BK AI Amateur Course

The Four Horsemen of the Apocalypse

Abstract In this report, we shall review briefly about Dung Lai Lap Trinh Course that we have studied through. Some algorithms that this report is going to cover are K-Means, K Nearest Neighbor, Gradient Descent, and Linear Regression. These algorithms will be shown from both mathematical and coding perspectives. To get through this report, you must have some introductory level knowledge at Calculus, Linear Algebra and Python, as we are going to use these for illustrating the algorithms.

Keyword: K-Means, K Nearest Neighbor, Gradient Descent, Linear Regression, Algorithms, Math, Python

Member: Trần Đức Nam
Phạm Tấn Phước
Trịnh Mạnh Hùng
Nguyễn Quốc Việt

Contents

1 K-Mean Algorithm Perspective

Abstract - In some pattern recognition problems, the training data consists of a set of input vectors x without any corresponding target values. The goal in such unsupervised learning problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine how the data is distributed in the space, known as density estimation. K-Means is one of the simplest unsupervised clustering algorithm which is used to cluster our data into K number of clusters. In this part, we will discuss about K-Means algorithm from mathematical perspective and use Pygame package to illustrate how this algorithm works.

1.1 Introduction

Clustering can be considered the most important unsupervised learning problem; so, as every other problem of this kind, it deals with finding a structure in a collection of unlabeled data. A loose definition of clustering could be "the process of organizing objects into groups whose members are similar in some way". A cluster is therefore a collection of objects which are "similar" between them and are "dissimilar" to the objects belonging to other clusters.

Clustering techniques can be used in various areas or fields of real-life examples such as data mining, web cluster engines, academics, bioinformatics, image processing & transformation, and many more, emerged as an effective solution to above-mentioned areas.

K-Means is one of the most popular clustering algorithms. The main goal of this algorithm is to find groups in data and the number of groups is represented by K. It is an iterative procedure where each data point is assigned to one of the K groups based on feature similarity. The data points here could be students who have some features such as Math score, SAT, English score, . . . or pixels in an image, etc

1.2 Mathematical perspective

1.2.1 Outline of the algorithm

Assuming we have input data points x_1, x_2, x_3, . . . , x_n and value of K (the number of clusters needed). We follow the below procedure:

1. Pick K points as the initial centroids from the dataset, either randomly or the first K.
2. Find the Euclidean distance of each point in the dataset with the identified K points (cluster centroids).
3. Assign each data point to the closest centroid using the distance found in the previous step.
4. Find the new centroid by taking the average of the points in each cluster group.
5. Repeat 2 to 4 for a fixed number of iteration or till the centroids don't change.

1.2.2 Euclidean distance

If p = $(p_1, p_2, \ldots, p_n)$ and q = $(q_1, q_2, \ldots, q_n)$ then the distance is given by

- Between 2 points in two dimensions space

$$d\left(p,q\right) = \sqrt{\left(q_1 - p_1\right)^2 + \left(q_2 - p_2\right)^2}$$

- In general, for points given by Cartesian coordinates in n-dimensional Euclidean space, the distance is

$$d\left(p,q\right) = \sqrt{\left(q_1 - p_1\right)^2 + \left(q_2 - p_2\right)^2 + \ \ldots \ + \left(q_i - p_i\right)^2 + \ldots + \left(q_n - p_n\right)^2}$$

1.2.3 Assigning each point to the nearest cluster

If each cluster centroid is denoted by c_i, then each data point x is assigned to a cluster based on

$$\underset{c_i \epsilon C}{argmin}\, dist(c_i, x)^2$$

Here dist() is the Euclidean distance.

1.2.4 Finding the new centroid from the clustered group of points

$$c_i = \frac{1}{|S_i|} \sum_{x_i \epsilon S_i} x_i$$

S_i is the set of all points assigned to the ith cluster.

1.3 Implement from scratch

In this section, we will use Python for implementing each step of Kmeans algorithm.

1.3.1 Euclidean Distance between two points in space

```python
def euclidean_distance(self, point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
```

In this code section, we create a function named euclidean_distance to calculate the Euclidean Distance between two points. This function will receive 2 parameters point1, point2 and then assign them to the formula introduced above for returning.

1.3.2 Assigning each point to the nearest cluster

```python
for point in data:
    distances = []
    for index in self.centroids:
        distances.append(self.euclidean_distance(point, self.centroids[index]))

    cluster_index = distances.index(min(distances))
    self.classes[cluster_index].append(point)
```

This code section includes two loops. The outer loop will iterate over every point in our dataset. The inner loop will find all the distances between a point and K centroids then add into the "distances" list.

The last two lines are used to find the cluster index that the datapoint belongs to and then add the new datapoint to the cluster group.

1.3.3 Finding the new centroid from the clustered group of points

```python
for cluster_index in self.classes:
    self.centroids[cluster_index] = np.average(self.classes[cluster_index], axis=0)
```

This code section loops over every cluster and calculate the average coordinates of them. These average coordinates will be the new centroid for each cluster.

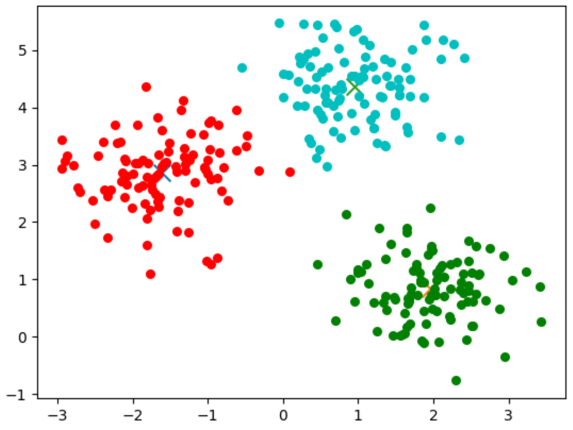
1.3.4 Full implementation

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import make_blobs
import math
```

```python

class K_Means:

    def __init__(self, k=3, max_iterations=500):
        self.k = k
        self.max_iterations = max_iterations

    def euclidean_distance(self, point1, point2):
        return math.sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2)

    def fit(self, data):

        # let the first K points from the dataset be the initial centroids
        self.centroids = {}
        for i in range(self.k):
            self.centroids[i] = data[i]

        # start K-Mean clustering
        for i in range(self.max_iterations):
            # create classifications the size of K
            self.classes = {}
            for j in range(self.k):
                self.classes[j] = []  # empty them

            # find the distance between the points and the centroids
            for point in data:
                distances = []
                for index in self.centroids:
                    distances.append(self.euclidean_distance(point, self.centroids[index]))

                cluster_index = distances.index(min(distances))
                self.classes[cluster_index].append(point)

            for cluster_index in self.classes:
                self.centroids[cluster_index] = np.average(self.classes[cluster_index], axis
    =0)


def main():
    K = 3
    X, y_true = make_blobs(n_samples=300, centers=K,
                           cluster_std=0.60, random_state=0)

    k_means = K_Means(K)
    k_means.fit(X)

    #print(k_means.centroids)

    #Plotting
    colors = 10 * ["r", "g", "c", "b", "k"]

    for centroid in k_means.centroids:
        plt.scatter(k_means.centroids[centroid][0], k_means.centroids[centroid][1], s=130,
    marker="x")

    for cluster_index in k_means.classes:
        color = colors[cluster_index]
        for features in k_means.classes[cluster_index]:
            plt.scatter(features[0], features[1], color=color, s=30)
    plt.show()
main()
```

In this program, we use sklearn.datasets.make_blobs to generate a random dataset and then use matplotlib.pyplot to visualize the result.

Output from the above program:



Number of Datapoints = 300, K = 3

1.4 Implement using Scikit-learn

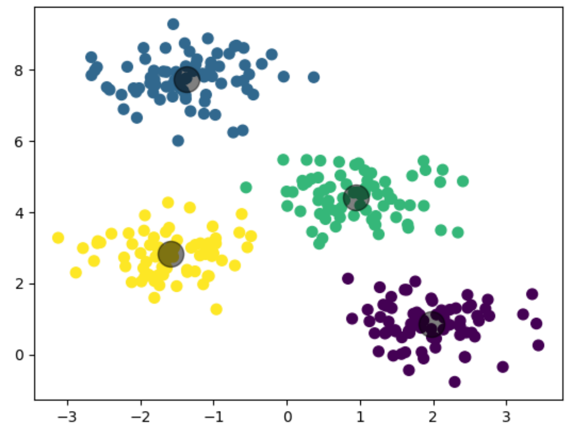
Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

In this section, we use Scikit-learn to build a Machine Learning model using Kmeans algorithm. Some methods in sklearn that we are going to use:

- sklearn.cluster.KMeans(): K-Means clustering - Initialize the model.
- fit(): Compute k-means clustering. Basically, this method is used to train the model that we have initialized based on a dataset.
- predict(): Using the trained model to predict the closest cluster each sample in X belongs to.

