Trí tuệ tính toán và ứng dụng

PGS.TS. Trần Văn Lăng

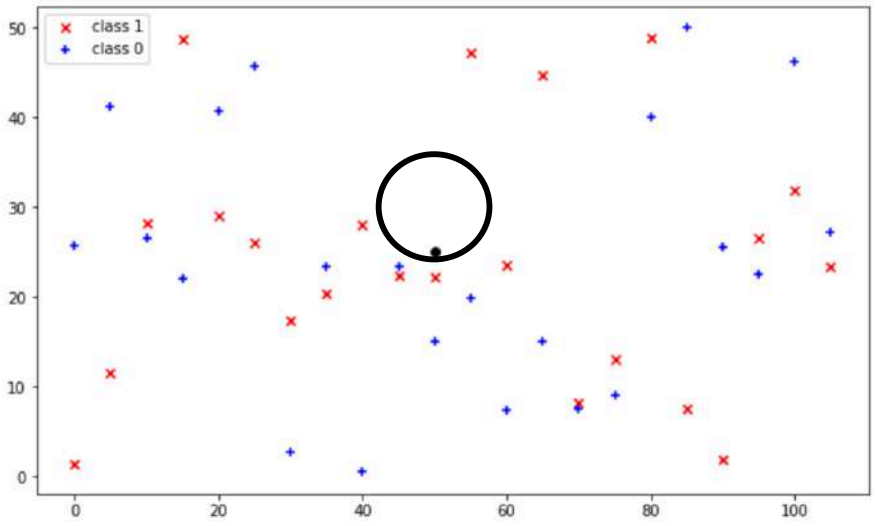
Thuật toán *K*-láng giềng gần nhất

- Tên tiếng Anh là KNN (K-Nearest Neighbor),
- Là thuật toán thuộc loại supervised-learning đơn giản nhất
- KNN có thể áp dụng được cho cả Classification và Regression
- Khi training, KNN không học một điều gì từ dữ liệu, mọi tính toán được thực hiện khi dự đoán kết quả của dữ liệu mới

- Với ***việc phân lớp***, nhãn (đầu ra) của một điểm dữ liệu (Data Point) mới được suy ra từ K điểm dữ liệu gần nhất trong tập huấn luyện.
- Với ***hồi quy***, đầu ra của một điểm dữ liệu bằng chính đầu ra của điểm dữ liệu đã biết gần nhất (trong trường hợp K=1)
- **Tóm tại**, KNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách chỉ dựa trên thông tin của K điểm dữ liệu trong tập huấn luyện gần nó nhất (K-lân cận), mà không quan tâm đến việc có một vài điểm dữ liệu trong những điểm gần nhất này là nhiễu (dữ liệu mà có lời giải sai)

Minh hoạ

- Giả sử có một dataset gồm 2 loại dữ liệu là **màu đỏ (class 1)** và **màu xanh (class 0)** như hình bên cạnh.
- Thuật toán *K*-NN hãy tìm điểm (50,25) – có **hình dấu O màu đen** – để biết điểm này thuộc **class 1** hay **class 2**
- Cách làm:
  - Tính khoảng cách từ **điểm O** đến tất cả các điểm có trong Data Set.
  - Sau đó chỉ ra *K* điểm gần (ở đây lấy *K* = 3) với điểm **dấu O** này nhất



- Có 2 điểm thuộc **class 1**, và 1 điểm thuộc **class 0**.
- Vậy **điểm O** này thuộc **class 1**

File: Draw-graph.ipynb

- Chương trình giải quyết vấn đề này như sau:
  - Tạo dữ liệu

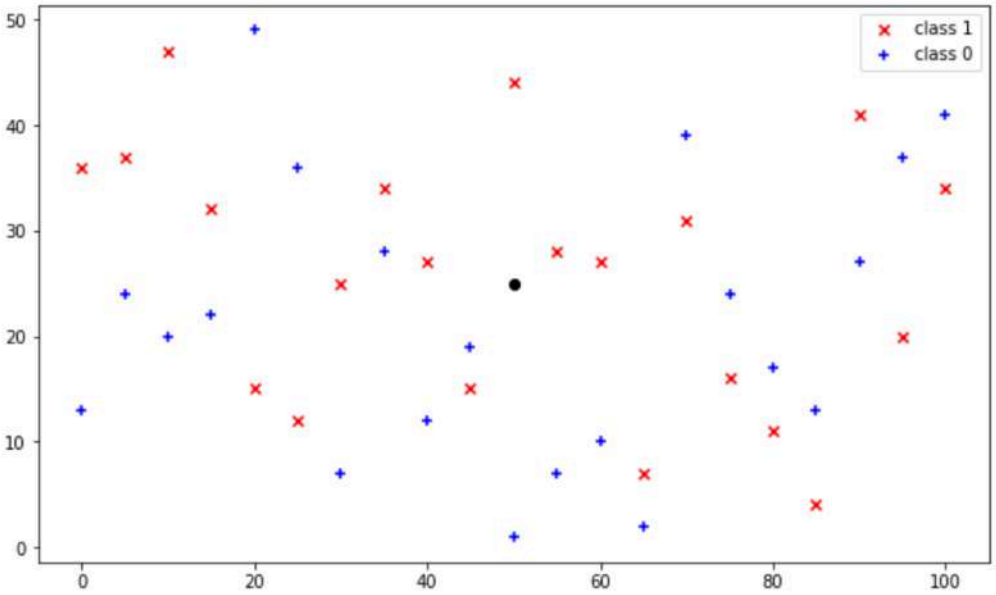
```
import random
N, M = 21, 2
X1, X2, Y1, Y2 = [], [], [], []
for i in range(0, N):
    X1.append([]), X2.append([])
    X1[i].append(i*5)
    X1[i].append( int(random.random()*50) )
    X2[i].append(i*5)
    X2[i].append( int(random.random()*50) )
    Y1.append( 1 )
    Y2.append( 0 )
y = Y1 + Y2
X = X1 + X2
```

  - Tạo dữ liệu

```
x = np.arange( 0,N*5,5 )
y1, y2 = [], []
for i in range(0,N):
    y1.append( X[i][1] )
    y2.append( X[i+N][1] )
```

