AMLLS Course Notes

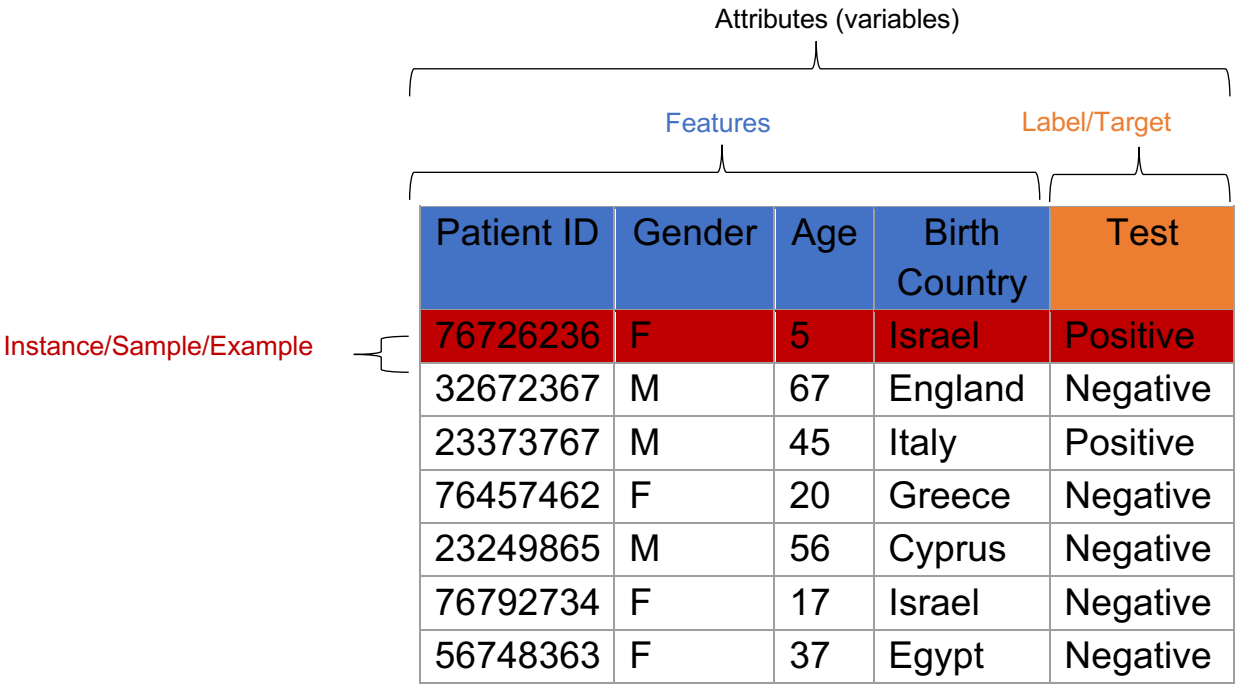
Applied Machine Learning for Life Sciences: 20243531

Feinberg Graduate School,
Weizmann Institute of Science
Winter Semester 2023/24

Ortal Dayan, ort.dayan@gmail.com

Introduction To Machine Learning

The Dataset Attributes and Instances



Patient ID	Gender	Age	Birth Country	Test
76726236	F	5	Israel	Positive
32672367	M	67	England	Negative
23373767	M	45	Italy	Positive
76457462	F	20	Greece	Negative
23249865	M	56	Cyprus	Negative
76792734	F	17	Israel	Negative
56748363	F	37	Egypt	Negative

ML Algorithms Can Be Inspected To See What They have Learned

E.g., using *feature importance ranking*.

Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable by the ML model.
These can assist us in performing feature selection.

E.g., using *feature importance ranking*.
For instance, with linear models the coefficients can be used to determine feature importance.

$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Supervised vs. Unsupervised Algorithms

Examples

Classification: predicting medical test result given a set of features age, sex, place of birth, etc. and the test result (positive/negative) per patient.

Regression: predicting respiratory rate, given a set of features ECG, PPG etc. and the respiration rate per patient.

Clustering: a clustering algorithm applied on the two mRNA gene expressions CTNNB1 and NOTCH1 assigns patients to one of two clusters.
Comparing the two clusters we observed that there is a statistically significant difference in overall survival in the patients between clusters.

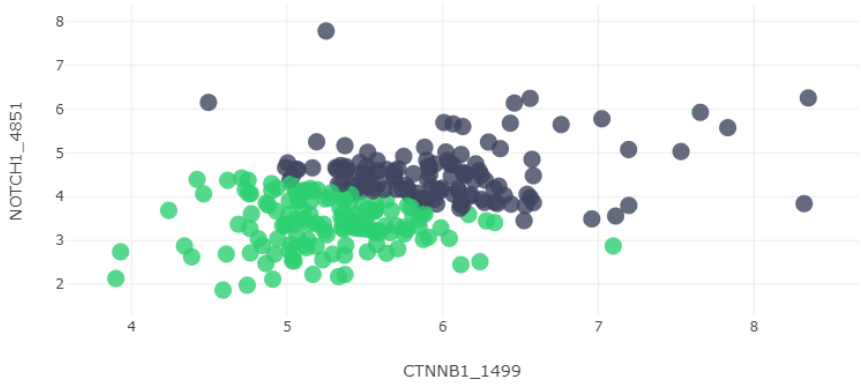


Image from: https://sonraianalytics.com/k-means-clustering-and-data-mining-in-precision-medicine/

Quick Review of Python Classes

Classes provide a means of bundling data (attributes) and functionality (methods) together.

A class describes the contents of the objects that belong to it:
it describes the data fields (called attributes) and defines the operations (called methods) that can be performed using the class data fields.

Object/instance is an element of a class; objects have the behaviors of their class. Initializing a class creates a new object/instance of the class with its unique instance variable values.