```python
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

K = 4
X, y_true = make_blobs(n_samples=300, centers=K,
                       cluster_std=0.60, random_state=0)
k_means = KMeans(n_clusters=K).fit(X)

cluster_centres = k_means.cluster_centers_

y_kmeans = k_means.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

for centroid in cluster_centres:
    plt.scatter(centroid[0],  centroid[1], s=300,  c='black', alpha=0.5)

plt.show()
```

Output from the above program:



Number of Datapoints = 300, K = 4

1.5 Conclusion

That is all we need to understand K-means algorithm. Now, let's give a glance at some pros and cons of this algorithm.

- Advantages:
    1. High Performance: K-Means algorithm has linear time complexity and it can be used with large datasets conveniently. With unlabeled big data K-Means offers many insights and benefits as an unsupervised clustering algorithm.
    2. Easy to Use: K-Means is also easy to use. It can be initialized using default parameters in the Scikit-Learn implementation. According to this approach, parameters like number of clusters (8 by default), maximum iterations (300 by default), initial centroid initialization (10 by default) can easily be adjusted later on to suit the task goals.
    3. Unlabeled Data: This is an advantage for every unsupervised machine learning algorithm. If your data has no labels (class values or targets) or even column headers, K-Means will still successfully cluster your data.
- Disadvantages:
    1. Result repeatability: One of the inconsistencies of K-Means algorithm is that results will differ based due to random centroid initialization. Unless you pick the centroids at fixed positions, which is not a common practice, K-Means can come up with different clusters after its iterations.
    2. Spherical Clustering Only: K-Means generates spherical clusters. So, if you have overlapping clusters or arbitrary shapes K-Means won't be able to cluster those.
    3. Clusters Everything: Another point about how K-Means works is that it will include every data sample in the clusters it generates. This means, if you would like to exclude outliers or certain sample groups it won't be possible with K-Means algorithm which creates spherical clusters that cover the whole dataset.

2 K-Mean Application

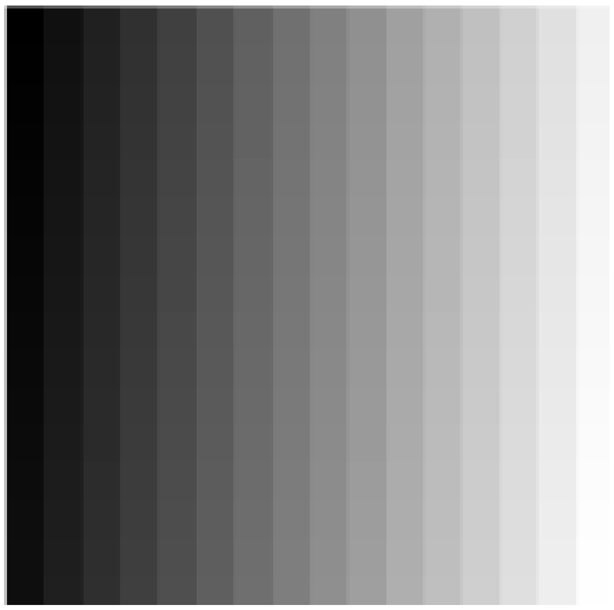
After understanding how the K Means algorithms works and the mathematic behind it? We'll introduce you one of the famous or popular application of K Means algorithm in real-life. That is **Data Compression**. In the other words, K-Means algorithm is applied to reduce the size of image and outputs a new image without the smaller number of color as compared to the original one.
Before walking into main part of this section, it should be necessary to take a look at "How is an image constructed?" or "Construction of Image in terms of matrice"!

2.1 Image and Matrix relationship

Abstract: The need for matrix presentation by image is clear to all who in their work use high dimension matrices. Particular problem here make matrices that could not normally be presented by images – those whose elements are arbitrary real, or even complex numbers. Here we propose a solution for visualization of arbitrary, high dimension matrix.
There is a relation between matrices and digital images. A digital image in a computer is presented by pixels matrix. On the other hand, there is a need (especially with high dimensions matrices) to present matrix with an image. The motive of this work is to perceive matrix elements nature and arrangement: to recognize matrix parts – positive from negative, real from complex, bigger from smaller (by magnitude).

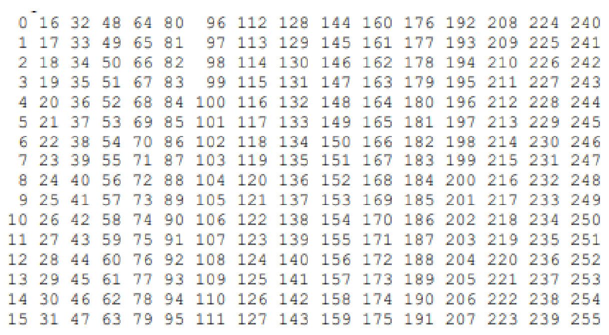
2.1.1 Image and Its Matrix

A **digital grayscale image** is presented in the computer by pixels matrix. Each pixel of such image is presented by one matrix element – integer from the set 0,1,2,..255. The numeric values in pixel presentation are uniformly changed from zero (black pixels) to 255 (white pixels).
For example:



Grayscale Image Pic1.

The above image is actually presented with the matrix:



```
 0 16 32 48 64 80  96 112 128 144 160 176 192 208 224 240
 1 17 33 49 65 81  97 113 129 145 161 177 193 209 225 241
 2 18 34 50 66 82  98 114 130 146 162 178 194 210 226 242
 3 19 35 51 67 83  99 115 131 147 163 179 195 211 227 243
 4 20 36 52 68 84 100 116 132 148 164 180 196 212 228 244
 5 21 37 53 69 85 101 117 133 149 165 181 197 213 229 245
 6 22 38 54 70 86 102 118 134 150 166 182 198 214 230 246
 7 23 39 55 71 87 103 119 135 151 167 183 199 215 231 247
 8 24 40 56 72 88 104 120 136 152 168 184 200 216 232 248
 9 25 41 57 73 89 105 121 137 153 169 185 201 217 233 249
10 26 42 58 74 90 106 122 138 154 170 186 202 218 234 250
11 27 43 59 75 91 107 123 139 155 171 187 203 219 235 251
12 28 44 60 76 92 108 124 140 156 172 188 204 220 236 252
13 29 45 61 77 93 109 125 141 157 173 189 205 221 237 253
14 30 46 62 78 94 110 126 142 158 174 190 206 222 238 254
15 31 47 63 79 95 111 127 143 159 175 191 207 223 239 255
```

Grayscale Image matrix.

Unlike "Grayscale Image", Color image represented as a 2D matrix. Each matrix cell (pixel) contains 3 parameters (R, G, B) that give the color saturation of Red, Green and Blue at the pixel location.

2.2 Application in Image Compression

In this subsection, to have clear illustration of K Means Image Compression, we works with the colorful image only. Let's each pixel be the observation (X) then the number of pixel in an image be the number of observations. Each observation has three properties which are RGB value. In this case, K-Means algorithm is applied to identify K main colors in that image. To illustration the application of K Means, this image will be chosen.

2.2.1 Import Library

There are 3 main important library that we should include, that are numpy, mathplotlib, and KMeans.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import  KMeans
```

2.2.2 Import Data

Here, Data means the above images which will be store in a variable named [img].

```python
img = plt.imread("beach.jpg")
```

After import the image, let's check the size of this image matrix.

```python
height = img.shape[0]
width = img.shape[1]
print("Number of row:", height)
print("Number of column:", width)
```

we see that, the matrix $X \in R^{1200*1600*3}$. And for each entry (or pixel) we has a column vector $x_i = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$, where R, G, B is a scalar number has range (0-255). To make X be standard, where ($X \in R^{n*m}$, n is number of observations, m is number of features). As we discussed before, each pixel is a observations so n = 1200*1600 and m = 3.

```python
img = img.reshape(width*height,3)
```

2.2.3 Build K Means Model

Instead of coding the algorithm by hand as we did, it is more convenient for us to use the available function from *Kmeans* library to buid up the KMeans model.

```python
kmeans = KMeans(n_clusters= 5).fit(img)
labels = kmeans.predict(img)
clusters = kmeans.cluster_centers_
```

At this point, we choose K = 5, that means the model will create a new image which just has only 5 basic colors.

2.2.4 Visualize New Image

First, we should create new image called [img2] which has same size as variable [img], and contains only zero for each entry.Then we changes each entry in [img2] to corresponding value with respect to the KMeans image.

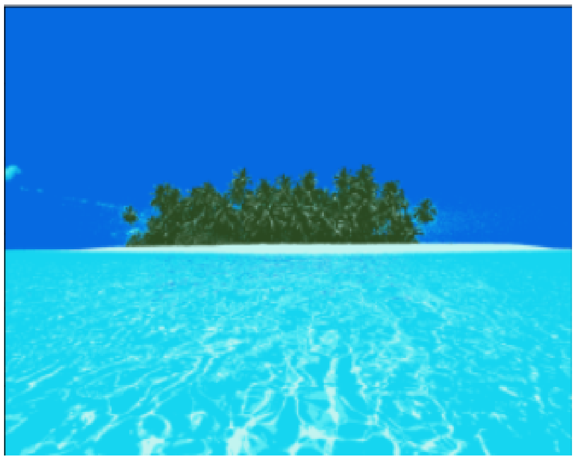
```python
img2 = np.zeros_like(img)
for i in range(len(img)):
    img2[i] = clusters[labels[i]]
```

After that, Reshaping the [img2] to be an standard formula matrix of an color image.