  - Vẽ hình

```
plt.figure( figsize=(10,6) )
plt.scatter( x,y1,c='r',marker='x',label='class 1' )
plt.scatter( x,y2,c='b',marker='+',label='class 0' )
plt.plot( 50, 25, 'o', color="black" )
plt.legend()
plt.show()
```




```
1 from sklearn import neighbors
2 model = neighbors.KNeighborsClassifier( n_neighbors=3 )
3 model.fit( X,y )
4 print( "Thuộc lớp: ", model.predict( [[50,25]]) )
5 print( "với xác suất là ", model.predict_proba( [[50,25]]) )
```

```
Thuộc lớp:  [1]
với xác suất là  [[0.33333333 0.66666667]]
```

- Huấn luyện và dự báo

```
from sklearn import neighbors
model = neighbors.KNeighborsClassifier( n_neighbors=3 )
model.fit( X,y )
print( "Thuộc lớp: ", model.predict( [[50,25]]) )
print( "với xác suất là ", model.predict_proba( [[50,25]]) )
```

File: Draw-graph.py

Định nghĩa về khoảng cách

- Cho 2 vector X, Y có các thành phần là $X = (x_1, x_2, \ldots, x_n)^T$, $Y = (y_1, y_2, \ldots, y_n)^T$
- Khoảng cách bậc p của 2 vector này là con số được ký hiệu là, người ta còn gọi là khoảng cách Miskowsky

$$\|X - Y\|_p = \left(\sum_{i=1}^{N} |x_i - y_i|^p \right)^{1/p}$$

- Trong một số trường hợp riêng
- Trong một số trường hợp riêng, có
  - Khoảng cách Euclide (hay là chuẩn L_2)

$$\|X - Y\|_2 = \sqrt{\sum_{i=1}^{N} |x_i - y_i|^2}$$

  - Khoảng cách tuyệt đối ($p = 1$) hay còn gọi là khoảng cách Hamming, hay Manhattan

$$\|X - Y\| = \sum_{i=1}^{N} |x_i - y_i|$$

Ví dụ KNN

- Khoảng cách Chebyshev

$$\|X - Y\| = \lim_{p \to \infty} \|X - Y\|_p$$

$$= \left(\sum_{i=1}^{N} |x_i - y_i|^p \right)^{1/p}$$

$$= \max_{1 \le i \le n} |x_i - y_i|$$

- Lấy dữ liệu bệnh tiểu đường của người Ấn Độ (https://www.kaggle.com/uciml/pima-indians-diabetes-database)
- Dữ liệu bao gồm:
  - Pregnancies: Số lần mang thai
  - Glucose: Lượng Glucose
  - BloodPressure: Áp huyết tâm trương
  - SkinThickness: Về da
  - Insulin: Nồng độ insulin
  - BMI: Body mass index (tỷ lệ giữa trọng lượng và chiều cao)
  - DiabetesPedigreeFunction: loại tiểu đường
  - Age: Tuổi
  - Outcome: Kết quả 1 hoặc 0

Python với sci-kit learn

- Tạo mô hình và thử nghiệm

```
NUM_K = 3
model = neighbors.KNeighborsClassifier( n_neighbors=NUM_K,p=2 )
model.fit( X_train,y_train )
y_pred = model.predict( X_test )
```

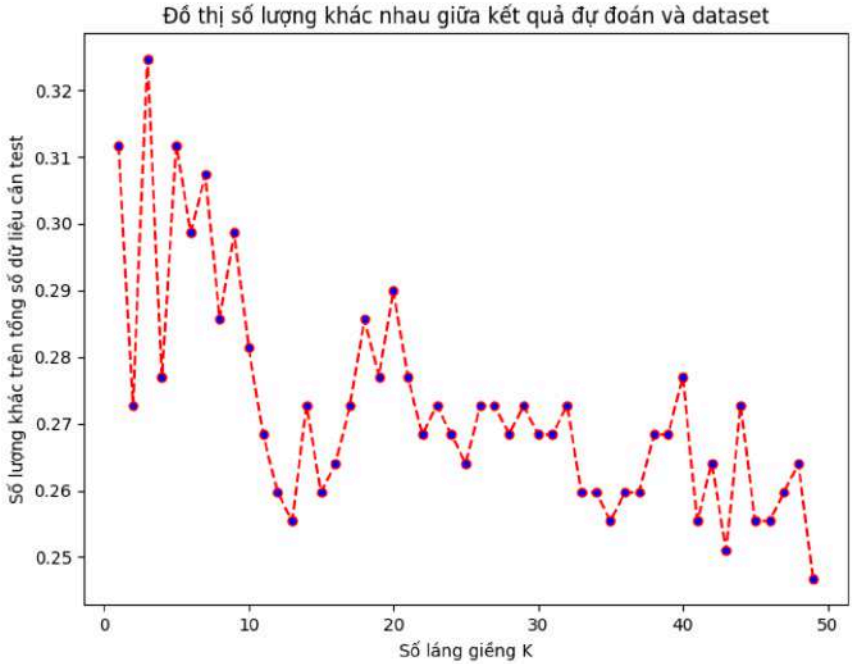
- Mô tả và chuẩn bị dữ liệu

```
import pandas as pd
import numpy as np
from sklearn import model_selection,metrics,neighbors
import matplotlib.pyplot as plt
data = pd.read_csv( '../dataset/diabetes.csv' )
N = len( data.columns ) - 1
X = data.iloc[:,:-1].values
y = data.iloc[:,N].values
X_train, X_test, y_train, y_test =
model_selection.train_test_split( X,y,test_size=0.3,random_state=42 )
```

```
plt.figure( figsize=(8,6) )
plt.title( "Đồ thị số lượng khác nhau giữa kết quả dự đoán và dataset" )
plt.plot( range(1,NUM_K),error,color='red',linestyle='dashed',marker='o',markerfacecolor='blue',
markersize=5 )
plt.xlabel( "Số láng giềng K" )
plt.ylabel( "Số lượng khác trên tổng số dữ liệu cần test" )
plt.show()
```

- Thử với nhiều *K* khác nhau để biểu diễn thành đồ thị, qua đó biết được với *K* bằng mấy thì có tỷ lệ số trường hợp dự đoán sai

```
NUM_K, error = 50, []
for i in range(1,NUM_K):
    model = neighbors.KNeighborsClassifier(
    n_neighbors=i )
    model.fit( X_train,y_train )
    y_pred = model.predict( X_test )
    error.append( np.mean(y_pred != y_test) )
```



