Linear Models: Regression

DS 392 JAN2022

Deepak Subramani
Assistant Professor
Dept. of Computational and Data Science
Indian Institute of Science Bengaluru

Department of Computational and Data Sciences

भारतीय विज्ञान संस्थान

① Frame the problem

② Data Munging : Xarray pandas statistics

$X \rightarrow [ML] \rightarrow y$

$m \times n \rightarrow$ Red X $m \times d$

③ ML Modeling
- ① $h_\theta \rightarrow$
- ② θ :
  - ① Train validate Test
  - ② Obj Fn → Performance Metric

Dim Red

$X_{m \times n} \rightarrow X_{m \times d}$

$d << n$

$X_{m \times d} \xrightarrow{ML} h_\theta(X_{m \times d}; \theta) \; y$

④ Fine tune ; Ensemble

⑤ Present

⑥ Deploy
- a)
- b) Monitor

Regression Models

- Predict the value of a continuous variable in a supervised setting
- Use other variables as predictors

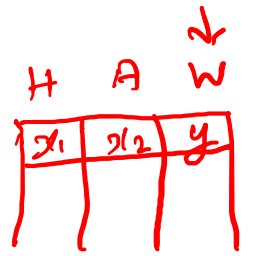
$$\hat{y} = h_\theta(X; \theta)$$

- Variable with a hat, $\hat{y}$ is usually the prediction from a model
- h is called the hypothesis, or the ML Model parametrized by θ
- θ can be calculated, estimated or ***learned*** from data by solving the optimization problem

$$\rightarrow \min \sum_{j=1}^{m} \left(\hat{y}^{(j)} - y^{(j)}\right)^2$$

Linear Regression

$y = 4x^2 - 3x + 1$



- In Linear Regression model h_θ is a linear function in θ

$$\hat{y} = \theta_1 x_1 + \theta_2 x_2 = X\theta$$

$\hat{W} = \theta_1 \cdot H + \theta_2 \cdot A$

param: θ_1, θ_2

$$\Rightarrow L = \left[\underbrace{\theta_1 x_1^{(1)} + \theta_2 x_2^{(1)}}_{\hat{y}^{(1)}} - y^{(1)}\right]^2 + \left[\theta_1 x_1^{(2)} + \theta_2 x_2^{(2)} - y^{(2)}\right]^2 + \left[\theta_1 x_1^{(3)} + \theta_2 x_2^{(3)} - y^{(3)}\right]^2$$

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \end{bmatrix}_{m \times n} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}_{n \times 1} \quad m \times 1$$

3×2

$\min_{\theta_1, \theta_2} L$

θ_1^*, θ_2^*

$\left.\frac{\partial L}{\partial \theta_1}\right|_{\theta_1^*} = 0$ & $\left.\frac{\partial L}{\partial \theta_2}\right|_{\theta_2^*} = 0$ Jacobian

Hessian

Convex

$$L = \left[\theta_1 x_1^{(1)} + \theta_2 x_2^{(1)} - y^{(1)}\right]^2$$

$$\frac{\partial L}{\partial \theta_1} = 2\left[\theta_1 x_1^{(1)} + \theta_2 x_2^{(1)} - y^{(1)}\right] \cdot x_1^{(1)} + 2\left[\theta_1 x_1^{(2)} + \theta_2 x_2^{(2)} - y^{(2)}\right] x_1^{(2)} + 2\left[\theta_1 x_1^{(3)} + \theta_2 x_2^{(3)} - y^{(3)}\right] x_1^{(3)} = 0$$

$$\frac{\partial L}{\partial \theta_2} = 2\left[\theta_1 x_1^{(1)} + \theta_2 x_2^{(1)} - y^{(1)}\right] \cdot x_2^{(1)} = 0$$

Set of linear equations.

