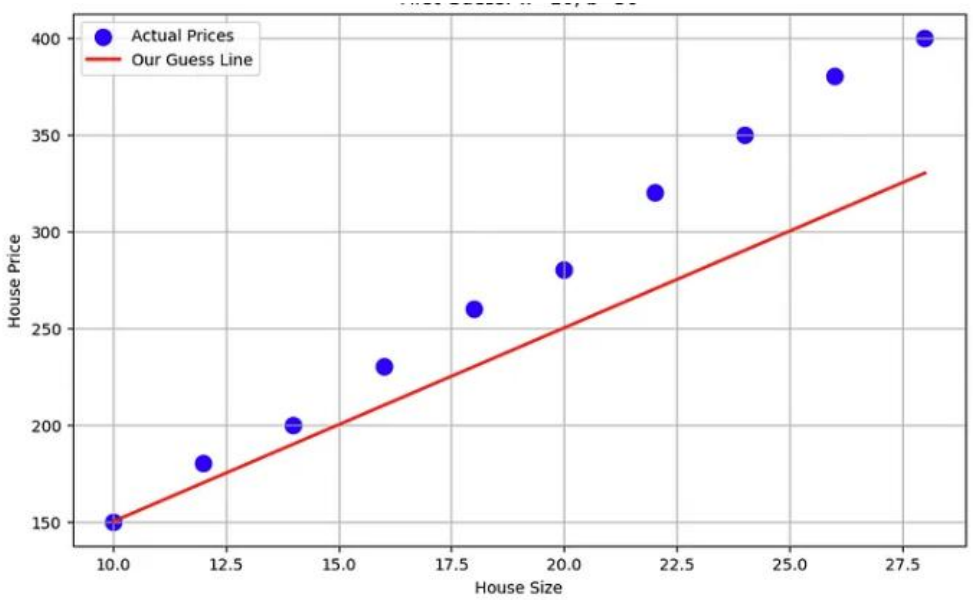
Adjusted R-Squared

R-Squared adjusted for the number of predictors, preventing overfitting.

Mean Squared Error

Measures average squared differences between actual and predicted values. Sensitive to outliers.

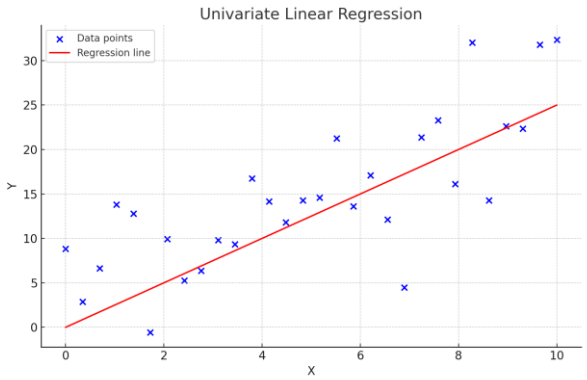
Regression

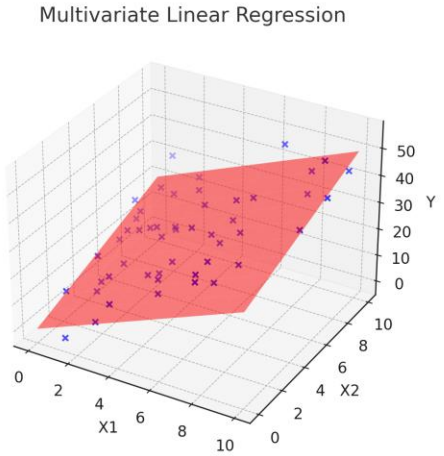


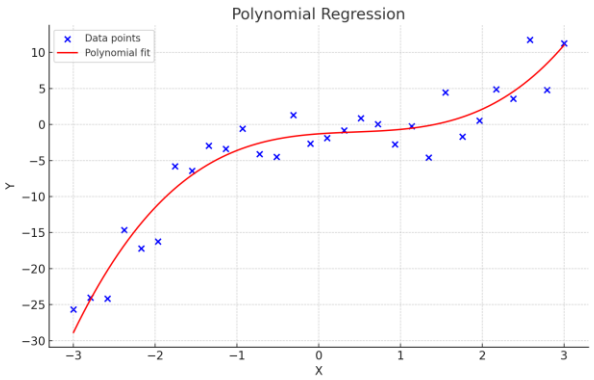
$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

And when one independent variable is not enough to model your data?