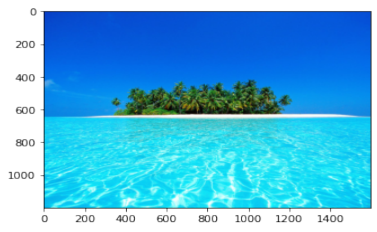
```python
img2 = img2.reshape(height,width,3)
```

Finally, we can check to see whether the different between two images: the initial image, and the compressed image.

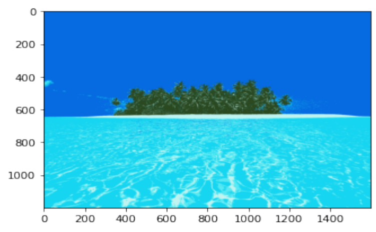
(a) Original image


(b) K Means image - K = 5

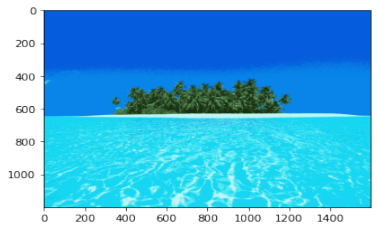
Clearly, the left image has higher resolution than the right image, which is re-draw by only 5 main colors (green, blue, white, light yellow and dark blue). Remember K = 5 is just the first test, to illustrate the application we'll choose 4 distinguished K (K = 5, K = 7, K = 10,K = 13) then make the table shows how file size of the original image is reduced. Finally, we get these four images:



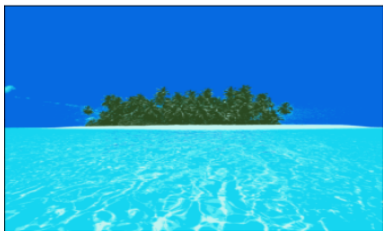
(a) The Original image



(b) K = 5



(c) K = 7


(d) K = 13

Hình 2: Image Segmentation on 1200x1600 image

The table below shows how file size of the original image is reduced.

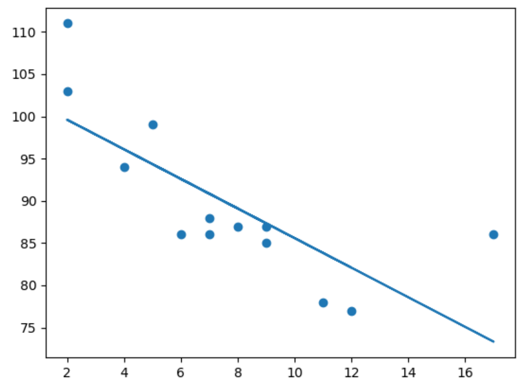
K (Color)	File Size (KB)
Original Image	114
13	91
7	75
5	67

2.3 Conclusion

K-Means Algorithm could be very simple and quick to be implemented, the clustering problems where all clusters are centroids and separated can be solved by the algorithms. However, it will not be effective when the dataset and clusters are more complex.
This report doesn't come with new idea to improve the effectiveness of the algorithm, the aim of the report is to introduce the reader to a basic, entry level clustering methods with some visual example on 2-dimensional and 3-dimensional dataset.

3 Linear Regression by Algebra

Abstract In Linear Regression, we are trying to draw a line through n available points in space such that the line does not need to go through all the points, but just needs to be as close to the points as possible. That will help when adding a new point and in space, we can easily estimate how much the value of that point is approximately through the line that we have found.



Linear Regression illustration.

3.1 Introduction

Linear Regression is supervised learning algorithm which the output of the training set has already been labeled. This is one of the most effectively basic algorithm in Machine Learning to address the "regression" or "predict" problem in the real-world.

Considering a simple real life problems! Assume that after a survey on 1000 students, we know on daily average how much time he/she spends for studying (x_1 hours), for playing videos game (x_2 hours), and how many math problems that he/she solve for one day (x_3) ? Now Whether a new transfer student comes to school with the parameters of study time, play time, number of problems solved per day, can we predict that student's final math score? If so, what will the prediction function $y = f(X)$ look like? where $X = [x_1, x_2, x_3]$ is a row vector contains the input information, y is a scalar which represent the output (or final math score).

Easily, we know that:

1. The more studying time, the higher score student get.
2. The more playing video game time, the less score student get.
3. The more mathematical problems solved for one day, the higher score student get.

The simplest function that can describe the relationship between final math score and the three inputs is:

$$y \approx f(X) = \hat{y}$$

$$f(X) = w_1x_1 + w_2x_2 + w_3x_3 + w_0$$

In which, w_1, w_2, w_3, w_4 is a constant (or **weights**), w_0 is **bias**. The above relationship $y \approx f(X)$ indicates a linear relationship. Our problems is a regression problems or find the optimization of parameters w_1, w_2, w_3, w_4. So these are reasons why we call it is *Linear Regression*.

3.2 MATHEMATICAL ANALYSIS

3.2.1 Input and Output

The Linear Regression Algorithms take a set of Observations X = $\begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{bmatrix} \in R^{nx(m+1)}$ where each observation is a $m-$ dimensional row vector, n is the number of observations (members) and m is the numbers of features in each observation. **Note:** To apply the algorithm, $n > m+1$.

The algorithm outputs is a prediction vector $\hat{y} = \begin{bmatrix} \hat{y^{(1)}} \\ \hat{y^{(2)}} \\ \vdots \\ \hat{y^{(n)}} \end{bmatrix} \in R^{n*1}$ which is the approximate result of the expected output.

3.2.2 Loss function and Algebra solving

Before we start, let's understand some important notation.
$w = [w_0, w_1, w_2, ..., wm]^T$ is a coefficient vector that needs to be optimized, w_0 is often called as bias.
$x^{(i)} = [1, x_1^{(i)}, x_2^{(i)}, ..., x_m^{(i)}]$ is the observation i^{th} in the set of n observations, each observation has m features.
Because our model is linear relation $\Longrightarrow \hat{y_i} = x^{(i)}w$
General form of Linear Regression model:

$X \in R^{n*(m+1)}, w \in R^{(m+1)*1}, y \in R^{n*1}$

$$\text{X} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{bmatrix}, \text{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}, \hat{y} = Xw = \begin{bmatrix} w_0 + w_1x_1^{(1)} + ... + w_mx_m^{(1)} \\ w_0 + w_1x_1^{(2)} + ... + w_mx_m^{(2)} \\ w_0 + w_1x_1^{(3)} + ... + w_mx_m^{(3)} \\ \vdots \\ w_0 + w_1x_1^{(n)} + ... + w_mx_m^{(n)} \end{bmatrix}, y-\hat{y} = \begin{bmatrix} y^{(1)} - (w_0 + w_1x_1^{(1)} + ... + w_mx_m^{(1)}) \\ y^{(2)} - (w_0 + w_1x_1^{(2)} + ... + w_mx_m^{(2)}) \\ y^{(3)} - (w_0 + w_1x_1^{(3)} + ... + w_mx_m^{(3)}) \\ \vdots \\ y^{(n)} - (w_0 + w_1x_1^{(n)} + ... + w_mx_m^{(n)}) \end{bmatrix}$$

Loss function: $L = \frac{1}{2} * \frac{1}{n} * \sum_{i=1}^{n} (y^i - x^i w)^2$

Definition of Euclidean norm or norm2: $\|z\|_2 = (z_1^2 + z_2^2 + ... + z_n^2)^{\frac{1}{2}} \Longrightarrow \|z\|_2^2 = (z_1^2 + z_2^2 + ... + z_n^2)$

$\Longrightarrow L = \frac{1}{2} * \frac{1}{n} \|y - \hat{y}\|_2^2 = \frac{1}{2} * \frac{1}{n} * (y - \hat{y})^T * (y - \hat{y})$
Now, we have already know the general formula of the loss function L Linear Regression in term of Linear Algebra. To completely solving the Linear Regression problems, we must try to find the global minimum of loss function L in which the parameters w is unknown. In the other words, we must find the optimized parameters w which is a vector such that the value of L is minimum.