Linear Regression

- In Linear Regression model h_θ is a linear function in θ

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

$$\Rightarrow \min \sum_{j=1}^{m} \left([\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n]^{(j)} - y^{(j)} \right)^2$$

$$\min (X\theta - y)^T (X\theta - y) \rightarrow \text{S.S.E}$$

$\Rightarrow \frac{\partial L}{\partial \theta} = 0$

$X\theta = y$ ($n \times 1$; $m \times n$)

$\rightarrow X^T X\theta = X^T y$ ($n \times n$, $n \times 1$, $n \times 1$)

$\hat{y} = X\theta$

$X\theta = \hat{y}$

$\hat{y} = y$

$X\theta = y$

Normal Equation $\rightarrow$ $\theta = X^{-1} y$

$X\theta = y$

$(X^T X)\theta = X^T y$

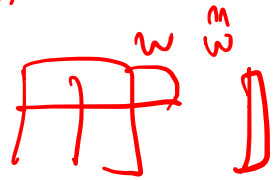
$\Rightarrow \theta = (X^T X)^{-1} X^T y$ — L.S.P

$(X^T X)^{-1} X^T$ — pseudo inverse

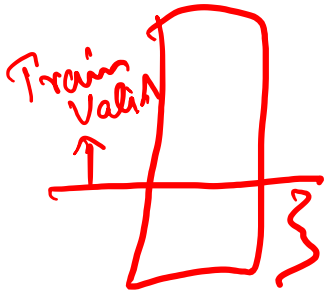
X: $m \times n$

5×2

3×3



Normal Equation



- The solution is the solution of the normal equation

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

- Solution:
  - Direct Evaluation,
  - Cholesky Decomposition,
  - QR Factorization,
  - SVD Method

Matrix Factorization

$X_{test}\,\theta^* = \hat{y}_{test}$

$L(y_{test}, \hat{y}_{test})$

Code from scratch for Direct Evaluation

```
import numpy as np
X = 3 * np.random.rand(1000,1)
y = 2 + 4*X + np.random.randn(1000,1)
X_b = np.concatenate((np.ones((1000,1)),X),axis=1)
theta_star = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
theta_star
```

Correlation is not Causation, But it helps Linear Models!

- While Data Cleaning, you will use correlation
- Predictors (columns of X) with high correlation to target (y) are good to make linear models
- What if X is correlated? - The issue is called multicollinearity
  - It is a problem if you want to identify contribution of one feature to the target
  - It is NOT a problem if you only want prediction accuracy

Correlation is not Causation, But it helps Linear Models!

$X^T X \theta^* = X^T y$

- Let us see how correlation helps
- Remove mean from X and y [Standard Scaler] and see the normal eq.

$$(X^T X)\theta^* = (X^T y); \quad \frac{1}{m}(X^T X)\theta^* = \frac{1}{m}(X^T y);$$

$$\rightarrow C_{XX}\theta^* = C_{Xy}$$

$E[[x - E[x]]^2]$

Σ $n \times n$ θ^* $=$ $n \times 1$

$\theta^* = C_{xy}$ $[-1, 1]$

$\theta_3 = 0$

$\theta_1 x_1 + \theta_2 x_1 + \theta_3 x_3$

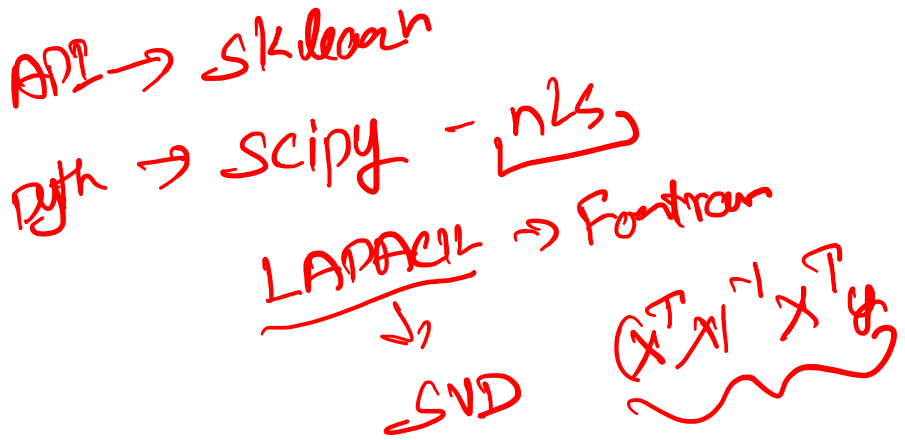
p-value

Hypothesis Testing

Scikit Learn

```
from sklearn.model_selection import train_test_split

from sklearn import linear_model
# Create linear regression object
lin_reg = linear_model.LinearRegression()
# Train the model using the training sets
lin_reg.fit(X_train, y_train)
# Make predictions using the testing set
y_pred = lin_reg.predict(X_test)
# The coefficients
print('Coefficients: \n', lin_reg.intercept_,lin_reg.coef_)
print('Prediction: \n', y_pred)
```