Every class has the self "keyword" which allows accessing the instance variables and methods of the class.

Note the difference between class variable and instance variable:

- Instance variables are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different.
- Unlike class variables (defined within the class construction) , instance variables are defined within methods.

Basic Class Example

```
class Greeter:

      # Class construction
      surname = "McCartney” # class variable

      # Constructor method
      def __init__(self, name):
            self.name = name # Create an instance variable/initialize the class attrinbute name


      # Instance method
      def greet(self, loud=False):
             if loud: print(f'HELLO, {self.name.upper()}’)
             else: print(f'Hello, {self.name}’)

p = Greeter('Paul') # Construct an instance of the Greeter class (and initialize it)
print(p)
print(p.name)
p.surname
print(p.greet())

j = Greeter('Jesse’)
print(j.name)
p.surname
print(j.greet(loud=True)) # Call an instance method
```

<__main__.Greeter at 0x7fa2803a25d0>
'Paul’
'McCartney'
Hello, Paul
'Jesse’
'McCartney'
HELLO, JESSE

Scikit-Learn Design

All objects share a consistent and simple interface.

Estimators are object that can estimate some parameters based on a dataset.
Estimations are performed by the fit() method.
E.g., the Linear_Regression class as an estimator:

```
lin_reg = Linear_Regression()
lin_reg.fit(X, y)
```

All the estimator's learned parameters are accessible via public instance variables with an underscore suffix.
E.g., t0, t1 = lin1.intercept[0], lin1.coef_[0][0]

Predictors are estimators that can make predictions using a dataset.
Predictions are performed by the predictor method.
E.g., the Linear_Regression class as a predictor: lin_reg.predict(x_new)

Data Centric Approach

Model-centric (dominant): hold the data fixed and iteratively improve the model.
Data-centric: hold the model and iteratively improve the quality of the data.

Based on: https://www.forbes.com/sites/gilpress/2021/06/16/andrew-ng-launches-a-campaign-for-data-centric-ai/?sh=7840177674f5

The Difference Between A Learning Algorithm and A Model

E.g., Linear Regression

Model:

- Model Data: Vector of parameters θ_0 and θ_1
- Prediction Algorithm: sums the product of the model parameters with the input $y = \theta_0 + \theta_1 x$

Algorithm: Finds set of parameters of the model θ_0 and θ_1 that minimize error (cost function) on training dataset (e.g., using gradient descent).

The algorithm's parameters are called hyperparameters.

E.g., The amount of regularization α to apply during training is a Hyperparameter. $cost = \frac{\sum_1^n (\hat{y}-y)^2}{n} + \alpha\,|\theta_1|$

A hyperparameter is a parameter (setting) of a *learning algorithm* (not of the model). It must be set prior to training (using a separate validation set) and remains constant during training.

E.g.,

- The number of nearest neighbors K in the K Nearest Neighbors algorithm
- The train and test sets' sizes
- The number of folds in cross validation

Based on: https://machinelearningmastery.com/difference-between-algorithm-and-%20model-in-machine-learning/

Exploratory Data Analysis (EDA)

EDA is an approach to analyzing datasets to summarize their main characteristics.

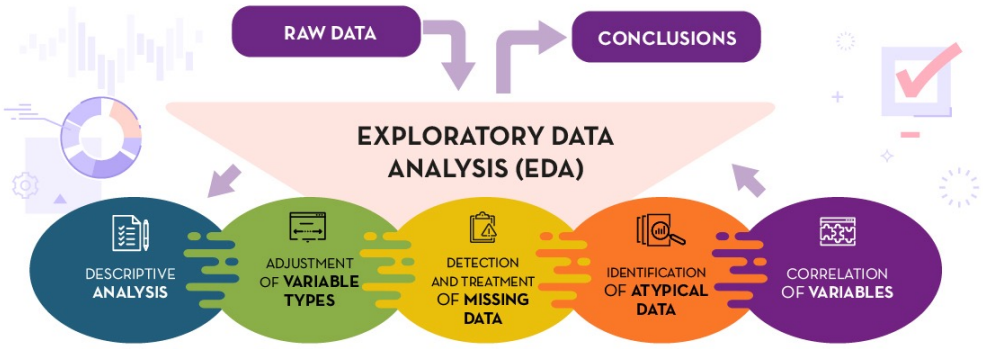


Image from: https://datos.gob.es/en/documentacion/practical-introductory-guide-exploratory-data-analysis

Seaborn Library

Seaborn is a library often used to make default **matplotlib** plots look nicer, and it also introduces some additional plot types.

Linear Regression Models

Linear Regression Models

In statistics, a regression model is linear when all terms in the model are one of the following:

• A constant

• A parameter multiplied by an independent variable

Then, you build the equation by adding the terms together.

These rules limit the form to just one type:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Based on: https://statisticsbyjim.com/regression/difference-between-linear-nonlinear-regression-models/

Linear Regression

When the dataset is 1D, we fit a line to the training data.

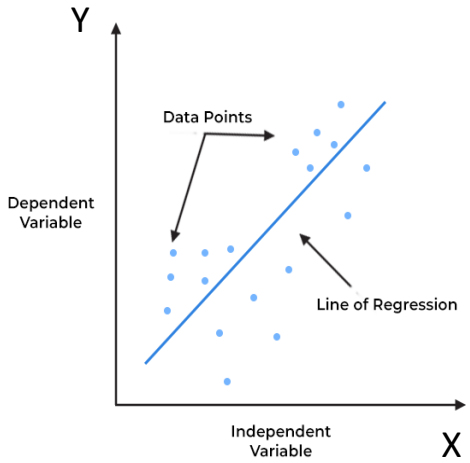
$\hat{y} = \theta_0 + \theta_1 x$

$\hat{y}$ is the predicted value

x is the only feature

θ_1 is the slope term

θ_0 is the bias term



Based on: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-linear-regression/

Multiple Regression

Also named Multivariate Linear Regression.