However, the min value of Loss function L will be the similar with $\frac{L}{N}$ so we'll find the minimum of function L_1:

$$L_1 = \frac{1}{2} * (y - \hat{y})^T * (y - \hat{y})$$

Take the derivative on both sides of function L_1 with respect to w we have:
$\Rightarrow \frac{d(L)}{d(w)} = \frac{d(\frac{1}{2}*(y-\hat{y})^T*(y-\hat{y}))}{d(w)}$
$\Leftrightarrow \frac{d(L_1)}{d(w)} = \frac{d(\frac{1}{2}*(y-(Xw))^T*(y-Xw))}{d(w)}$
$\Leftrightarrow \frac{d(L_1)}{d(w)} = \frac{1}{2} * 2 * (-X^T) * (y - Xw)$

$\Leftrightarrow \frac{d(L_1)}{d(w)} = X^T X w - X^T y$

To find the minimum, let the left hand side be zero:
$\Leftrightarrow 0 = X^T X w - X^T y$
$\Leftrightarrow X^T X w = X^T y$
Assume that, $X^T X$ is invertible matrix:
$\Leftrightarrow (X^T X)^{-1}(X^T X)w = (X^T X)^{-1}(X^T y)$
$\Leftrightarrow Iw = (X^T X)^{-1}(X^T y)$
$\Leftrightarrow w = (X^T X)^{-1}(X^T y)$

Conclusion, the Loss function L_1 or L reach the minimum at $w = (X^T X)^{-1}(X^T y)$ if and only if $X^T X$ is invertible.

3.3 Code Implementation

3.3.1 Import Libraries

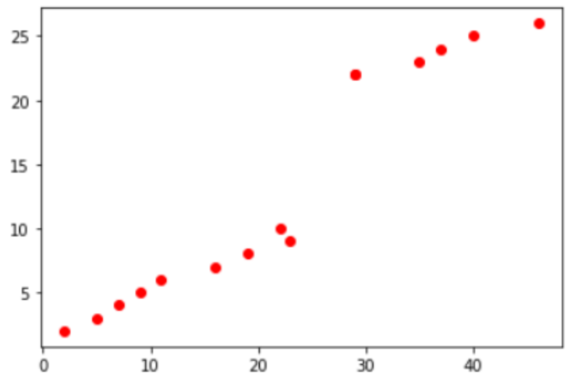
We'll import two important libraries: numpy and mathplotlib .

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

3.3.2 Data visualization

To make it easy, we just choose $X \in R^{n*(m+1)}$ where n = 12 (observations) and m = 1 (feature) so that we can draw them in 2 dimensional space.

```
X = [2,5,7,9,11,16,19,23,22,29,29,35,37,40,46]
y = [2,3,4,5,6,7,8,9,10,22,22,23,24,25,26]
plt.plot(X,y,'ro')
```



Data visualization

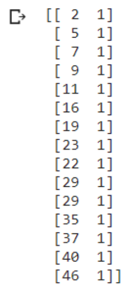
Transfer the data type of X and y to be numpy array for re-using the Algebra functions of numpy library.

```
X = np.array([A]).T
y = np.array([b]).T
```

3.3.3 Double Check the data

In here we see that $X \in R^{n*m}$ is not the standard form which should belong to $R^{n*(m+1)}$. We solve it by concatenating the "ones" vector, containing 1s, to the tail of X. So that, X can be describe as: X =
$$\begin{bmatrix} x_1^{(1)}, x_2^{(1)}, ..., x_m^{(1)}, 1 \\ x_1^{(2)}, x_2^{(2)}, ..., x_m^{(2)}, 1 \\ \vdots \\ x_1^{(n)}, x_2^{(n)}, ..., x_m^{(n)}, 1 \end{bmatrix}$$

```python
ones = np.ones(X.shape, dtype= np.int8)
X = np.concatenate((X,ones),axis=1)
print(X)
```



```
[[ 2  1]
 [ 5  1]
 [ 7  1]
 [ 9  1]
 [11  1]
 [16  1]
 [19  1]
 [23  1]
 [22  1]
 [29  1]
 [29  1]
 [35  1]
 [37  1]
 [40  1]
 [46  1]]
```

X content

3.3.4 Applying the Algebra formula for Linear Regression

After make sure that the data (matrix X and vector y) are in the right form. We'll using the final equation to find out the w.

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

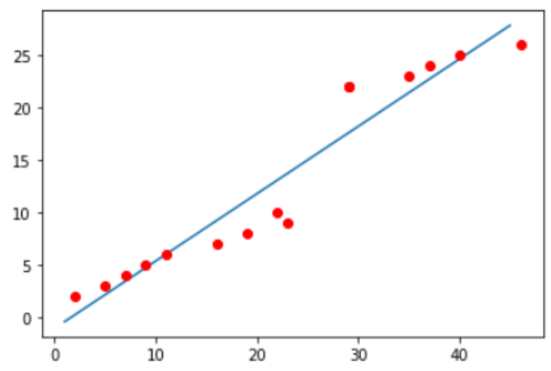
```python
w = np.linalg.inv(X.transpose().dot(X)).dot(X.transpose()).dot(y)
```

In here, w[0][0] will contains weight or coefficient, w[1][0] contains the intercept.

3.3.5 Visualize the Linear Regression model

After finding out the optimized weights and bias, we should test and evaluate the Linear Regression model.

```python
predict_points = np.array([1,23,45]).T
predict_results = predict_points*w[0][0] + w[1][0]
print(predict_results)
plt.plot(predict_points,predict_results)
plt.plot(X,y,'ro')
plt.show()
```



Linear Regression model visualization

Briefly, the green line is a suitable line to perform all the points in our data set, which lead to the minimum value of loss function

$$L = \frac{1}{2} * \frac{1}{n} * (y - \hat{y})^T * (y - \hat{y})$$

.

3.4 Conclusion

Solving Linear Regression by Linear Algebra help us to find exactly the most optimized weights and bias by applying the just only one formula:

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

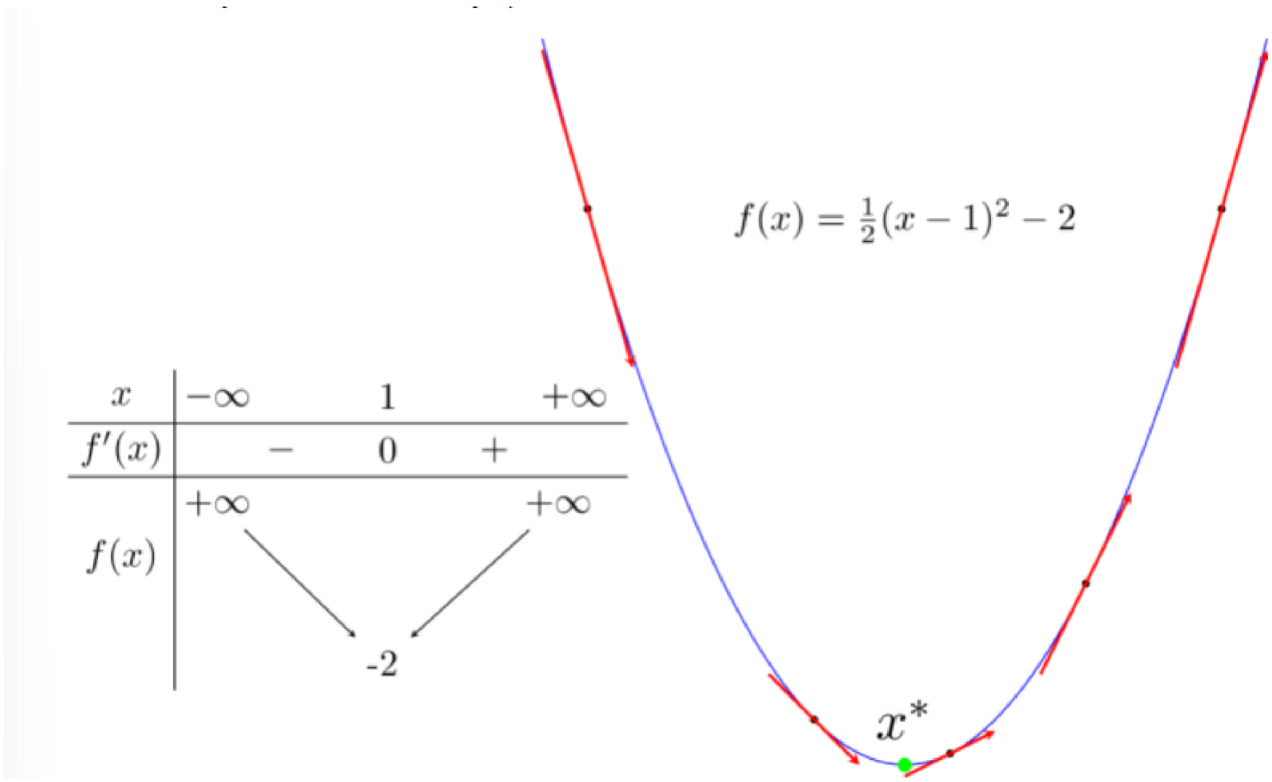
However,This way is not always the best way because when computer work with the matrix function such as dot product, inverse matrix, transpose matrix, it consume a lot of resource than when work with scalar numbers.Considering the case that n - number of observations is a very large numbers (such as 10^4 observations) and m - number of features for each observation is large also (m $>$ 20), computer can take up to several hours to inverse, transpose the $X \in R * n * (m + 1)$ matrix and perhaps a day to completely execute the above formula. Sometimes when the data is extensively abundant, it must to connect to severs of super-computer to find w because the personal computer has limited resource. The other way to solve the Linear Regression called Gradient Descent ,which has strong relation with Calculus instead of Algebra, will be detailed in next section.