Regression


Univariate Linear Regression
Data points
Regression line
X
Y


Multivariate Linear Regression
X1
X2
Y


Polynomial Regression
Data points
Polynomial fit
X
Y

Multivariate Regression

Multivariate Regression

Problem: Learning to predict the housing price as a function of living area & number of bedrooms.

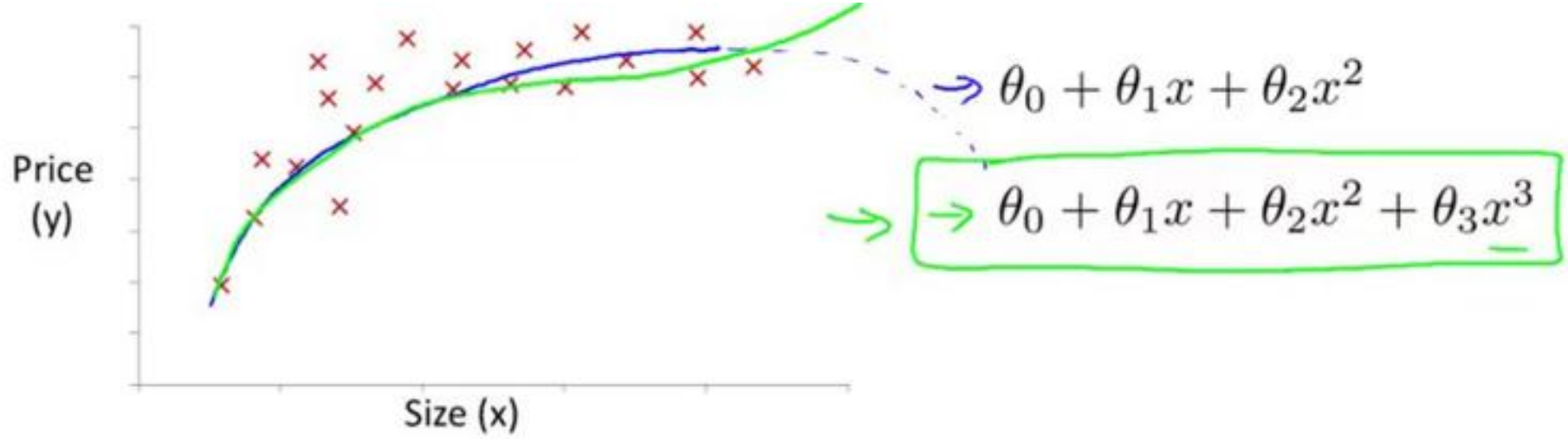
Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = [\theta_0 \quad \theta_1 \quad \theta_2] \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \end{bmatrix} = \vec{\theta}^T \vec{x}$$

Polynomial Regression

If the univariate linear regression model is not a good model, you may also try a polynomial model.

Univariate (x1=size) housing price problem transformed into a multivariate (still linear !!!) regression model
x=[x1=size, x2=size^2, x3=size^3]



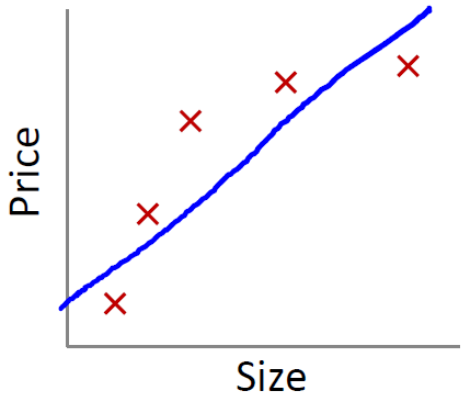
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

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

Overfitting

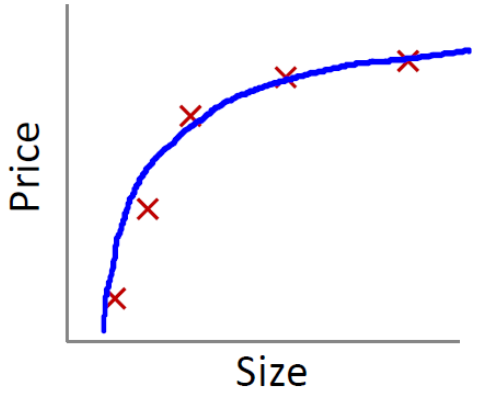
Overfitting

Overfitting: If we have too many features (e.g. high order polynomial model), the model may fit the training data too well but fail to generalize to new examples (e.g. predict prices on new examples).