When we the dataset is 2D or above, we fit a multi dimensional object (with same number of dimensions as the features) to our training data.

$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

$\hat{y}$ is the predicted value

n is the number of features

x_i is the i^{th} feature value

θ_i is the i^{th} feature weight

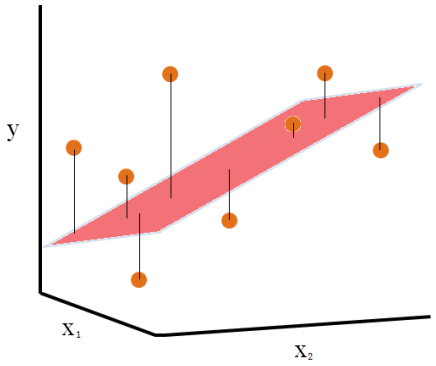


Image from: https://www.analyticsvidhya.com/blog/2021/11/startups-profit-prediction-using-multiple-linear-regression/

Polynomial Regression

We can use a linear model to fit a 1D nonlinear data as well.

Polynomial regression is a form of regression analysis in which the relationship between the independent variable *x* and the dependent variable *y* is modelled as an *n*th degree polynomial in *x*.

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 \cdots + \theta_n x^n$$

$\hat{y}$ is the predicted value

n is the degree of the polynomial

x is the only feature

θ_i is the i^{th} model parameter

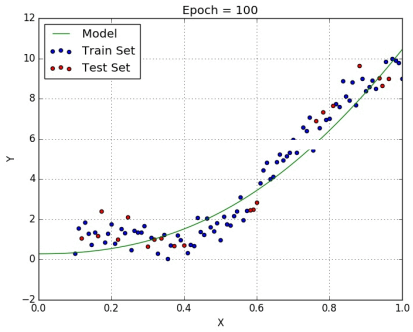


Image from: https://medium.com/@hrishavkmr/linear-and-polynomial-regression-1d666e25b016

Based on: https://en.wikipedia.org/wiki/Polynomial_regression

Polynomial Regression

Although it allows for a nonlinear relationship between y and x, polynomial regression is still considered linear regression since it follows the equation form for linear regression models.

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 \cdots + \theta_n x^n \quad \equiv \quad \hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

E.g., x^2 can be considered as equivalent to x_2 and x^n equivalent to x_n

Multiple Polynomial Regression

Also called Multivariate Polynomial Regression.

Allows for a nonlinear relationship between y and X where X is 2D or above.

E.g., a 2-degree multivariate polynomial regression for a 2D dataset:

$$\hat{y} = \theta_5 x_2^2 + \theta_4 x_1^2 + \theta_3 x_1 x_2 + \theta_2 x_2 + \theta_1 x_1 + \theta_0$$

$\hat{y}$ is the predicted value

n is the degree of the polynomial

x_1 and x_2 are the features

θ_i is the i^{th} model parameter

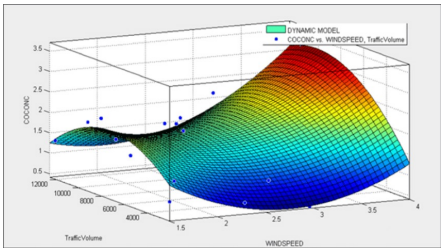


Image from: https://www.researchgate.net/figure/Multiple-polynomial-regression-based-surface-fitting-curve-of-CO-concentration-verses_fig3_333352304

Vectorization

Vectorized form is much more concise and used for programming:

$$h(\theta) = \theta^T X = [\theta_0 \quad \theta_1 \quad \cdots \quad \theta_n] \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \cdots & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_0^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(m)} \end{bmatrix} = \begin{bmatrix} \theta_0 x_0^{(1)} + \theta_1 x_1^{(1)} + \cdots + \theta_n x_n^{(1)} \\ \theta_0 x_0^{(2)} + \theta_1 x_1^{(2)} + \cdots + \theta_n x_n^{(2)} \\ \vdots \\ \theta_n x_n^{(m)} + \theta_n x_n^{(m)} + \cdots + \theta_n x_n^{(m)} \end{bmatrix}$$

$x_0\ terms = 1$

1st sample

Where:

- $h(\theta)$ is the linear/multiple/polynomial regression hypothesis function with the model parameters θ
- θ is the model's parameter vector, containing the bias term θ_0 and the feature weights θ_1 to θ_n
- X is the dataset matrix with the shape (n, m). Such that $x_j^{(i)}$ denotes a sample's feature

Read on matrix vector multiplication: https://mathinsight.org/matrix_vector_multiplication

Obtaining The Coefficients of the Hypothesis Function

Given $h_\theta(x)$ we search for the values of θ that give the best line fit (minimize residuals).

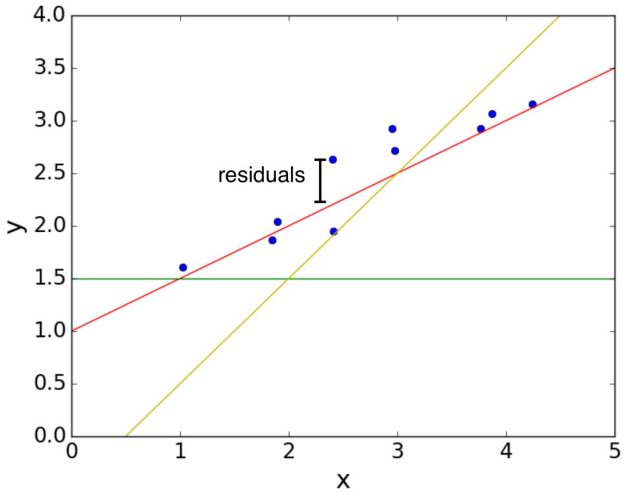


Image from: https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/readings/notes_on_linear_regression.pdf

In other words, we search for the values of θ that minimize the function that is the difference between the predicted values and the actual values.