File: KNN-sklearn.py

KNN dùng TensorFlow ver 2

- Lấy lại dữ liệu đã có trong ví dụ trước về bệnh tiểu đường.
  - Mô tả thêm thư viện TensorFlow

```
import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn import model_selection, metrics
```

  - Load dữ liệu từ tập tin sau đó dành ra 30% làm dữ liệu kiểm tra, và dữ liệu này cố định trong các lần thực thi khác nhau của chương trình

```
data = pd.read_csv( '../dataset/diabetes.csv' )
N = len( data.columns ) - 1
X = data.iloc[:,:-1].values
y = data.iloc[:,N].values
X_train,X_test,Y_train,Y_test =
model_selection.train_test_split( X,y,test_size=0.3
3,random_state=42 )
```

- Ở đây ta định nghĩa hàm để tìm khoảng cách nhỏ nhất từ các phần tử trong tập kiểm tra đến tập huấn luyện.

```
def mindist( x_test, x_train ):
    L = tf.sqrt( tf.reduce_sum(
      tf.square(tf.subtract(x_test,x_train)),1 ) )
    return tf.argmin( L,0 ).numpy()
```

- Gọi hàm mindist() ở trên để lần lượt tìm phần tử nào trong tập huấn luyện mà gần với từng phần tử của tập kiểm tra nhất, sau đó lấy nhãn của phần tử tương ứng cuả tập huấn luyện gán cho tập dự đoán này.
- Từ đó tính độ chính xác

```
y_pred = tf.Variable( [1]*len(X_test) )
for i in range( len(X_test) ):
    k = mindist( X_test[i,:],X_train )
    y_pred[i].assign( Y_train[k] )
print( "Accuracy: ",
metrics.accuracy_score(y_pred.numpy(),Y_test) )
```

- Kết quả

```
Accuracy:  0.6968503937007874
```

File: KNN-tensorflow.py

- **Tổng kết lại**: KNN được sử dụng cho việc dự báo cả phân loại và hồi quy.
- Tuy nhiên, nó được sử dụng rộng rãi cho phân loại, và thường được sử dụng trong các ứng dụng tìm kiếm.
- 1-NN để tạo một mô hình có các lớp dựa trên 1 điểm dữ liệu ở khoảng cách nhỏ nhất. Tương tự, 2-NN có nghĩa là chúng ta phải tạo một mô hình có các lớp dựa trên 2 điểm dữ liệu với khoảng cách nhỏ nhất.
- Các bước thuật toán bao gồm:
    - Tính khoảng cách giữa các điểm dữ liệu mới so với tất cả các dữ liệu huấn luyện
    - Chọn ra K mục có trong dữ liệu huấn luyện gần nhất với điểm dữ liệu mới.
    - Bình chọn lớp hay nhãn phổ biến nhất trong số các mục K , đó là lớp của điểm dữ liệu mới.

- Thuật toán KNN được chọn khi các số lượng các lớp dữ liệu tương ứng nhau về kích cỡ
- Khi một điểm dữ liệu phụ thuộc nhiều tham số thì KNN sẽ phải tính khoảng cách giữa các điểm rất chậm
- Chọn *K* bằng bao nhiêu cũng là vấn đề khó trên một dataset cụ thể

Thao tác thông dụng về TensorFlow version 2

- Khi dùng TensorFlow chúng ta phải khai báo trong vùng của nó.
  - Khai báo biến dùng tf.Variable()
  - Khai báo hằng, dùng tf.constant()
  - Truy cập một phần tử của tensor, ví dụ:
    - t4[1,1,2] có giá trị là 11.0
    - t3[0,3] có giá trị là 4

```
In [1]:
import tensorflow as tf

In [2]:
p0 = 57                                                          # python variable
p1 = ["Tran","Thuy","Anh","Quynh"]
p2 = [[1,2,3,4],[5,6,7,8]]
p3 = [[[0,1,2],[3,4,5]],[[6,7,8],[9,10,11]]]
t1 = tf.Variable( 61 )                                           # rank 0 tensor (scalar)
t2 = tf.Variable( ["Tran","Van","Lang"] )                        # rank 1 tensor (vector)
t3 = tf.Variable( [[1,2,3,4],[5,6,7,8]] )                        # rank 2 tensor (matrix)
t4 = tf.Variable( [[[0.,1.,2.], [3.,4.,5.]],[[6.,7.,8.],[9.,10.,11.]]] ) # rank 3 tensor

In [3]:
p0, p1, p2, p3, t1, t2, t3, t4

Out[3]: (57,
 ['Tran', 'Thuy', 'Anh', 'Quynh'],
 [[1, 2, 3, 4], [5, 6, 7, 8]],
 [[[0, 1, 2], [3, 4, 5]], [[6, 7, 8], [9, 10, 11]]],
 <tf.Variable 'Variable:0' shape=() dtype=int32, numpy=61>,
 <tf.Variable 'Variable:0' shape=(3,) dtype=string, numpy=array([b'Tran', b'Van', b'Lang'], dtype=object)>,
 <tf.Variable 'Variable:0' shape=(2, 4) dtype=int32, numpy=
 array([[1, 2, 3, 4],
        [5, 6, 7, 8]], dtype=int32)>,
 <tf.Variable 'Variable:0' shape=(2, 2, 3) dtype=float32, numpy=
 array([[[ 0.,  1.,  2.],
         [ 3.,  4.,  5.]],

        [[ 6.,  7.,  8.],
         [ 9., 10., 11.]]], dtype=float32)>)
```

- Chuyển một biến của TensorFlow trở thành một biến của Python dùng phương thức numpy() của đối tượng đó
- Ngược lại, dùng phương thức convert_to_tensor() của TensorFlow

```
In [4]:
1 p4 = t2.numpy()
2 t5 = tf.convert_to_tensor( p1 )

In [5]:
1 p4, t5

Out[5]: (array([b'Tran', b'Van', b'Lang'], dtype=object),
 <tf.Tensor: shape=(4,), dtype=string, numpy=array([b'Tran', b'Thuy', b'Anh', b'Quynh'], dtype=object)>)
```

- Chuyển vị ma trận, nhân ma trận

```
1 u = tf.constant([[1,2,3],[4,5,6]])
2 v = tf.constant([[1,0,0],[0,0,1]])
3 w = tf.matmul(u, tf.transpose(v))
```

```
1 w
```