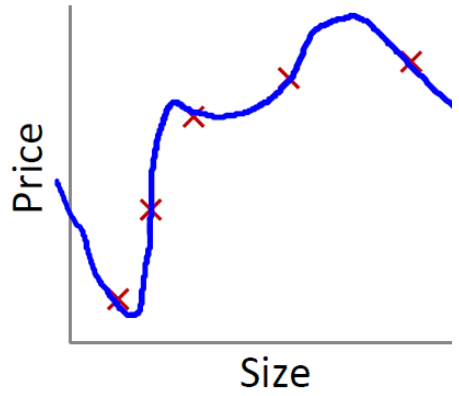
underfit
(1st order polin. model)

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



just right
(3rd order polinom. model)

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



overfit
(higher ord. polinom. Model)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_{16} x^n$$

Overfitting

Overfitting: If we have too many features (x1,…x100) the model may fit the training data too well but fails to generalize to new examples.

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \vec{\theta}^T \vec{x}$$

How to deal with overfitting?

1. **Reduce** number of features.
    1. Manually select which features to keep.
    2. Algorithm to select the best model complexity.

2. **Regularization** (add extra term in the cost function)

Regularization methods shrink model parameters ϑ towards zero to prevent overfitting by reducing the variance of the model.

    1. **Ridge** Regression (L2 norm):
        1. Reduce magnitude of θ (but never make them =0) => keep all features
        2. Works well when all features contributes a bit to the output y.
    2. **Lasso** Regression (L1 norm):
        1. May shrink some of the elements of vector ϑ to become = 0.
        2. Eliminate some of the features => Serve as feature selection

Regularization works by adding a penalty to the loss function that is proportional to the size of the model parameters. This helps to prevent the model from overfitting the training data, and can make it less likely to get stuck in a local minimum.

Regularization

Ridge Regression

Our Goal

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

Unregularized cost function

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

Regularized cost function

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

(add extra regularization term don't regularize θ_0)

Ridge Regression

Regularization

Ridge Regression

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

Unregularized cost function **gradients**

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

Regularized cost function **gradients**

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \quad \text{for } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \quad \text{for } j \geq 1$$

Regularized Linear Regression

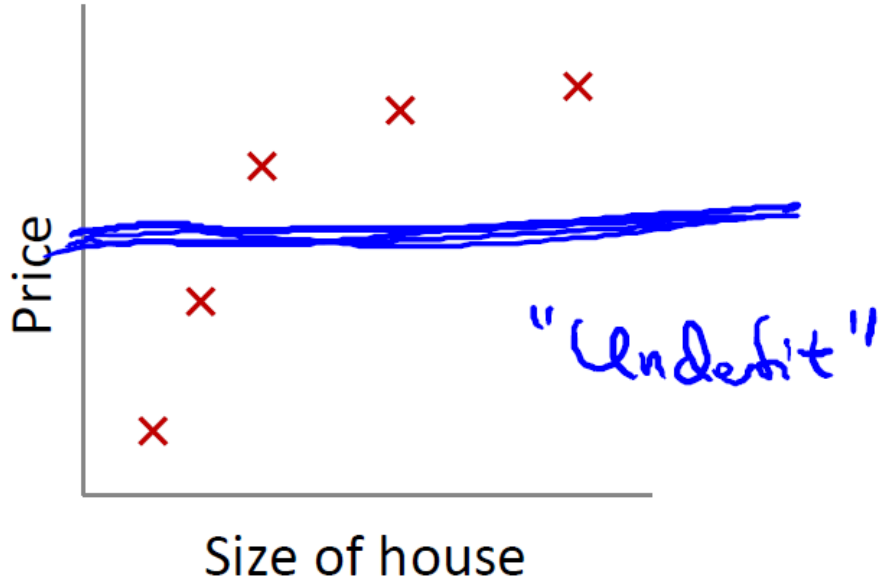
What if lambda is set to an extremely large value ?