4 Gradient Descent

4.1 Overview

In Machine Learning, we often consider the problem to minimize or maximize the value, for example lost function of Linear Regression. Also in its perspective, sometimes the task of finding the global extrema of the lost shows to be very complicate or impossible. Local extremas is chose instead. In some points, local extremas show to be very close to the value of global extrema but obviously, much easier to find with a lot of algorithms that already have been proposed. For those reason, solution found at local minimum can be used at the solution to the problem.
In practice, data usually spreads over a large range with numerous data points with a large value for dimension. The method which starts from one point we consider as the solution, iterate a number of steps so it moves close to the real solution or let say the derivative of that point is very close to 0, is called Gradient Descent (or GD for short).

4.2 Methodology



Hình 3: GD example

In the example above, the green dot is the local minimum that we wish to achieve. At that point, the derivative is computed to be 0. GD, in the name of the algorithm give us the flow to address as follow. Suppose we choose a point that lie on the right curve $(8, 30)$ and we are going to get x^*. The function $f(x)$ has the derivative $f'(x)$:

$$f'(x) = x - 1$$

With $x = 8$ in the data point above, we can compute the derivative of that point to be $7 > 0$ or $f'(x) > 0$. We wish to achieve the point x^* which is going down the curve, opposite direction of the trend on that side.

$$x_{t+1} = x_t + \Delta$$

Δ is opposite sign to $f'(x_t)$ We can think of a new equation to be more simple:

$$x_{t+1} = x_t - \eta f'(x_t)$$

η (eta) is a positive number also called as *learning rate*. The minus sign shows that we need to go down the trend here, in this case go left, if in the initial step, we choose a point that the x component is -8 for example. We will get the data point lie on the left curve and in this case we will need to go right, down to the point we wish. The reason why this algorithm is called GD is that when running, each step will change a little bit in the x component so that it can approximate x^* each time (Gradient), the word "Descent" stands for the work to go in the opposite direction of the derivative at that point. The example above is just a simple clarify to the method.

4.3 Python code

To keep up with the idea, we will consider the function that is mentioned above:

$$f(x) = \frac{1}{2}(x-1)^2 - 2$$

This function has the derivative:

$$f'(x) = x - 1$$

Suppose that an arbitrary point is chosen x_0, follow a number of iteration and result in the value of x_t. By using GD, the next step to be updated will be:

$$x_{t+1} = x_t - \eta(x - 1)$$

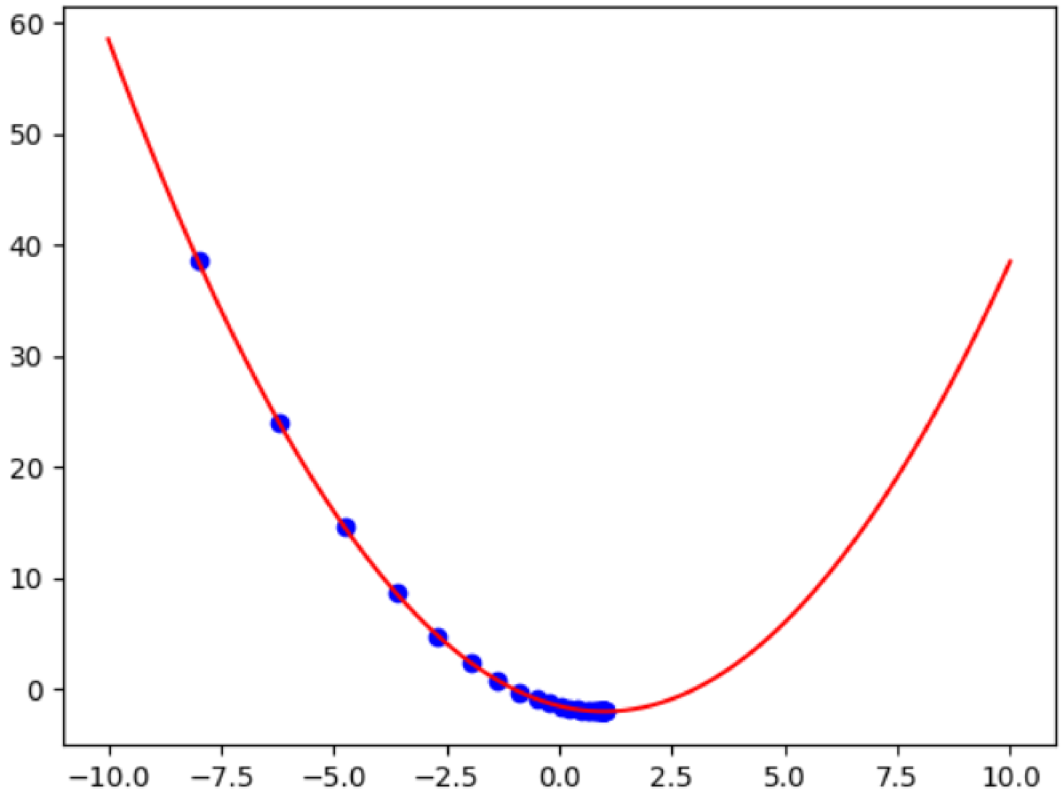
With η is the learning rate, in this example, η will be set to 0.1 so that the going down step is not too large or too small.

```python
import numpy as np
import matplotlib.pyplot as plt

eta = 0.1 # Learning rate

def cost(x):
    return 0.5 * (x-1)**2 - 2

def grad(x):
    return x - 1

def GD(x0):
    x_list = [x0]
    for i in range(10000):
        x_step = x_list[-1] - eta * grad(x_list[-1])
        if abs(grad(x_step)) < 0.001:
            break
        x_list.append(x_step)
    return x_list, i
```

The **cost(x)** function and **grad(x)** implement the function and its derivative. To compute the Gradient Descent, we will apply the formula above but give it a threshold (0.001) where it can stop.

```python
x = np.linspace(-10, 10, 200)
y = 0.5 * (x-1)**2 - 2

plt.plot(x, y, 'r')
x_list1, i1 = GD(-8)
x_list2, i2 = GD(8)
print(i1, i2)
for i in range(len(x_list1)):
    fx = cost(x_list1[i])
    plt.scatter(x_list1[i], fx, color='blue')
plt.show()
```

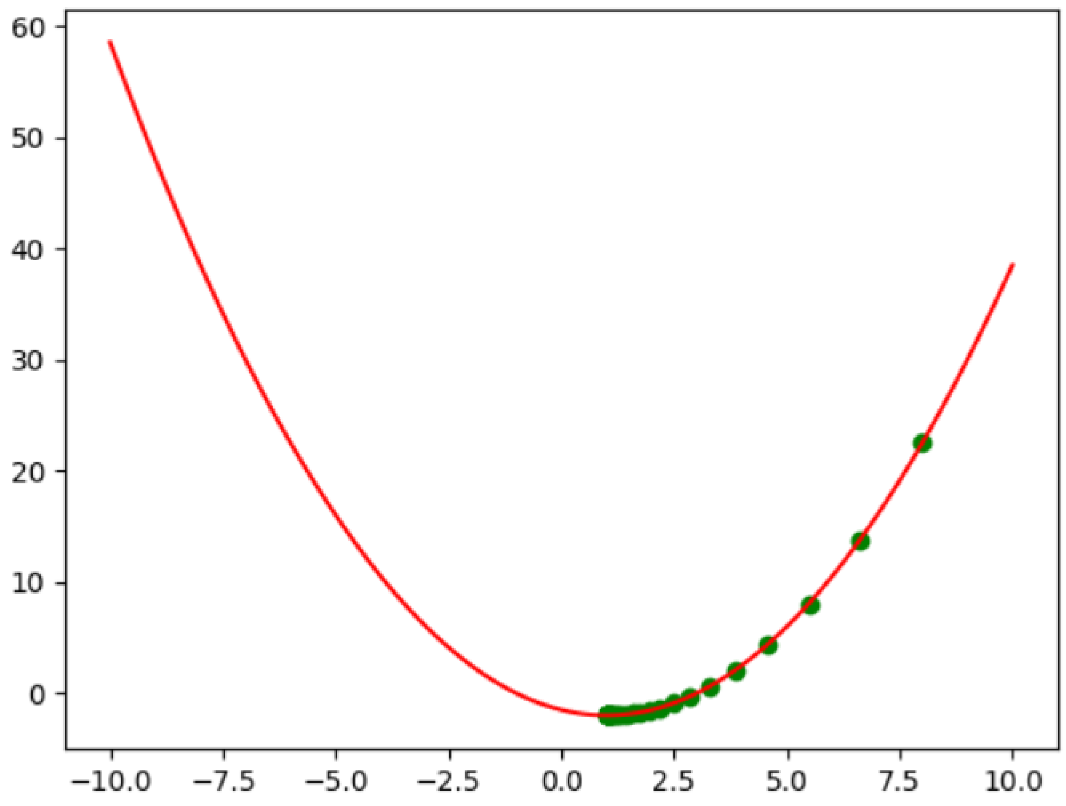
Output



As we can see the plot, we start from the point $x = -8$ and we go down the curve with each step is smaller and smaller till it reaches the threshold and stop iterate.
One more example for the right side of the graph to be seen below.

```python
for i in range(len(x_list2)):
    fx = cost(x_list2[i])
    plt.scatter(x_list2[i], fx, color='green')
plt.show()
```