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[1, 3],
       [4, 6]], dtype=int32)>
```

- Tính giá trị trung bình

```
1 t3 = tf.Variable( [[1,2,3,4],[5,6,7,8]] )
2 t3.numpy()
```

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]], dtype=int32)
```

```
1 tf.reduce_mean( t3 ).numpy()
```

```
4
```

```
1 tf.reduce_mean( t3,axis=0 ).numpy()
```

```
array([3, 4, 5, 6], dtype=int32)
```

```
1 tf.reduce_mean( tf.cast(t3,dtype=tf.float32),axis=1 ).numpy()
```

```
array([2.5, 6.5], dtype=float32)
```

Thuật toán K-means

- Trong thực tế thường hay có việc phân loại mang tính tương đối, chẳng hạn trong một tổ chức do con người điều hành và quản lý, thì có những nhóm người có cùng một nhận định về một sự kiện nào đó.
- Vấn đề đặt ra là cần phân loại để biết một người trong tổ chức này cơ bản là thuộc về nhóm nào.
- Cũng như vậy, có bao nhiêu người trong cùng một loại bệnh trong một cộng đồng dân cư
- Vấn đề phức tạp ở đây là không biết trước các đặc trưng của nhóm để phân lớp như trong thuật toán KNN.
- K-means hay K-means clustering, là thuật toán ***gom cụm K nhóm theo trung bình***.
- Thuật toán giúp cho chúng ta giải quyết vấn đề này. Đây chính là thuật toán thuộc loại học không giám sát

Minh hoạ

- Phân cụm K-means là thuật toán đơn giản trong unsupervised learning.
- Trong K-means clustering, nhãn của từng điểm dữ liệu (Data point) không biết trước.
- Vấn đề là làm thể nào để phân dữ liệu thành các nhóm/cụm (cluster) sao cho dữ liệu trong cùng một cụm có những tính chất giống nhau.

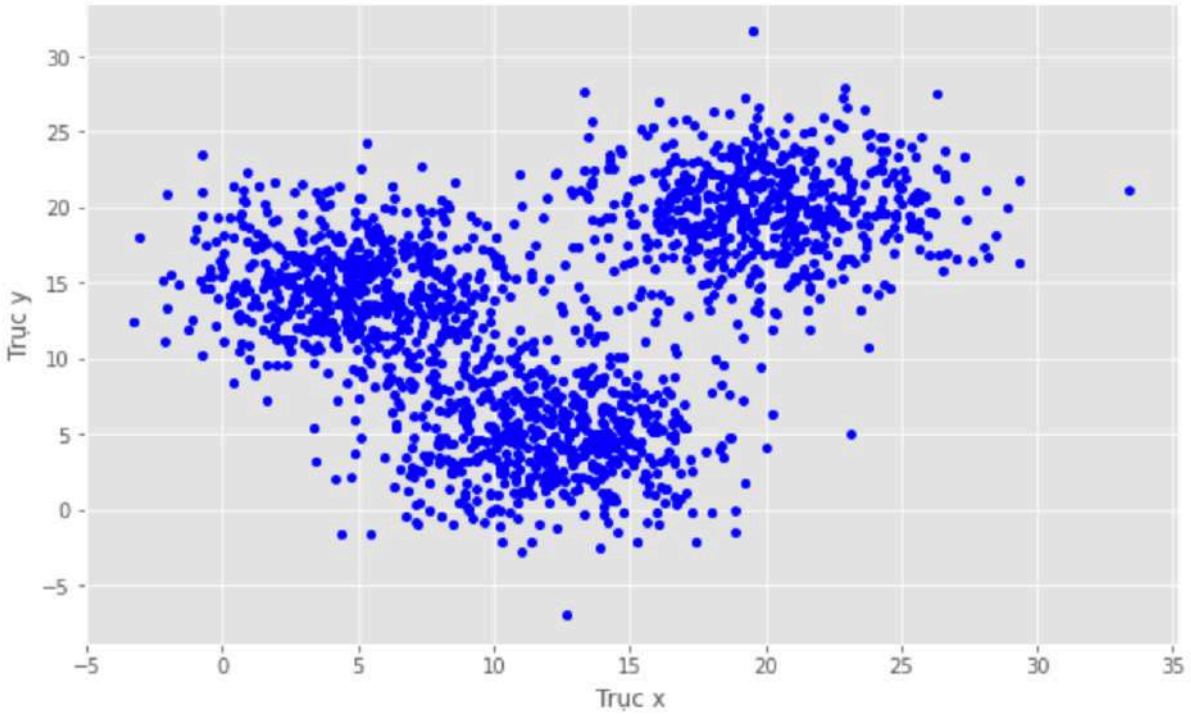
- Trong KNN, sau khi phân lớp xong thì mỗi điểm dữ liệu chỉ thuộc một lớp hay nhãn duy nhất.
- Trong khi đó với K-means, mỗi phần tử có thể thuộc nhiều nhóm hay cụm.
- Nhóm/cụm ở đây là tập hợp các điểm có các vector đặc trưng gần nhau.
- Việc đo khoảng cách giữa các vector thường được thực hiện dựa trên các chuẩn như đã trình bày, trong đó khoảng cách Euclide được sử dụng phổ biến nhất.

- Thuật toán gồm các bước dựa trên các điểm dữ liệu X– lưu ý là không có nhãn và số nhóm K.
  - Chọn K điểm bất kỳ làm điểm trung tâm của các nhóm (Center) gọi là trọng tâm (Centroid) hay còn gọi là điểm đại diện (Representative Point)
  - Tính khoảng cách từ các điểm dữ liệu đến K điểm Centroid
  - Phân bổ các điểm dữ liệu này vào các nhóm có khoảng cách đến Centroid của nhóm này gần nhất
  - Tính lại Centroid của K nhóm bằng cách lấy trung bình cộng của các điểm đã được gán vào nhóm.