[Ref: https://scikit-learn.org/]

lin_reg.fit() – Behind the Scene

- Ordinary Least Squares (scipy.linalg.lstsq);
  - scipy.linalg.lstsq uses a LAPACK driver to solve the normal equations
  - Default is DGELSD – Double Precision, General Least Squares using SVD
  - DGELSY – Double Precision, General Least Squares using QR or LQ factorization
  - GE – General Matrices (Unsymmetric, no special property, convention in LAPACK)

Computational Cost

- The Cholesky-based algorithm is particularly useful when $m >> n$ and it is practical to store $X^T X$ but not X itself.
- It can also be less expensive than the alternatives when $m >> n$ and X is sparse.
- However, this approach must be modified when X is rank-deficient or ill conditioned to allow pivoting of the diagonal elements of $X^T X$.
- The QR approach avoids squaring of the condition number and hence may be more numerically robust, but often comes at an extra cost.
- While potentially the most expensive, the SVD approach is the most robust and reliable of all.

Computational Cost

- Solving the normal equation by QR or SVD method is $O(mn^2)$ – square in the number of features, and linear in the number of data points
- If we increase features by a factor 2, then computational cost increases by a factor 4.
  - Both the Normal Equation and the SVD approach get very slow when the number of features grows large (e.g., 100,000).
- As they are linear in the number of data points, they can handle large training sets efficiently, provided data can fit in memory.
- We need a better optimization algorithm when train data is large.

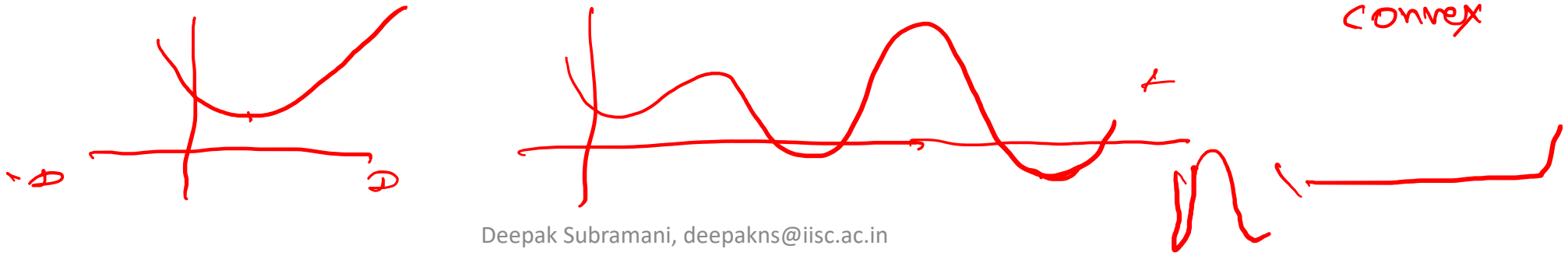
Need for Iterative Algorithms

- Large problem size – lots of feature vectors and lots of data
- We used squared loss for the accuracy, what if we have another accuracy metric?
- What about nonlinear accuracy metrics? Nonlinear function $h()$
- Traditionally, nonlinear function (e.g., neural network) were trained with squared loss using a class of algorithms known as nonlinear least squares
- Nonlinear least squares algorithm Gauss-Newton and Levenberg Marquardt algorithms were very popular to train shallow neural network models
- Now we use gradient descent algorithms which is what is called as a "first-order method".
- Let us see these iterative algorithms now.

Deepak Subramani, deepakns@iisc.ac.in

Gradient Descent - Intuition

- Suppose you are lost in the mountains in a dense fog, and you can only feel the slope of the ground below your feet.
- A good strategy to get to the bottom of the valley quickly is to go downhill in the direction of the steepest slope.
- This is exactly what Gradient Descent does: it measures the local gradient of the error function with regard to the parameter vector θ, and it goes in the direction of descending gradient.
- Once the gradient is zero, you have reached a minimum!