Output



4.4 Gradient Descent with numerous points

The sections above have shown how GD is implemented and how it works with only one variable of x change over time. When it comes to multiple data points, the merits of using GD tend to overflow. When we have multiple data points or we can say a big matrix with a number of rows and columns, Linear Regression for example, computer only can compute a simple matrix because of its resources, when computers encounter a dataset can we say 100000 points, the work is getting slower and slower. GD is used to solve this drawback. We need to find the global minimum for the function $f(\theta)$ with θ (theta) is a vector consists of data points for the optimization. The derivative of this function is

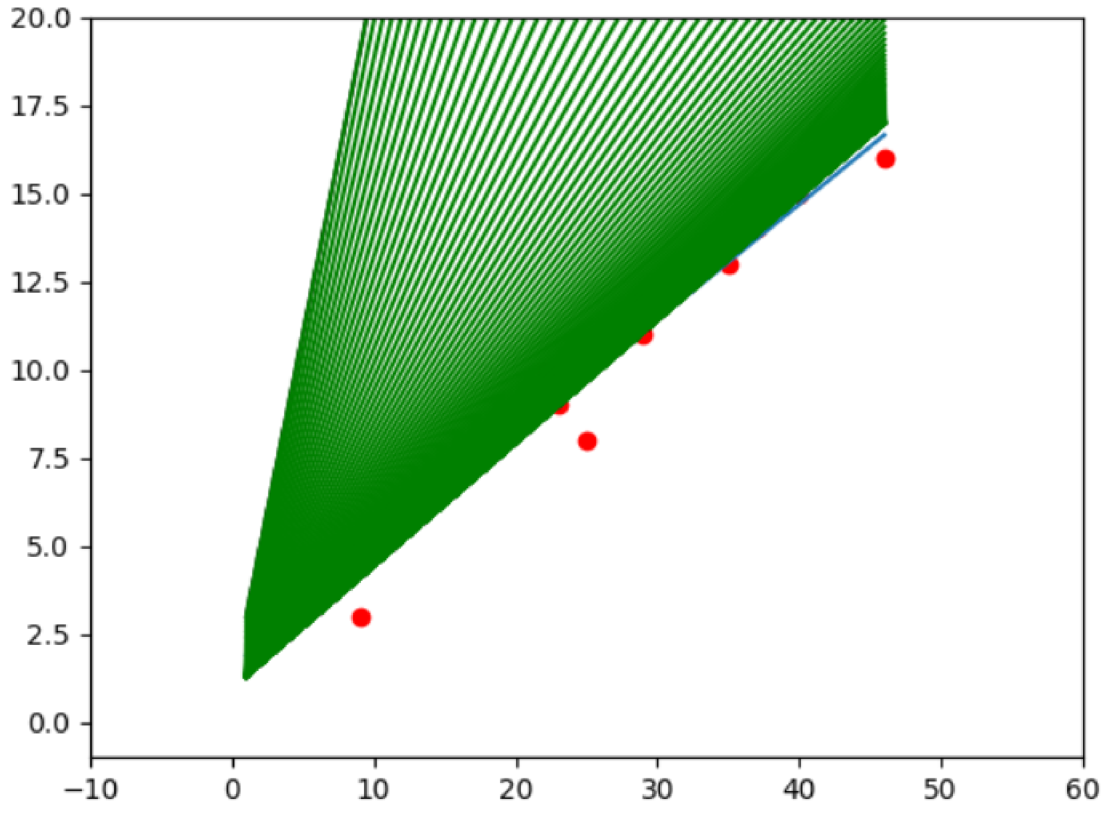
$$\nabla_\theta f(\theta)$$

. The formula takes after the formula for one variable above:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)$$

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model


def cost(x):
    m = A.shape[0]
    return 0.5 / m * np.linalg.norm(A.dot(x) - b, 2) ** 2


def grad(x):
    m = A.shape[0]
    return 1 / m * A.T.dot(A.dot(x) - b)


def gradient_descent(x_init, learning_rate, iterations, A):
    x_list = [x_init]
    for i in range(iterations):
        x_new = x_list[-1] - learning_rate * grad(x_list[-1])
        x_list.append(x_new)

    return x_list


# Data
A = np.array([[2,9,7,9,11,16,25,23,22,29,29,35,37,40,46]]).T
b = np.array([[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]]).T

fig1 = plt.figure("GD for Linear Regression")
ax = plt.axes(xlim=(-10, 60), ylim=(-1, 20))
plt.plot(A, b, 'ro')
# plt.show()

lr = linear_model.LinearRegression()
lr.fit(A, b)
x0_gd = np.linspace(1, 46, 2)
y0_sklearn = lr.intercept_[0] + lr.coef_[0][0] * x0_gd
plt.plot(x0_gd, y0_sklearn)

inCo = np.array([[1], [2]])
y0_init = inCo[0][0] + inCo[1][0] * x0_gd
plt.plot(x0_gd, y0_init, color="black")

ones = np.ones((A.shape[0], 1), dtype=np.int8)
A = np.concatenate((ones, A), axis=1)

iterations = 200
learning_rate = 0.0001

x_list = gradient_descent(inCo, learning_rate, iterations, A)
for i in range(len(x_list)):
    y0_x_list = x_list[i][0] + x_list[i][1] * x0_gd
    plt.plot(x0_gd, y0_x_list, color="green")