Ứng dụng

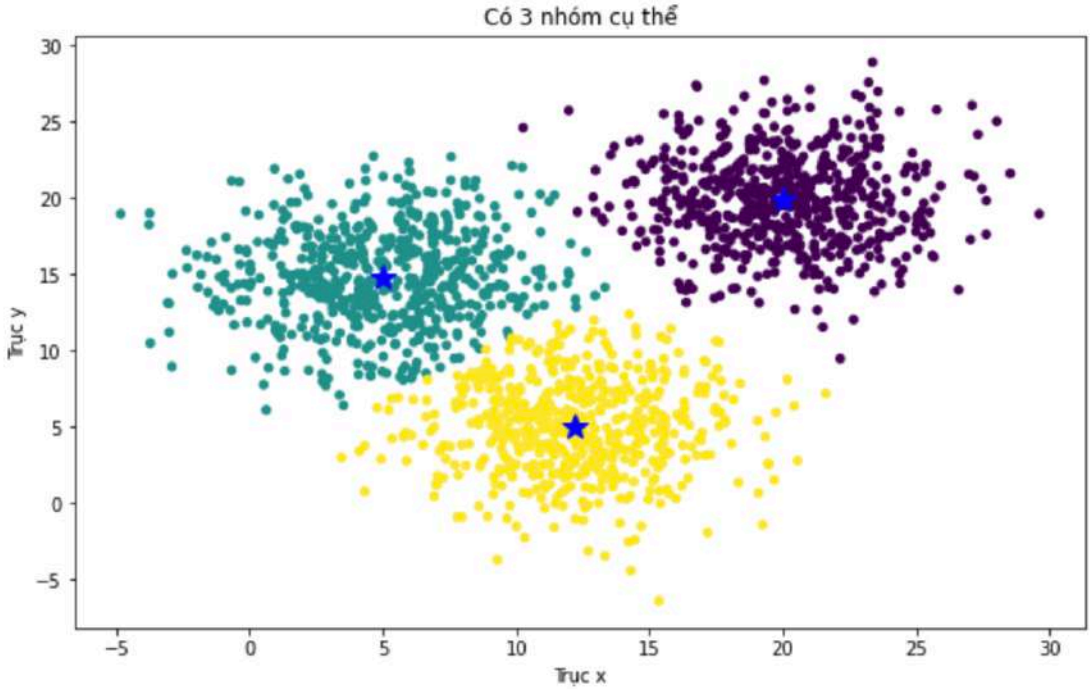
```
1 plt.figure( figsize=(10,6) )
2 plt.xlabel('Trục x')
3 plt.ylabel('Trục y')
4 plt.plot( X[:,0],X[:,1],'bo',markersize=4 )
5 plt.plot()
6 plt.show()
```

- Để dễ hình dung ta giả sử một cá thể (một người dân, một người làm việc, ...) trong quần thể (một cộng đồng dân cư, một tổ chức, ...) được số hoá bằng một toạ độ (x_i, y_i)
- Chẳng hạn ta có 600 điểm dữ liệu được phân bổ như hình



- Dữ liệu này được tạo ngẫu nhiên bằng phân phối Gauss xung quanh 3 điểm chính (ta tạo ra để định hướng) với ma trận hiệp phương sai chỉ định

```
centroids = [[5,15],[12,5],[20,20]]
DATASIZE  = 600
K = 3
def myrand(centroid):
    return np.random.multivariate_normal( mean=centroid,cov=[[10,0],[0,10]],size=DATASIZE )
X0 = myrand( centroids[0] )
X1 = myrand( centroids[1] )
X2 = myrand( centroids[2] )
X  = np.concatenate((X0, X1, X2), axis = 0)
```



- Để huấn luyện bằng K-means

```
model = cluster.KMeans( n_clusters=3 )
model.fit(X)
y_pred = model.predict(X)
```

- Sau khi có kết quả dự báo, ta thử coi lại các điểm trung tâm

```
print( "Các điểm trọng tâm là\n", model.cluster_centers_ )
print( "Kết quả phân nhóm\n", y_pred )
for i in range(len(y_pred)):
    print( y_pred[i], end=' ' )
```

- Coi qua hình minh hoạ

Dùng TensorFlow

- Lấy dữ liệu như trong ví dụ dùng phương thức của sklearn, ở đây cũng giống như khi dùng TensorFlow cho KNN, ta khai báo thêm các biến Tensor như sau:

```
X = tf.Variable( X_train )
```

- Lấy K thành phần trong tập huấn luyện X để làm K trọng tâm

```
cents = tf.Variable( X[0:K]
)
```

- Xây dựng hàm huấn luyện

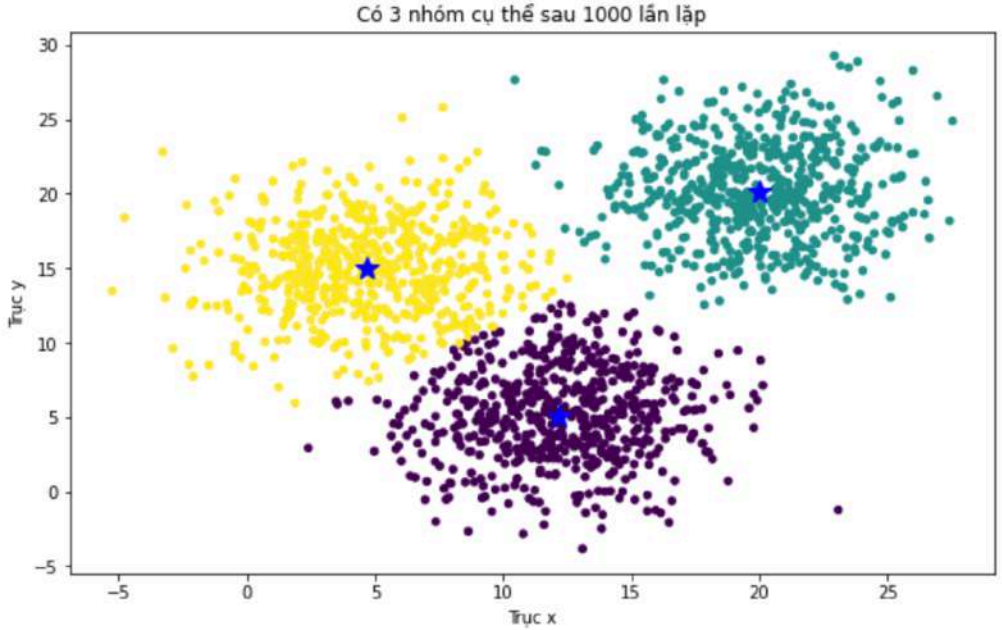
```
def pred(tt):
    cents_expanded = tf.expand_dims( tt,1 )
    L = tf.sqrt(tf.reduce_sum(tf.square(tf.subtract(
     X,cents_expanded)),axis=2))
    y = tf.argmin( L,0 )
    means = []
    for c in range(K):
     means.append( tf.reduce_mean(tf.gather(X,tf.reshape(tf.where(tf.
     equal(y,c)),[1,-1])),[1]) )
     new_cents = tf.concat( means,0 )
     return new_cents, y
```