θ $\quad \frac{\partial L}{\partial \theta} = 0$

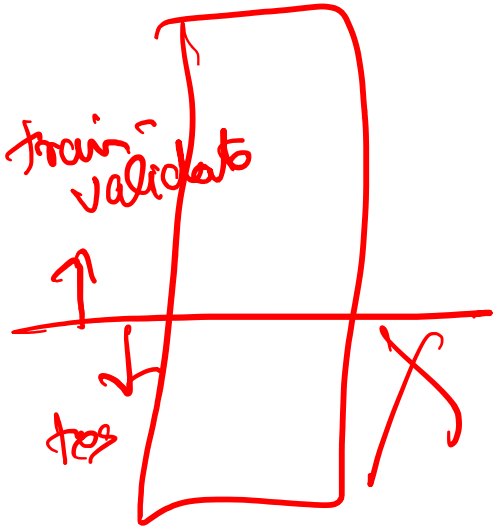
$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \vdots \end{bmatrix}$$

$$\Rightarrow \theta_{(k+1)} = \theta_{(k)} - \eta \nabla L_{\theta(k)}$$

θ : param

η : hyper param
tolerance
max_iter

$$\theta_{(0)} - \eta \nabla L$$



$\nabla L \approx 0$ $\quad$ tol

η : Learning Rate
tol
max_iter

$\theta^* = \theta_{99}$

$$\frac{\theta_{(k+1)} - \theta_{(k)}}{\eta}$$

Gradient Descent - Variants

- Batch Gradient Descent
  - Use the full data and calculate the Jacobian of the objective function

$$\nabla f(\theta) = \sum_{j=1}^{m} r_j(\theta)\nabla \mathrm{r_j}(\theta) = J(\theta)^T r(\theta) = X^T(X\theta - y)$$

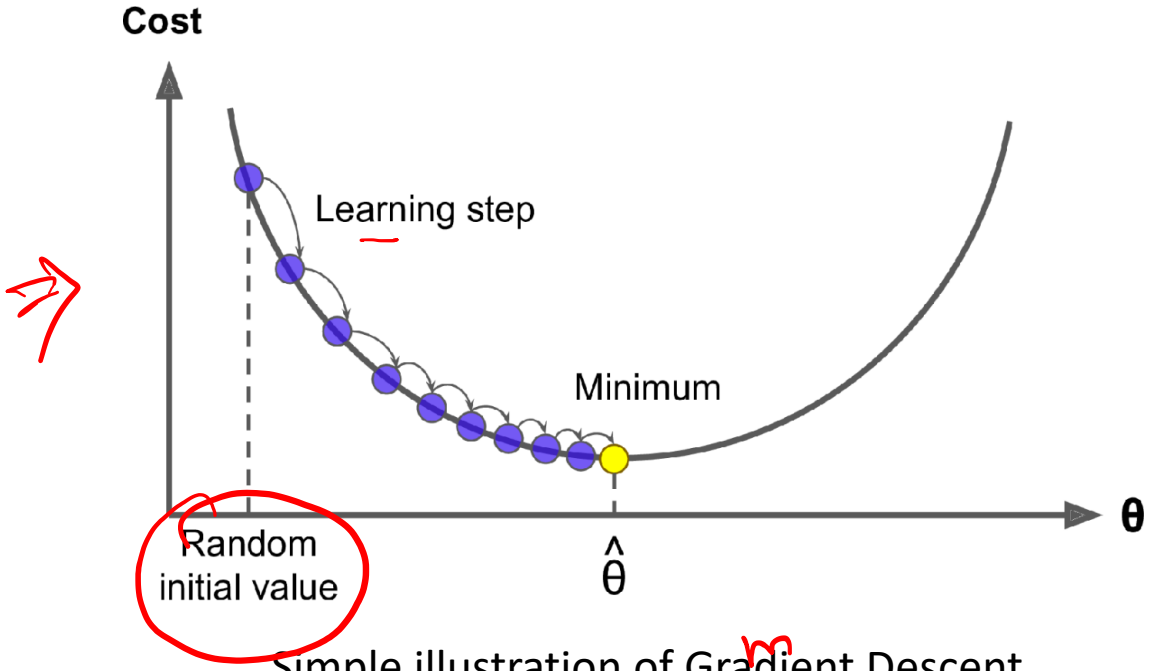
  - Iteration of the optimization algorithm

$$\theta_{k+1} = \theta_k - \eta_k \nabla f(\theta_k)$$

- Stochastic Gradient Descent
  - Use only one data point at a time.
  - Usually m iterations is called an epoch