plt.show()
```

Output:



The code above is the example for the idea to use GD and approximate the true fitting line. There are a little of error regardless to iterations have threshold but in a point of view, the solution is good enough as it can predict the data value very close to what we have with Linear Regression by using matrix multiplication.

4.5 Conclusion

Gradient Descent is by far the most popular optimization strategy used in machine learning and deep learning at the moment. It is used when training data models, can be combined with every algorithm and is easy to understand and implement. Everyone working with machine learning should understand its concept. Gradient Descent is a old algorithm and there are many variants like: Batch Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent. Each of them can be use to address some specific problems in Machine Learning which you may encounter in the next step of digging deep into this field. Stay tuned and keep being optimistic.

5 K Nearest Neighbor

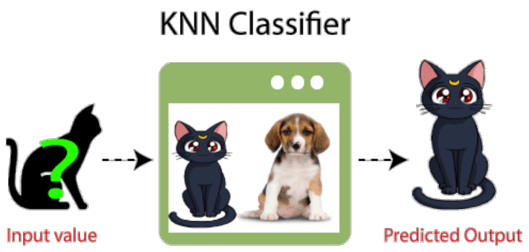
5.1 Introduction

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry.
KNN has two properties:

- **Lazy learning algorithm** - KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** - KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



5.2 Mathematical perspective

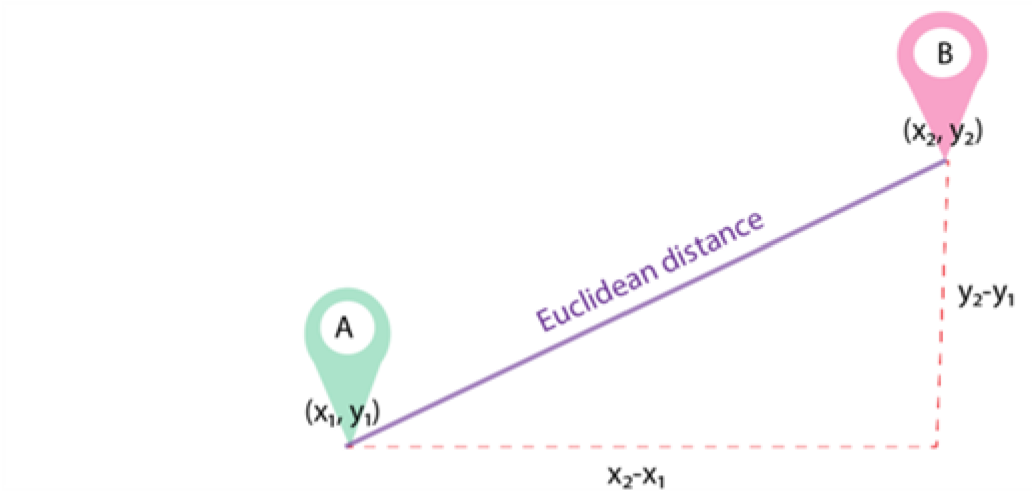
The KNN algorithm works by finding the distance between the mathematical values of data points. It computes the distance between each data point and the test data and then finds the probability of the points being similar to the test data. Classification is based on which points share the highest probabilities. The distance function can be Euclidean, Minkowski or the Hamming distance. However, we will figure out about Euclidean and Minkowski distance only in this report.

5.2.1 Euclidean Distance Function

Euclidean distance can simply be defined as the shortest between the 2 points irrespective of the dimensions. The most common way to find the distance between is the Euclidean distance. According to the Euclidean distance formula, the distance between two points in the plane with coordinates (x, y) and (a, b) is given by

$$dist((x,\ y),\ (a,\ b))\ = \sqrt{(x-a)^2+(y-b)^2}$$

To visualize this formula, it would look something like this:



For a given value of K, the algorithm will find the k-nearest neighbors of the data point and then it will assign the class to the data point by having the class which has the highest number of data points out of all classes of the K neighbors.
After computing the distance, the input x gets assigned to the class with the largest probability:

$$P(y = j|X = x) = \frac{1}{K}\sum_{i\in\mathcal{A}} I(y^{(i)} = j)$$

5.2.2 Minkowski Distance Function

Minkowski distance is a distance measurement between two points in the normed vector space (N-dimensional real space) and is a generalization of the Euclidean distance.
Consider two points P1 and P2:

$$P1 : (X1, X2, ..., XN), P2 : (Y1, Y2, ..., YN)$$

Then, the Minkowski distance between P1 and P2 is given as:

$$\sqrt[p]{(x1 - y1)^p \; + \; (x2 - y2)^p \; + \; ... \; + \; (xN - yN)^p}$$

When p=2, Minkowski distance is same as Euclidean distance.

5.3 Algorithm

The KNN method predicts the values of new datapoints using 'feature similarity,' which implies that the new data point will be assigned a value depending on how closely it matches the points in the training set.

There are 4 steps to do this method:

1. For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.
2. Then, we need to choose K, which is the number of nearest data points. K can be any integer.
3. For each point in the test data do the following:

(a) Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.

(b) Based on the distance value, sort them in ascending order.

(c) Next, choose the top K rows from the sorted array.

(d) Finally, assign a class to the test point based on most frequent class of these rows.

4. Our model is ready.

How to select the value of K in the K-NN Algorithm? Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

5.4 Python Implementation

5.4.1 KNN as Classifier

There are two examples for KNN classification, which are iris flower classification and digit classification.

5.4.1.1 Iris flower classification:

In this problem, we code from scratch

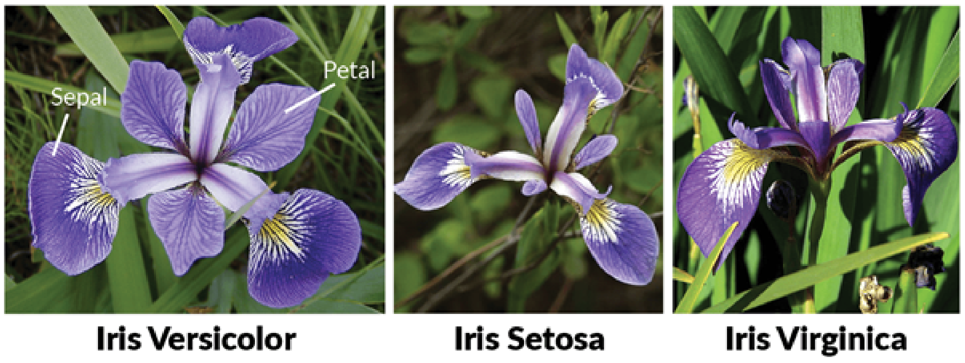
1. Step 1: First, start with importing necessary python packages

```python
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
import math
import operator
```

Next, we import iris dataset from datasets:

```python
iris = datasets.load_iris() #150 data
iris_X = iris.data #data, each has 4 domains, which are sepal length, sepal width, petal
    length, petal width respectively
iris_Y = iris.target #label(0, 1, 2)
```

Categories type of iris flower.
Here is the picture of label 0, 1, 2 of iris versicolor, iris setosa, iris virginica respectively.



Next, we need to shuffle our data, so it can distribute randomly

```python
randIndex = np.arange(iris_X.shape[0])
np.random.shuffle(randIndex)
```

Then, we have new data after shuffling

```python
iris_X = iris_X[randIndex] #data
iris_Y = iris_Y[randIndex] #label
```

Next, we split training data and test data:

- The first way is taking the first 100 data of iris_X into X_train, and the last 50 data of iris_X into X_test. The same for iris_Y

```python
X_train = iris_X[:100,:]
X_test = iris_X[100:,:]
y_train = iris_Y[:100]
y_test = iris_Y[100:]
```

- The second way is taking randomly 100 data of iris_X into X_train, randomly 50 data of iris_X into X_test, and the same for iris_Y by using train_test_split()

```python
X_train, X_test, Y_train, Y_test = train_test_split(iris_X, iris_Y, test_size = 50) #test_size is the number of test data, iris_X and iris_Y must have the same dimension
```

2. Step 2: We choose k = 5
3. Step 3: We need to implement the following function: This function return the distance of 2 points p1, p2.

```python
def calculate_distance(p1, p2):
   dimension = len(p1) #the number of dimension
   dis = 0 #distance
   for i in range(dimension):
      dis += (p1[i] - p2[i])**2
   return math.sqrt(dis)
```

This function will return k points' label that nearest to the point we need to predict by sorting training points base on their distance to predict point, then take label of k nearest point we have just found.

```python
def get_k_neighbors(training_x, label_y, point, k):
   distances = []
   neighbors = []

   for i in range(len(training_x)):
      distance = calculate_distance(training_x[i], point)
```

```
        distances.append((distance, label_y[i]))

    #first sorting way
    distances.sort(key = operator.itemgetter(0)) #sort by distance
    for i in range(k):
        neighbors.append(distances[i][1]) #label of k min distance

    #second sorting way
    index = []
    while len(neighbors) < k:
        i = 0
        min_distance = 999999
        min_idx = 0
        while i < len(distances):
            if i in index:
                i += 1
                continue
            if distances[i] <= min_distance:
                min_distance = distances[i]
                min_idx = i
            i += 1
        index.append(min_idx)
        neighbors.append(label_y[min_idx])

    return neighbors
```

This function return the label with the highest appearance.

```
def highest_votes(labels):

    labels_count = [0,0,0]
    for label in labels:
        labels_count[label] += 1
    max_count = max(labels_count)
    return labels_count.index(max_count) #return the label that appear the most
```

This function helps to calculate the accuracy of KNN method in this problem.

```
def accuracy_score(predict, labels):
    total = len(predict) #so luong du doan
    correct_count = 0
    for i in range(total):
        if predict[i] == labels[i]:
            correct_count += 1
    accuracy = correct_count/total
    return accuracy
```

4. Step 4: Our model is ready to use

```
y_predict = []
for p in X_test:
    label = predict(X_train, y_train, p, k)
    y_predict.append(label)

acc = accuracy_score(y_predict, y_test)
print(acc)
```

Output

```
0.98
```

5.4.1.2 Digit classification: In this problem, we use package to solve it.

1. Step 1:
   Firstly, we import importing necessary python packages:

```python
from sklearn import datasets, neighbors
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

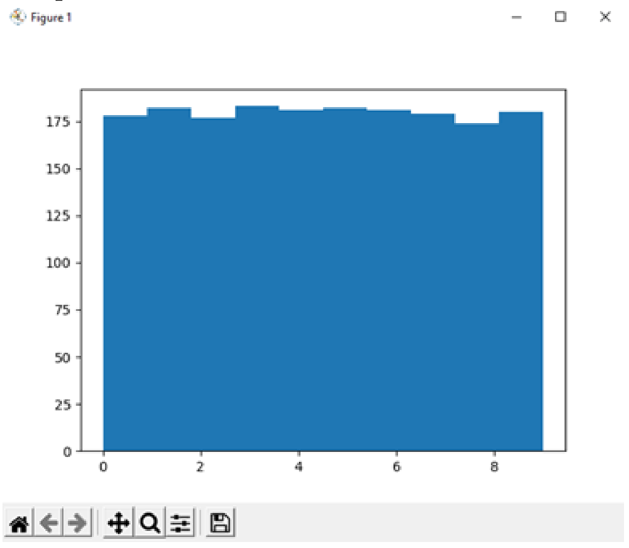
Next, we import digit dataset from sklearn.datasets, which is called MNIST

```python
digit = datasets.load_digits()
digit_X = digit.data # datapoint, consist of 1797 pictures, each has size 8x8
digit_Y = digit.target # label
```

We can check the distribution of numbers by drawing histogram:

```python
#distribution of numbers
plt.hist(digit_Y)
plt.show()
```

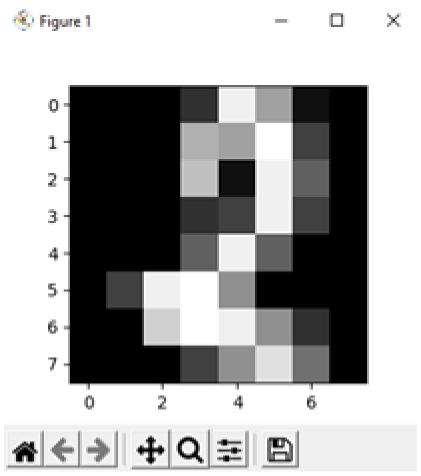
Output



We also can plot any image to see such as a picture of number 2:

```python
# print out picture to see
img1 = digit_X[10]
plt.gray() #anh trang den
plt.imshow(img1.reshape(8,8))
plt.show()
```