We call this function the **objective function,** or **criterion** or **cost function, error function** or **loss.**

Note in some resources loss is defined as the cost for only one example.

Cost Functions

$\text{MAE}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left\|\theta^T x^{(i)} - y^{(i)}\right\|$	L1 Loss or Least Absolute Deviation	Assigns equal weight to all errors. The cost only increases linearly.
$\text{MSE}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T x^{(i)} - y^{(i)}\right)^2$	L2 Loss or Least Square Errors	More sensitive to outliers. Higher errors weigh more due to the nature of the power function.

Which Is Better, MAE or MSE?

MSE If you want to train a model which focuses on reducing large outlier errors.

MAE if you prefer greater interpretability (it is measured in the same units as the target).
E.g., if the dataset has many outliers.

RMSE is a compromise for sensitivity to outliers and interoperability.

$$\text{RMSE}(\theta) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2}$$

Based on: https://stephenallwright.com/mse-vs-mae/

The Normal Equation

For linear models, the precise value of $\boldsymbol{\theta}$ that minimizes the cost function can be found using a closed form solution by setting $\frac{\partial J(\theta)}{\partial \theta} = 0$

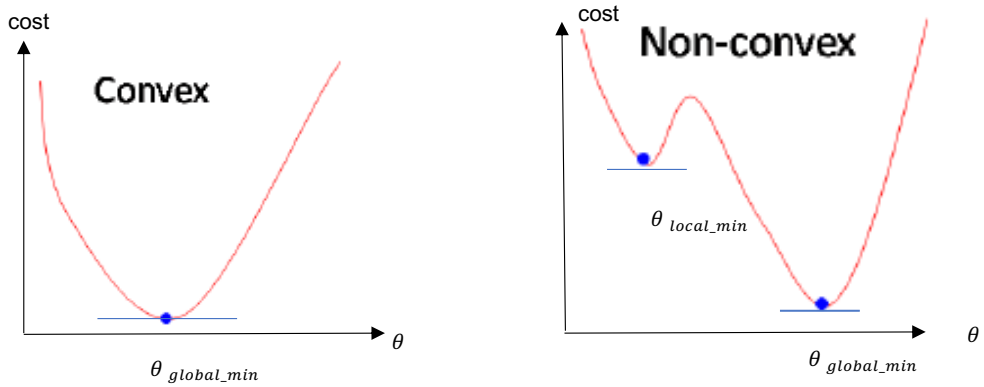


Image from: https://www.quora.com/Why-is-nonconvex-optimization-so-difficult-compared-to-convex-optimization

Cost Function Vectorized Form

It is Much More Concise and Used for Programming

$$\mathrm{MSE}(\theta) = \frac{1}{m}\left(\begin{bmatrix}\theta^T x^{(1)}\\ \theta^T x^{(2)}\\ \vdots \\ \theta^T m\end{bmatrix} - \begin{bmatrix}y^{(1)}\\ y^{(2)}\\ \vdots \\ y^{(m)}\end{bmatrix}\right)^2 = \left(\begin{bmatrix}\theta^T x^{(1)} - y^{(1)}\\ \theta^T x^{(2)} - y^{(2)}\\ \vdots \\ \theta^T x^{(m)} - y^{(m)}\end{bmatrix}\right)^2$$

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

$$\mathrm{MSE}(\theta) = \frac{1}{m}\left(\theta^T X - y\right)^2$$

$\frac{\partial J(\theta)}{\partial \theta} = \frac{2}{m}(\theta^T X - y)\mathrm{X}$ $/\frac{\partial\left(\frac{1}{m}(\theta^T X - y)^2\right)}{\partial \theta}$ *using the chain rule*

$\frac{2}{m}(\theta^T X - y)X = 0$ $/\frac{\partial J(\theta)}{\partial \theta} = 0$

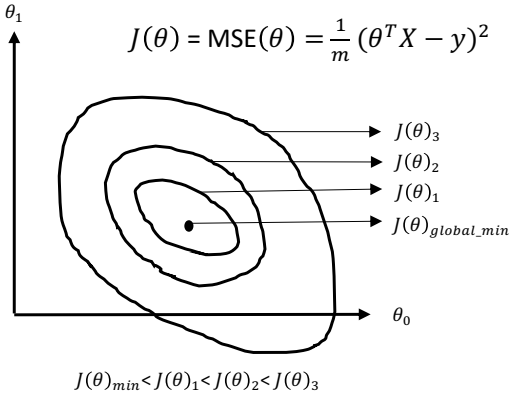
The Normal Equation: $\theta = (X^T X)^{-1} X^T y$ */ Isolating* θ

Read on the chain rule

Gradient Descent

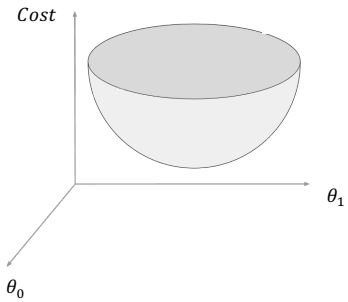
Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function by obtaining the direction of the steepest descent.

A common optimization methodology for many ML algorithms with nonconvex cost functions.