Deepak Subramani, deepakns@iisc.ac.in

Gradient Descent

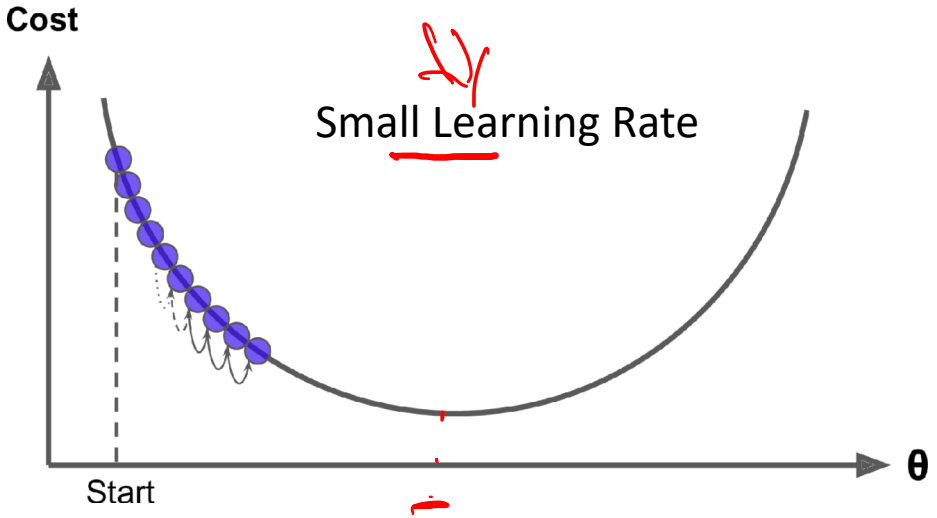


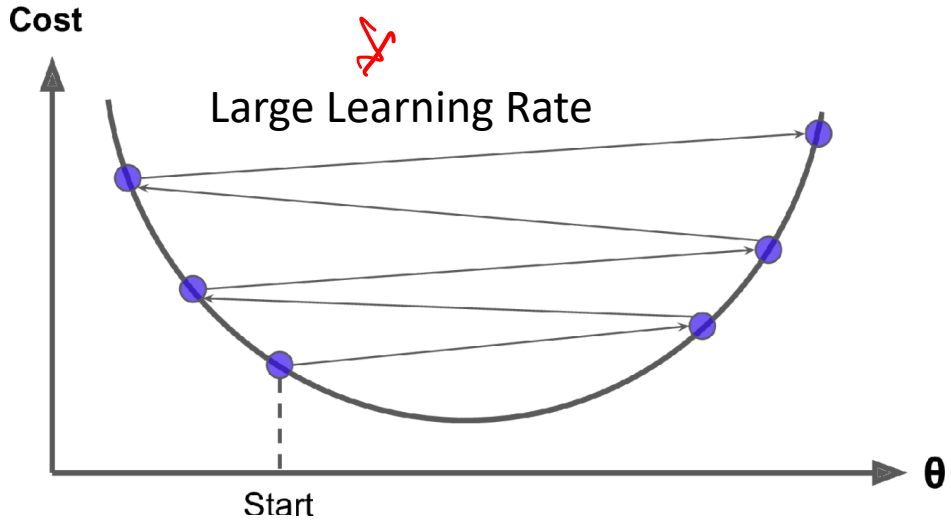
Simple illustration of Gradient Descent

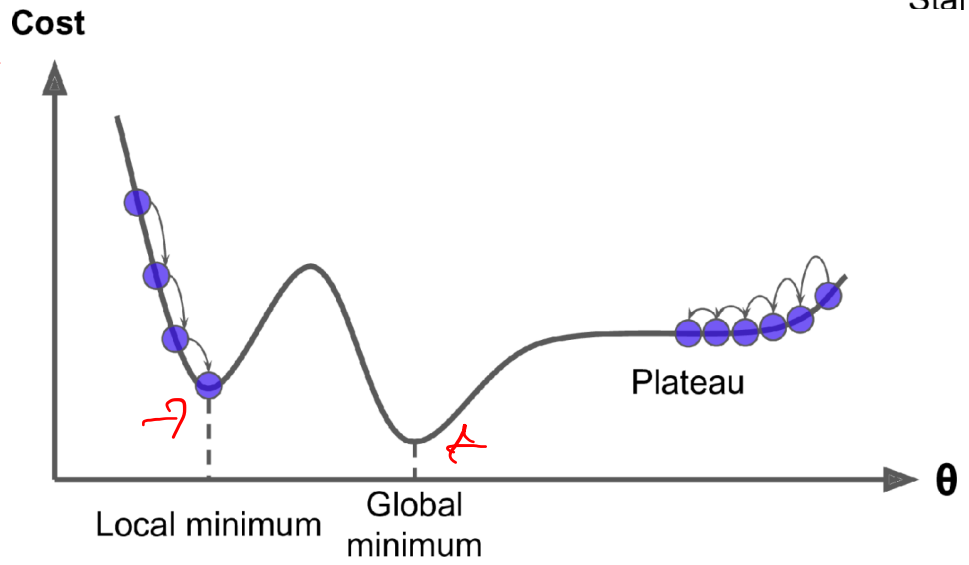
Optimization vs Learning:

- In Optimization, we have to compute the step length accurately
- In learning, we don't care as much about being exactly optimal at every iteration, so we choose a learning rate η_k as a hyperparameter, instead of an exact step length
- Define Hyperparameter – They are parameters of the training/model that are not trainable with data directly, but are chosen by the designer/developer/user using some other information source (usually, validation)

Gradient Descent







- Gradient Descent may miss global minima
- Very slow in a Plateau
- Problem Scaling Issues
- To ensure fast convergence, use scaled features