Output



Next, we shuffle our data:

```
randIndex = np.arange(digit_X.shape[0])
np.random.shuffle(randIndex)

digit_X = digit_X[randIndex]
digit_Y = digit_Y[randIndex]
```

Then, we split training set and test set. Because there are 1797 datapoints so I take 20% of them ≈ 360 to be test set.

```
X_train, X_test, y_train, y_test = train_test_split(digit_X, digit_Y, test_size = 360)
```

2. Step 2: Choose K = 5.
3. Step 3: Train model.

```
    knn = neighbors.KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
Y_predict = knn.predict(X_test)
result = accuracy_score(Y_predict, y_test)
print("Accuracy: ", result)
```

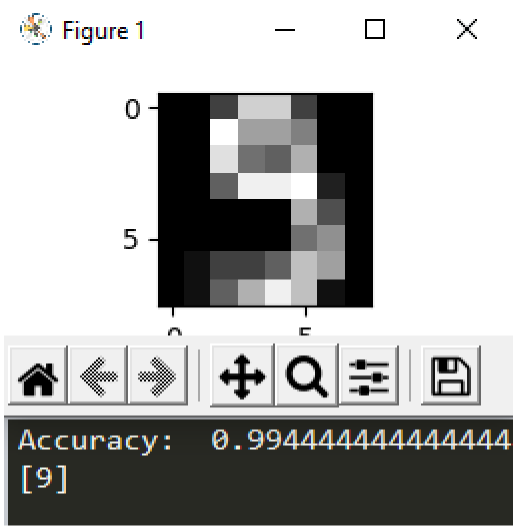
Output

```
    Accuracy:  0.9888888888888889
```

4. Step 4: Our model is ready to use.
We can test it with any image, for example:

```
    print(knn.predict(X_test[0].reshape(1,-1))) #reshape(1,-1) to put data into a list.
    Ex: [1] become [[1]]
plt.gray()
plt.imshow(X_test[0].reshape(8,8))
plt.show()
```

Output



```
Accuracy:  0.9944444444444444
[9]
```

Comment: The KNN algorithm, when used for digit recognition, has a limitation that when 2 pictures are the same number 1 but 1 image number 1 in the left corner, and 1 image number 1 in the right corner, it will be evaluated as not close to each other.

5.4.2 KNN as Regressor

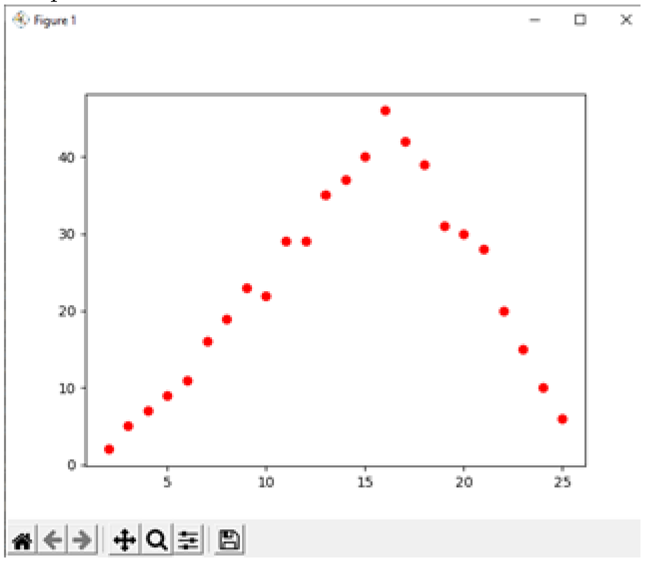
In this part, assume we have a set of points with coordinates Ox, Oy as follows:

```
X = [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]
Y = [2,5,7,9,11,16,19,23,22,29,29,35,37,40,46,42,39,31,30,28,20,15,10,6]
```

We want to estimate with a given X, what value of Y will be?
We can plot those points for a better visualization

```
plt.plot(X, Y, 'ro')
plt.show()
```

Output



1. Step 1:
   Firstly, we import necessary packages:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import neighbors
```

   Next, we format our data into 2d array:

```
X = np.array(X).reshape(-1, 1) # [[2] [3] [4]], but reshape(1, -1) be [[2 3 4]]
Y = np.array(Y)
```

   Next, we use a test dataset:

```
x0 = np.linspace(3, 25, 10000).reshape(-1, 1)
y0 = []
```

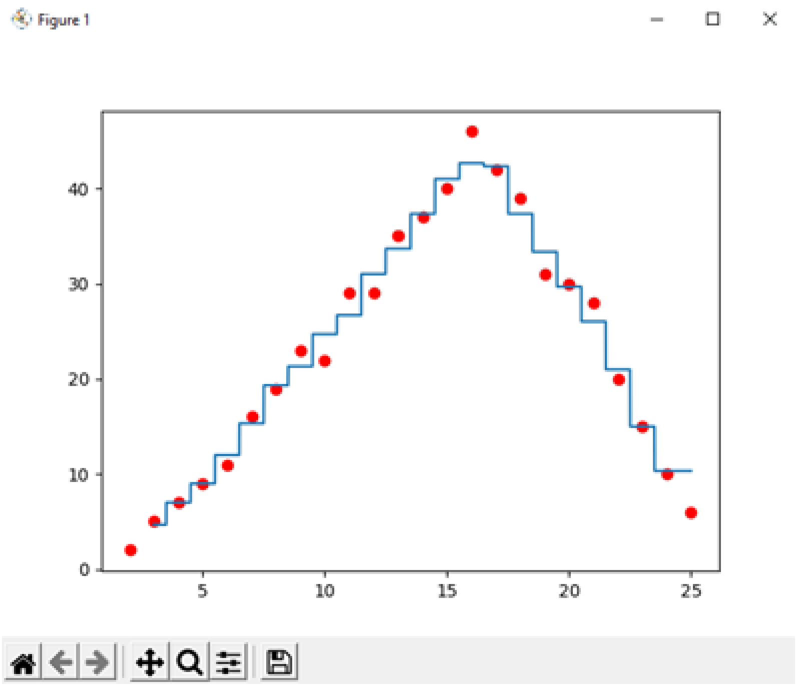
2. Step 2: Choose k = 3
3. Step 3: Train the model

```
knn = neighbors.KNeighborsRegressor(n_neighbors = 3)
knn = knn.fit(X, Y)
```

4. Step 4: Our model is ready to use
   For example, we want to predict every y0 value corresponding to x0 value and plot them

```
y0 = knn.predict(x0)
plt.plot(x0, y0)
plt.show()
```

   Output



   We can also predict a single value y with our model

```
test = [[15.6]]
predict_test = knn.predict(test)
print(predict_test)
```

Result

```
[42.66666667]
```

Comment: If there are outliers in the data set, it will skew the results a lot. Whether that's good or bad depends on the problem.

5.5 Conclusion

That is all we need to understand K-nearest neighbors algorithm. In order to have a clearer view of KNN method, let us compare advantages and disadvantages of it.
Pros

- It is a simple algorithm to understand and interpret
- It is very useful for nonlinear data because there is no assumption about data in this algorithm
- It is a versatile algorithm as we can use it for classification as well as regression
- It has relatively high accuracy but there are much better supervised learning models than KNN

Cons

- It is computationally a bit expensive algorithm because it stores all the training data
- High memory storage required as compared to other supervised learning algorithms
- Prediction is slow in case of big N
- It is very sensitive to the scale of data as well as irrelevant features

6 Conclusion

To recapitulate briefly, in this report we mainly focus on most three fundamental Machine Learning algorithms:

- K-Means: an unsupervised classification algorithm which allows the users to divide the dataset into any number of groups as they wish> Additional, we also report about the application of K-Means in Image Compression.
- K-Nearest Neighbor: a supervised classification or regression algorithm. This is one the most special algorithm since when its *lazy learing*. In the other words, when training it totally do nothing but when testing period, it do a lot of computaions with high complexity.
- Linear Regression: a supervised algorithm which is used mostly in regression but not classification. This alogrithm can be solved by two different ways: Linear Algebra or Gradient Descent (Calculus). In which the Linear Algebra can solve and give more correct outputs than Gradient. In contrast, the Gradient Descent take less of computer resource in the training model period when compared to the Linear Algebra solution.

Besides that, the report also help you have insights into "How a picture is constructed ?", the deeper meaning of derivative or optimization problems.Although these three algorithms are not the most powerfull and highly applicated in nowadays Machine Learning, they are important fundamental for the beginers to learning more complicate Machine Learning algoritms such as Lasso Regression, Grid Regression.

Author

The Four Horsemen of the Apocalypse