Contour plot

Image from: https://allmodelsarewrong.github.io/gradient.html

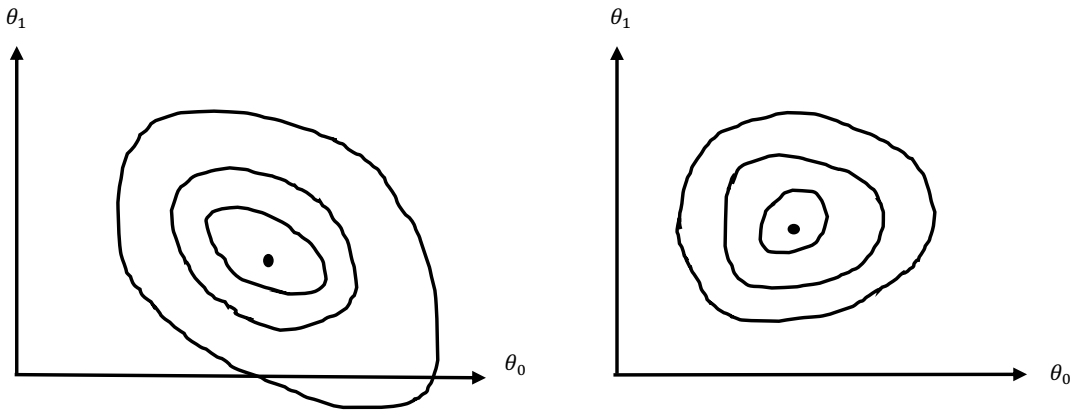


The cost function has the training set embedded in it.

E.g., $J(\theta)$ = MSE$(\theta) = \frac{1}{m}(\theta^T X - y)^2$

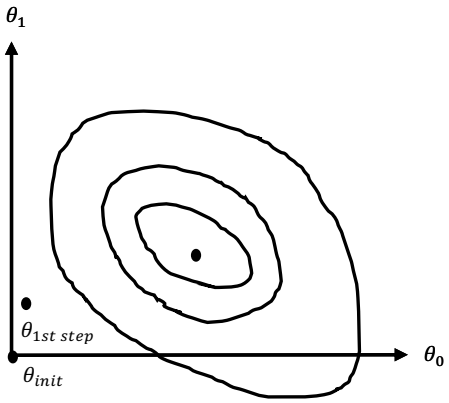
↓

The cost function shape depends on the training set (X, y).



The Optimization Process

Start with random θ values and then tweak them iteratively to minimize the cost function until convergence.



Tweaking the θ values is done by subtracting the gradient of the cost function with respect to the parameter θ.

$$\theta^{(next\ step)} = \theta - \eta\ \nabla_\theta J(\theta)$$

$$\nabla_\theta J(\theta) = \nabla_\theta \mathrm{MSE}(\theta) = \mathrm{X^T X\theta} - \mathrm{X^T y}$$

Each round of iteration through the entire dataset is called an epoch.

For functions with multiple inputs, we must make use of the concept of partial derivatives.

The partial derivative $\frac{\partial}{\partial\theta_j} f(x)$ measures how f changes as only the variable θ_j increases at point θ.

Obtaining the partial derivative of $\mathrm{MSE}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T x^{(i)} - y^{(i)}\right)^2$ with respect to θ_j:

$$\frac{\partial \mathrm{MSE}(\theta)}{\partial\theta_j} = \frac{2}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_j^{(i)}$$

$$\theta^T x^{(i)} = \theta_n x_n + \theta_j x_j + \cdots + \theta_1 x_1 + \theta_0$$

The gradient of f with respect to θ is the vector containing all the partial derivatives of f with respect to each of the θ elements, denoted $\nabla_\theta f(x)$.

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

$$\nabla_\theta MSE(\theta) = \begin{pmatrix} \frac{\partial \mathrm{MSE}(\theta)}{\partial \theta_0} \\ \frac{\partial \mathrm{MSE}(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial \mathrm{MSE}(\theta)}{\partial \theta_n} \end{pmatrix} = \begin{pmatrix} \frac{2}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_0^{(i)} \\ \frac{2}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_1^{(i)} \\ \vdots \\ \frac{2}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_n^{(i)} \end{pmatrix}$$

Matrix Form Is More Concise

Partial Derivatives Form

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

$$\nabla_\theta MSE(\theta) = \frac{2}{m}\begin{pmatrix} \sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_0^{(i)} \\ \sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_n^{(i)} \end{pmatrix}$$

Matrix Form

$$\mathrm{MSE}(\theta) = \frac{1}{m}(\theta^T X - y)^2$$

$$\mathrm{MSE}(\theta) = \frac{1}{m}(X^T\theta - y)^2$$

$$\nabla_\theta \mathrm{MSE}(\theta) = \frac{2}{m}\mathrm{X}^{\mathrm{T}}(\mathrm{X}\theta - \mathrm{y})$$

$$\nabla_\theta \mathrm{MSE}(\theta) = \frac{2}{m}(\mathrm{X}^{\mathrm{T}}\mathrm{X}\theta - \mathrm{X}^{\mathrm{T}}\mathrm{y})$$

We can account for the $\frac{2}{m}$ constant and remove it by choosing a different η when updating θ

$$\theta^{(next\ step)} = \theta - \eta\ \nabla_\theta MSE(\theta)$$

$$\nabla_\theta \mathrm{MSE}(\theta) = \mathrm{X}^{\mathrm{T}}\mathrm{X}\theta - \mathrm{X}^{\mathrm{T}}\mathrm{y}$$

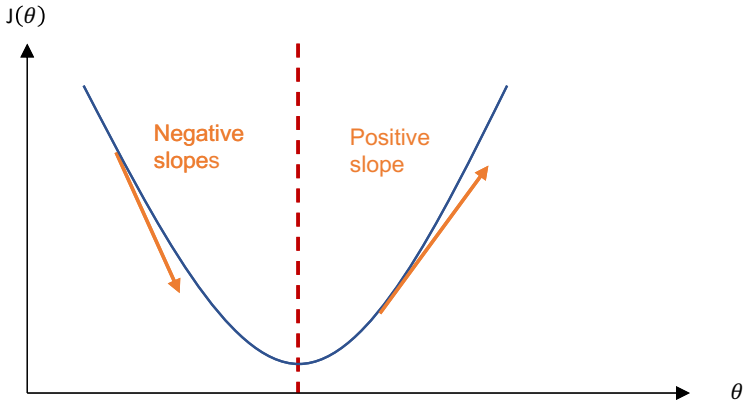
Why Do We Subtract the Gradient?

$$\theta^{(next\ step)} = \theta - \underbrace{\eta}_{f(x)} \nabla_\theta J(\theta)$$

For a 1D dataset the derivative of $f(x)$ gives the slope of $f(x)$ at the point x.