- Lặp lại 1000 lần

```
for _ in range(1000):
    cents, y_pred = pred( cents )
```

- Vẽ lại hình để trực quan hoá

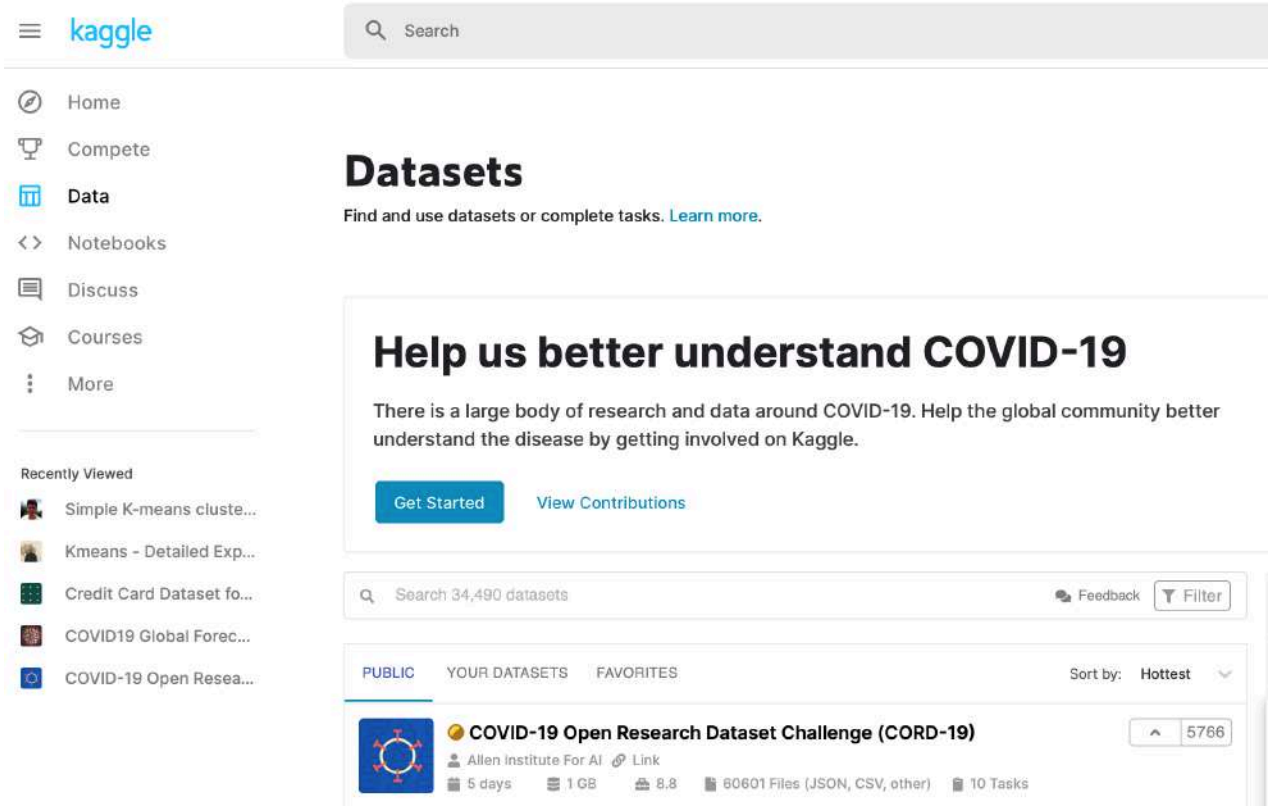
```
plt.figure( figsize=(10,6) )
plt.title( "Có 3 nhóm cụ thể sau 1000 lần lặp"
plt.xlabel('Trục x')
plt.ylabel('Trục y')
plt.scatter( X_train[:,0],X_train[:,1],c=y_pred,s=20 )
plt.plot( cents[:,0],cents[:,1],'k*',color='blue',markersize=15)
plt.show()
```