Example of Non-Convex Function

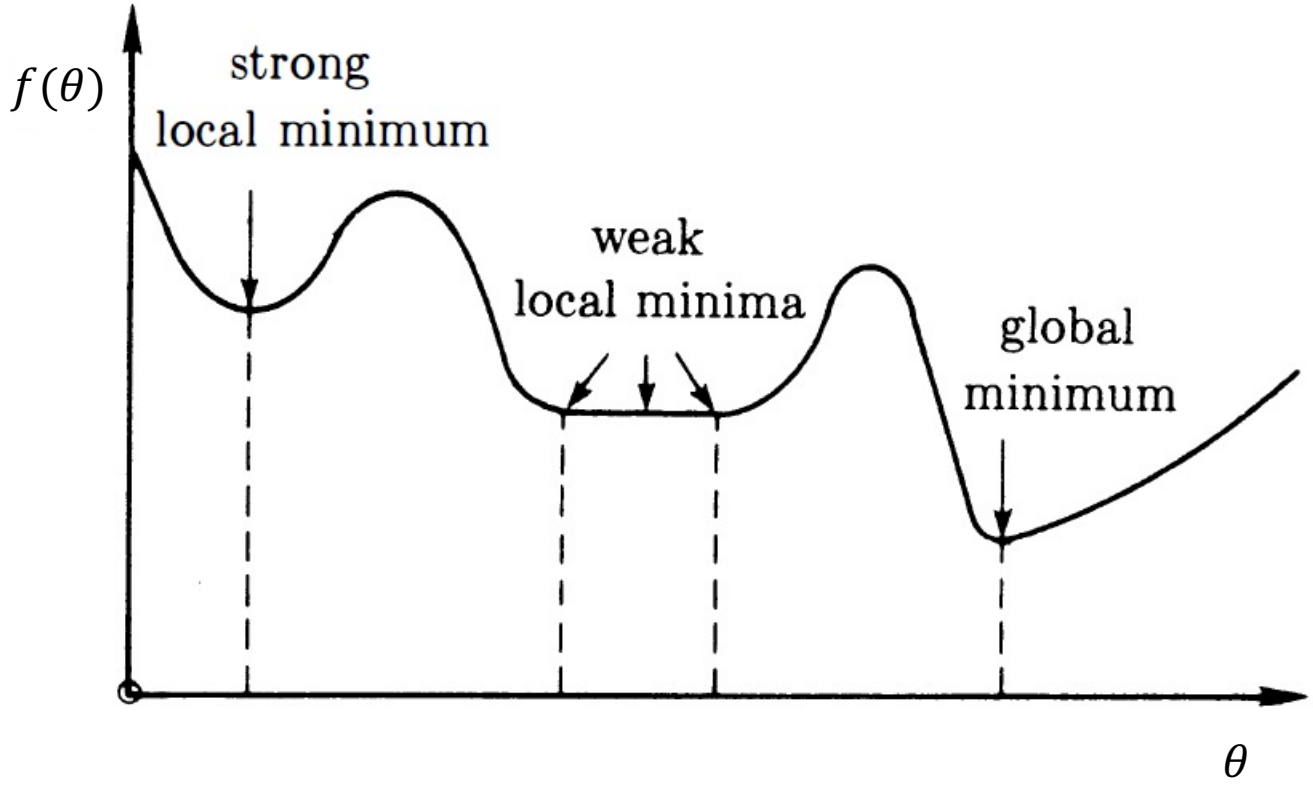


Figure 3b. Examples of minima in the univariate case.

Scikit Learn – SGDRegressor

```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3, penalty=None, eta0=0.1)

sgd_reg.fit(X_train,y_train.ravel())


print(sgd_reg.intercept_, sgd_reg.coef_)
```

Note: .ravel() method of numpy array is used to make y_train a 1-d array which is expected by SGDRegressor

penalty is used to specify regularization which we will discuss shortly

[Ref: https://scikit-learn.org/; Geron]

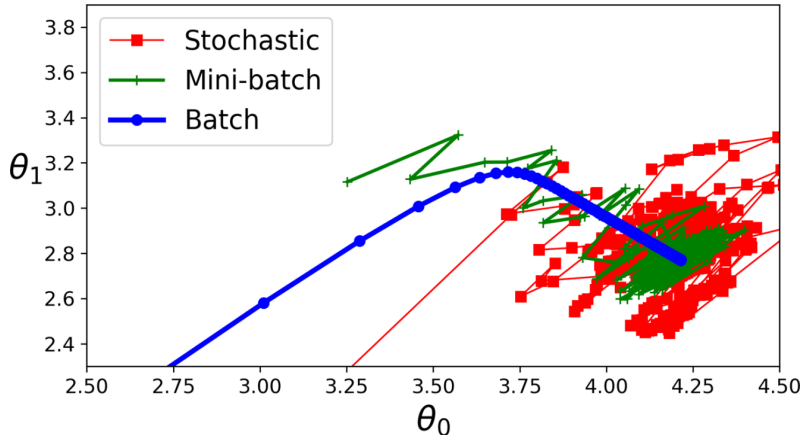
Department of Computational and Data Sciences

INDIAN INSTITUTE OF SCIENCE
भारतीय विज्ञान संस्थान

@the_data_department creates comics about AI, ML, math and stats

Mini-batch Gradient Descent

- Mini-batch GD is in between Batch GD (Full data set at once) and Stochastic GD (one instance at once)
- Mini-batch computes the gradient on small random sets of instances called mini-batches
- The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs.
- Min-batch SGD is less erratic than SGD



Comparison of algorithms for Linear Regression

Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

Geron Chapter 4, Page 128, Table 4-1

Out of core algorithms are those that can process data that are too large to fit into a computer's main memory at once.

Audience Poll

1. If there are m train data points, how many data points does batch GD use?
   i. m,
   ii. 1,
   iii. m/2,
   iv. one pre-determined fraction of m
2. Which of the following is TRUE
   i. Batch GD is a stochastic algorithm – FALSE
   ii. When SGD sees all points once, it is called an epoch – TRUE
   iii. Mini Batch GD is a deterministic algorithm – FALSE
   iv. Batch GD uses one data point at a time – FALSE
3. Which of the following are the hyperparameters of Batch GD
   i. Learning Rate,
   ii. Number of Iterations,
   iii. Size of Batch,
   iv. Sampling Strategy