We would like to subtract the slope because it is in the ascending direction of $f(x)$.

$$Slope = \frac{\Delta J(\theta)}{\Delta \theta}$$



Why Do We Subtract the Gradient?

$$\theta^{(next\ step)} = \theta - \eta \nabla_\theta J(\theta)$$

The gradient of the cost function provides us with the direction of the steepest ascent in the cost function with respect to model's parameters.

Read page 84 Deep Learning Book for proof and [Watch this video] for demonstration.

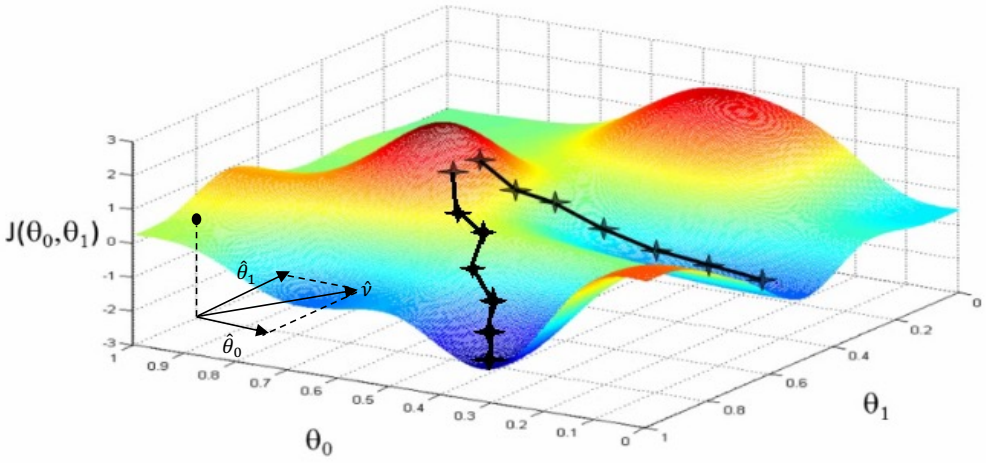


Image from: https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/

The method is named gradient descent because we move in the direction of the negative gradient.

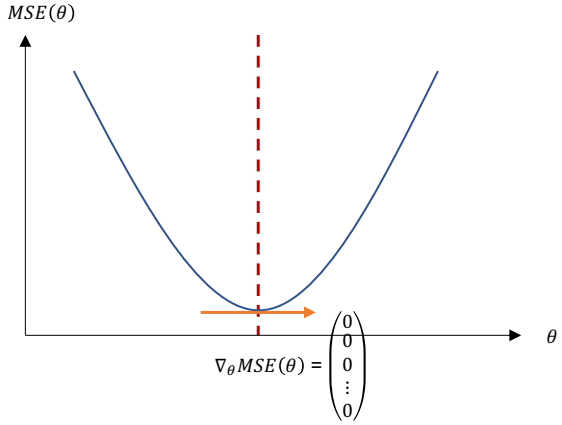
$$\theta^{(next\ step)} = \theta - \eta \nabla_\theta MSE(\theta)$$

Defining Convergence

Once the gradient is zero, you have reached a critical point.

The gradient is zero where every element of the gradient is equal to zero.

$$\theta^{(next\ step)} = \theta - \eta \nabla_\theta MSE(\theta) = \theta - \eta \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \theta$$

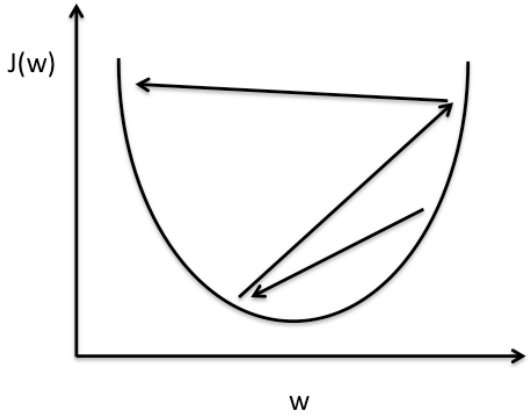


The Practical Definition For Convergence

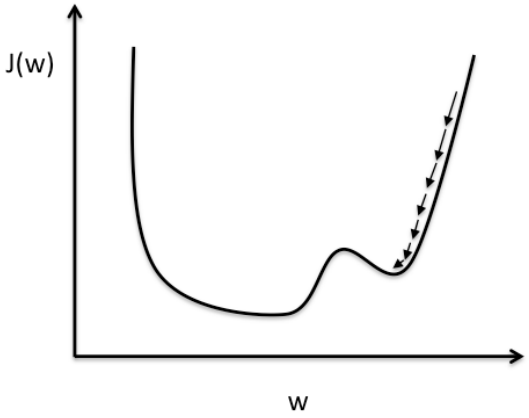
$$\left\|\nabla_\theta J(\theta)^{(t)}\right\| < \varepsilon \qquad \left\|\theta^{(t)} - \theta^{(t-1)}\right\| < \varepsilon \qquad \left\|J(\theta)^{(t)} - J(\theta)^{(t-1)}\right\| < \varepsilon$$