File: Kmeans-tensorflow.py

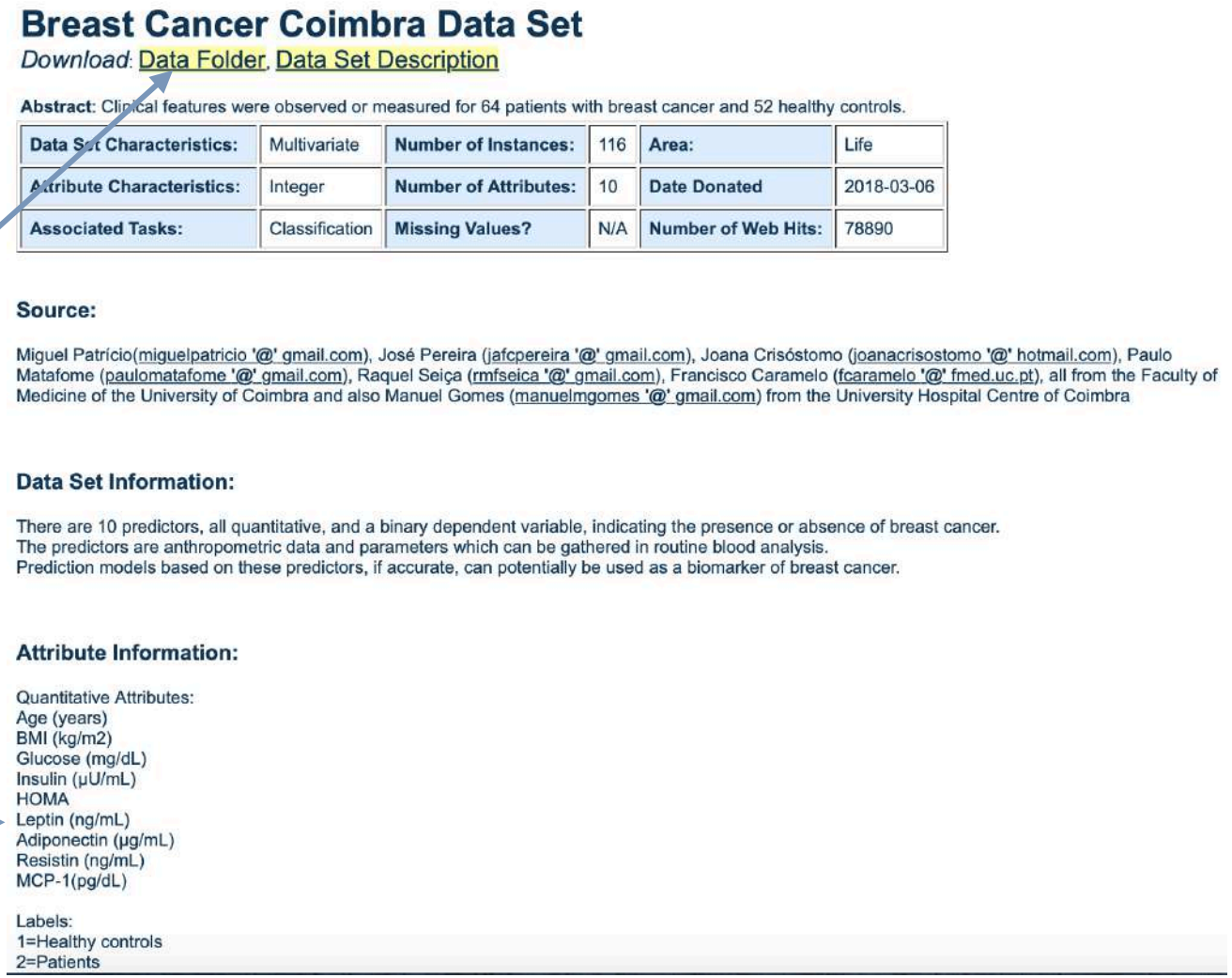
Ví dụ thực tế về K-means

- Trên internet có rất nhiều dataset để cho phép chúng ta thực nghiệm các phương pháp, cũng như để dần quen với việc giải quyết những vấn đề thực tế bằng các tính toán thông minh.
- Chẳng hạn, https://www.kaggle.com/datasets



- Hay tại kho về Machine Learning của Center for Machine Learning and Intelligent Systems (http://archive.ics.uci.edu/ml/datasets.php) thuộc University of California-Irvine.
- Tại đây lưu trữ dataset của rất nhiều lĩnh vực từ năm 1987 đến nay. Nhằm để phân tích thực nghiệm các thuật toán học máy.

- Giả sử ta dùng dữ liệu về bệnh ung thư vú (http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Coimbra)
- Dataset tại http://archive.ics.uci.edu/ml/machine-learning-databases/00451/dataR2.csv
- Có 9 thuộc tính và được gán nhãn là 1 khoẻ mạnh (healthy controls) và 2 là bị bệnh (patients)



Breast Cancer Coimbra Data Set

Download: Data Folder, Data Set Description

Abstract: Clinical features were observed or measured for 64 patients with breast cancer and 52 healthy controls.

Data Set Characteristics:	Multivariate	**Number of Instances:**	116	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	10	**Date Donated**	2018-03-06
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	78890

Source:

Miguel Patrício(miguelpatricio '@' gmail.com), José Pereira (jafcpereira '@' gmail.com), Joana Crisóstomo (joanacrisostomo '@' hotmail.com), Paulo Matafome (paulomatafome '@' gmail.com), Raquel Seiça (rmfseica '@' gmail.com), Francisco Caramelo (fcaramelo '@' fmed.uc.pt), all from the Faculty of Medicine of the University of Coimbra and also Manuel Gomes (manuelmgomes '@' gmail.com) from the University Hospital Centre of Coimbra

Data Set Information:

There are 10 predictors, all quantitative, and a binary dependent variable, indicating the presence or absence of breast cancer.
The predictors are anthropometric data and parameters which can be gathered in routine blood analysis.
Prediction models based on these predictors, if accurate, can potentially be used as a biomarker of breast cancer.

Attribute Information:

Quantitative Attributes:
Age (years)
BMI (kg/m2)
Glucose (mg/dL)
Insulin (µU/mL)
HOMA
Leptin (ng/mL)
Adiponectin (µg/mL)
Resistin (ng/mL)
MCP-1(pg/dL)

Labels:
1=Healthy controls
2=Patients

- Chúng ta có thể đọc trực tiếp file này từ internet cũng với những import như ví dụ trước

```
dataset = pd.read_csv( "http://archive.ics.uci.edu/ml/
machine-learning-databases/00451/dataR2.csv" )
```

- Có thể dữ liệu không đầy đủ, bằng cách kiểm tra, rồi sau đó có thể xử lý bằng cách thay giá trị trung bình tương ứng,

```
dataset.isnull().any()
```

```
data = dataset.fillna( dataset.mean() )
```

```
dataset.isnull().any()
```

```
Age             False
BMI             False
Glucose         False
Insulin         False
HOMA            False
Leptin          False
Adiponectin     False
Resistin        False
MCP.1           False
Classification  False
dtype: bool
```

- Do cột cuối cùng là nhãn được gán (đã phân lớp, nên ta dùng cột này như là dữ liệu để so sánh kết quả), từ đây tạo ra X_train và y_train như cách làm trong các vị dụ trước

```
N = len(data.columns)-1
X_train = data.iloc[:,0:-1].values
y_train = data.iloc[:,N].values
```

- Chọn số cụm là 2, và lặp lại 100 lần tối đa

```
model = cluster.KMeans( n_clusters=2, max_iter=100 ).fit(X_train)
y_pred = model.predict( X_train )
print( "Number of iterations run: ", model.n_iter_ )
print( "Sum of squared distances of samples to their closest cluster center: ", model.inertia_ )
print( "Coordinates of",NUM_CLUSTERS,"cluster centers\n", model.cluster_centers_ )
y_pred += 1  # Do nhãn lưu giá trị là 1 và 2
print( "Accuracy: ", metrics.accuracy_score(y_train,y_pred) )
```

```
Number of iterations run:  13
Sum of squared distances of samples to their closest cluster center:  5522774.8408874925
Coordinates of 3 cluster centers
 [[ 59.32258065  29.11360799 105.61290323  12.47903226   3.85361469
   29.10946774   8.63150145  21.74355613 980.28445161]
 [ 56.56470588  27.0235648   94.94117647   9.11237647   2.27242944
   25.70536235  10.74593924  12.16660941 372.1204    ]]
Accuracy:  0.49137931034482757
```