- The overfitting will not be overcome.
- The model will be underfitted (fails to fit even training data).
- Gradient descent will fail to converge.

Regularized Linear Regression

What if lambda is set to an extremely large value ?

- The overfitting will not be overcome.
- The model will be underfitted (fails to fit even training data).
- Gradient descent will fail to converge.



Regularization

Lasso Regression

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

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)-\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]+\frac{\lambda}{2m}\sum_{j=1}^{n}\left|\theta_j\right|$$

Ridge Regression shrinks ϑ towards zero, but never equal to zero => all features are kept in the model.

Lasso Regression may get some ϑ to become exactly zero => reduces the number of features, serves as feature selection method.

Lasso Regression involves absolute values (not differentiable)=> computing is challenging

Lasso regression is preferred when the goal is feature selection, resulting in a simpler and more interpretable model with fewer variables.

Ridge regression tends to favor a model with a higher number of parameters, as it shrinks less important coefficients but keeps them in the model.

Regularization: Final Notes

Lasso vs. Ridge

Ridge regularization:
- produce more complex models with better prediction power
- may suffer from overfitting due to its reliance on all available features

Lasso regularization:
- reduce the number of features used in a model and eliminate noisy ones
- produce simpler models that are more likely to generalize.

Lasso and Ridge regularization can both be used together.

The combination is known as **elastic net regularization** and can produce simpler models while still utilizing most or all of the available features.

→ Popular in machine learning due to its capability to improve prediction accuracy while minimizing overfitting.

Important Note:
Consider which technique is more suitable for a given problem, some scenarios require specific approach.

Regularization: Final Notes

Lasso and Ridge Regression are two of the most popular techniques for regularizing linear models, which often yield more accurate predictions than traditional linear models. These methods reduce the model's complexity by introducing shrinkage or adding a penalty to complex coefficients.

•The Ridge-Lasso approach is limited in requiring the input features to be standardized before fitting the model. It means that any feature with a large range of values can bias results because of its scale relative to other features with smaller ranges.

•Furthermore, if the data points contain outliers or noise, then this could produce inaccurate predictions due to the penalty terms.

•Additionally, Ridge and Lasso Regressions can be slow when applied to large datasets because of the computation time needed to perform regularization.

•Lastly, these methods require careful selection of hyperparameters (i.e., regularization strength), which can induce further computational costs and time.

Regularization: Final Notes

Lasso and Ridge Regression are two of the most popular techniques for regularizing linear models, which often yield more accurate predictions than traditional linear models. These methods reduce the model's complexity by introducing shrinkage or adding a penalty to complex coefficients.

- The Ridge-Lasso approach is limited in requiring the input features to be standardized before fitting the model. It means that any feature with a large range of values can bias results because of its scale relative to other features with smaller ranges.

- Furthermore, if the data points contain outliers or noise, then this could produce inaccurate predictions due to the penalty terms.

- Additionally, Ridge and Lasso Regressions can be slow when applied to large datasets because of the computation time needed to perform regularization.

- Lastly, these methods require careful selection of hyperparameters (i.e., regularization strength), which can induce further computational costs and time.

Careful data study, description and visualization?

Regularization: Final Notes

Lasso and Ridge Regression are two of the most popular techniques for regularizing linear models, which often yield more accurate predictions than traditional linear models. These methods reduce the model's complexity by introducing shrinkage or adding a penalty to complex coefficients.

- The Ridge-Lasso approach is limited in requiring the input features to be standardized before fitting the model. It means that any feature with a large range of values can bias results because of its scale relative to other features with smaller ranges.

- Furthermore, if the data points contain outliers or noise, then this could produce inaccurate predictions due to the penalty terms.

- Additionally, Ridge and Lasso Regressions can be slow when applied to large datasets because of the computation time needed to perform regularization.

- Lastly, these methods require careful selection of hyperparameters (i.e., regularization strength), which can induce further computational costs and time.

Which scaling technique?