The step size is determined by the ***learning rate hyperparameter Eta*** ($\boldsymbol{\eta}$)

$$\theta^{(next\ step)} = \theta - \eta \ \nabla_\theta MSE(\theta)$$



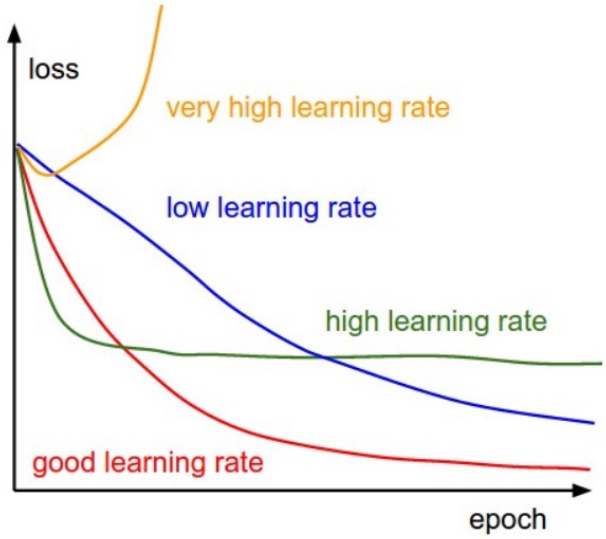
Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

Image from: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

The learning rate η effect on the loss



Each round of iteration through the entire dataset is called an epoch

The ***learning step size*** is proportional to the slope of the cost function.

$$\theta^{(next\ step)} = \theta - \eta \ \nabla_\theta MSE(\theta)$$

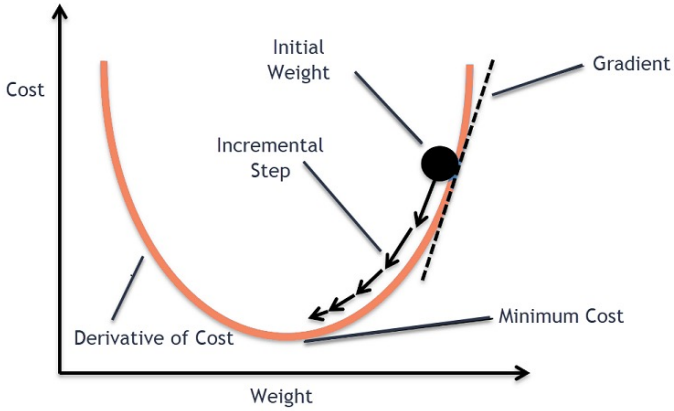


Image from: https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/

Since linear models cost functions are convex, they are guaranteed to reach the global minimum.

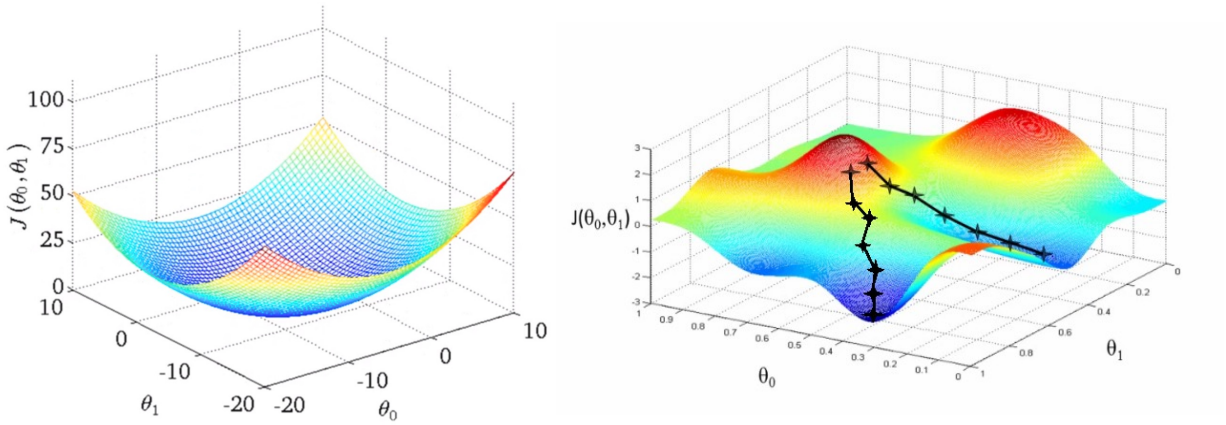


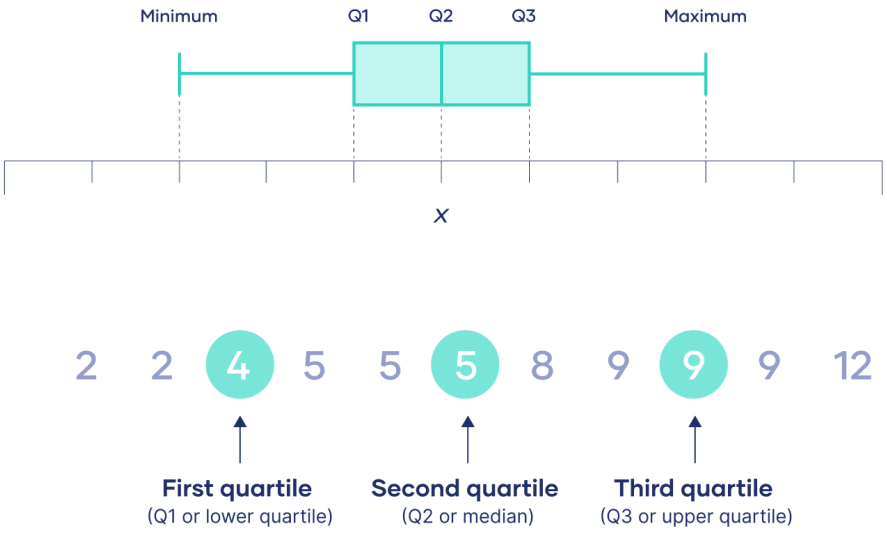
Image from: https://math.stackexchange.com/questions/2682688/why-cost-function-for-linear-regression-is-always-a-convex-shaped-function

When using Gradient Decent it is important to perform feature scaling.

Scaling the target values is generally not required.

There are 3 common ways to get all features to have the same scale

Scikit-Learn transformer	Formula	Values	Use
`MinMaxScaler`	$X_{min-max} = \frac{X - X_{min}}{X_{max} - X_{min}}$	Ranging from 0 to 1	Important for some algorithms e.g., Neural Networks
`StandardScalar`	$X_{standardization} = \frac{X - X_{mean}}{X_{SD}}$	Resulting distribution has zero mean and unit variance	Standardized data is much less affected by outliers in comparison to Min Max scaler.
`RubustScalar`	$X_{rubust} = \frac{X - X_{med}}{X_{q3} - X_{q1}}$	Most robust to outliers.	Scale features using statistics that are robust to outliers. Outliers can often influence the sample mean and variance in a negative way.



Images from: https://www.scribbr.com/statistics/quartiles-quantiles/

Note: Anything you learn, must be learned from the model's training data.

Therefore, it is important to **fit the scaling functions to the training data only**.

Only then can we use them to transform the training set, test set and new data.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train) # Obtains the scaler parameters based on
                                        Xtrain and uses them to scale X_train
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

Stochastic Gradient Descent (SGD)

Gradient Descent (Batch Gradient Descent) is very slow on very large training sets.

SGD computes proxy gradients based only on one single instance picked uniformly at random at each iteration (without replacement).

Therefore, it is much faster.

$$\begin{pmatrix}\theta_0\\ \theta_1\\ \vdots\\ \theta_n\end{pmatrix}^{(next\ step)} = \begin{matrix}\theta_0\\ \theta_1\\ \vdots\\ \theta_n\end{matrix} - \eta \begin{pmatrix}(\theta^T x^{(i)} - y^{(i)})x_0^{(i)}\\ (\theta^T x^{(i)} - y^{(i)})x_1^{(i)}\\ \vdots\\ (\theta^T x^{(i)} - y^{(i)})x_n^{(i)}\end{pmatrix}$$

$$\begin{pmatrix}\frac{2}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_0^{(i)}\\ \frac{2}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_1^{(i)}\\ \vdots\\ \frac{2}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_n^{(i)}\end{pmatrix}$$

With SGD it is essential for good generalization (less overfit) for each epoch to pick the instances randomly.

So, we shuffle the data after each epoch to keep learning general. By shuffling the data, we ensure that each data point creates an "independent" change on the model, without being biased by the same points before them.
E.g., if data point 20 is always used after data point 19, its own gradient will be biased with whatever updates data point 19 is making on the model.

Based on: https://datascience.stackexchange.com/questions/24511/why-should-the-data-be-shuffled-for-machine-learning-tasks

SGD is characterized by decreasing only on average since we are calculating the gradient with respect to a proxy cost function using only 1 sample.

Its random trajectory (sometimes in the opposite direction of the minima) has the advantage that it enables escaping from local minima.

This random trajectory has the disadvantage that the algorithm can never settle at the minimum. It will only reach a region close to the minima and keep bouncing around it.

To enable SDG settle at the minima, one solution is to gradually reduce the learning rate.

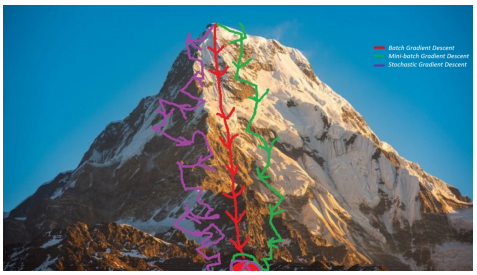
The steps start out large which helps make quick progress and escape local minima, then get smaller and smaller, allowing the algorithm to settle at the global minimum.

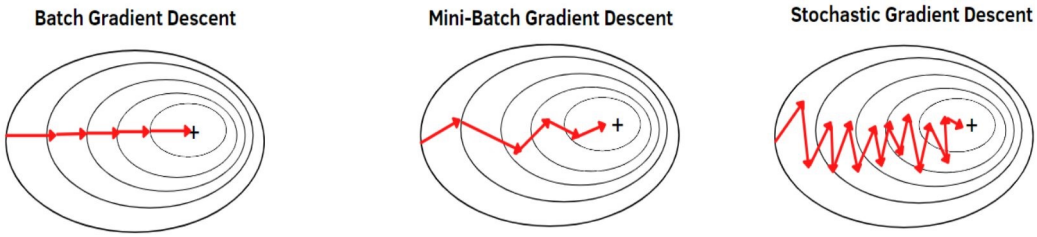
Mini-Batch Gradient Descent

Computes the gradients on small random sets of instances called mini-batches.

Shuffling at the beginning of each epoch prior to dividing the dataset into mini-batches helps with better generalization (less overfit) .

There is almost no difference after training; the Normal Equation and Gradient Decent variations produce very similar models.





Images from: https://areadinglife.com/2015/09/04/same-same-but-different/ and https://areadinglife.com/2015/09/04/same-same-but-different/

Gradient Descent scales better for large number of features and for many models its not possible to obtain the best coefficient values using a closed form solution.

Image from: https://areadinglife.com/2015/09/04/same-same-but-different/

We would like to convert the generalization error to percentage error

E.g., $Percentage\ Error = \frac{RMSE}{Mean\ True\ Label} \times 100$