File: Kmeans-sklearn2.py

Tóm lại

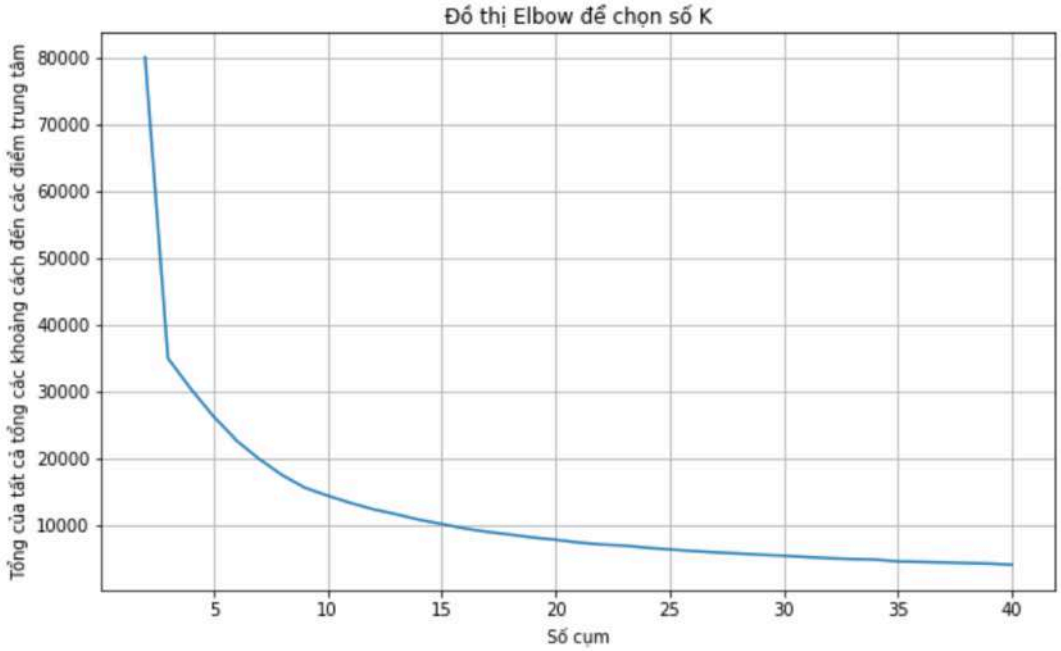
- Với ý tưởng của thuật toán là bộ dữ liệu khảo sát (mà ta hay gọi là dataset) được phân thành K nhóm.
- Mỗi nhóm tìm centroid (trọng tâm) tương ứng
- Sau đó tính tổng khoảng cách (sum of squared distances) của tất cả các dữ liệu thuộc nhóm này đến centroid; sau đó cộng dồn để được tổng của tất cả (Total sum of squared distances - TSS).
- Mục tiêu phân bộ dữ liệu ban đầu thành K cụm khác nhau sao cho các dữ liệu thuộc cùng một cụm là tương đồng nhất; điều đó có nghĩa TSS là nhỏ nhất.
- Khó khác của sử dụng K-means là lựa chọn K sao cho tối ưu.
- Một hạn chế của K-means đó là việc gom cụm mang tính tối ưu cục bộ vì lời giải tìm được căn cứ vào điểm trọng tâm được "định hướng" ban đầu

  - Do TSS giảm khi K tăng, và giảm cho đến không nếu K bằng số dữ liệu.
  - Mặt khác khi tăng K (có nghĩa là số cụm tăng), nên thuật toán trở nên không có ý nghĩa khi số cụm là quá nhiều.
- Chọn K bằng đồ thị Elbow: khi đồ thị có chỗ gấp này rõ nét nhất, đó là số K tối ưu.
- Chẳng hạn với ví dụ trước, cho K được thử từ 2 cho đến 40

```
tss = []
K = range(2,41)
for k in K:
    model = cluster.KMeans( n_clusters=k, max_iter=100 ).fit(X)
    tss.append( model.inertia_ )
```

- Nhìn vào đồ thị Elbow, ta thấy $K = 3$ là chỗ bị gấp khúc nhất

```
plt.figure( figsize=(10,6) )
plt.plot( K,tss )
plt.title( "Đồ thị Elbow để chọn số K" )
plt.xlabel( "Số cụm" )
plt.ylabel( "Tổng của tất cả tổng các
khoảng cách đến các điểm trung tâm" )
plt.grid( True )
plt.show())
```



- Nhưng do ta không thể chỉ cho máy "nhìn vào đồ thị" được như người, mà phải dùng tiếng nào đó để chỉ, ở đây ta lại dùng "tiếng Python".
- Nhưng trước hết phải dùng Toán học để nhận biết đâu là Elbow Point.
  - Điểm Elbow là điểm mà khoảng cách từ nó đến đường thẳng đi qua 2 điểm đầu và cuối của đường cong là lớn nhất.
  - Trước tiên: đây là phương trình đường thẳng qua 2 điểm A, B (là mảng các giá trị tung độ)

```
def fx(X,A,B):
    y = []
    for i in range(len(X)):
        x = X[i]
        temp = A[1] + (X[i]-A[0])*(B[1]-A[1])/(B[0]-A[0])
        y.append( temp )
    return y
```

- Tính khoảng cách giữa những điểm trên đường thẳng này và những điểm ở trên đường cong mà có cùng hoành độ, sau đó tìm điểm có khoảng cách lớn nhất để trả về.

```
import numpy as np
from scipy.spatial import distance
def ElbowPoint( x_axis, curve ):
    N = len(curve)
    S = [x_axis[0],curve[0]]
    E = [x_axis[N-1],curve[N-1]]
    y = fx( K,S,E )
    dist = []
    for i in range(N):
        x = x_axis[i]
        dist.append( distance.cdist([[x,curve[i]]],[[x,y[i]]]) )
    return np.argmax( dist )
```

- Số lượng cụm cần phân nhóm đó là vị trí của khoảng cách lớn nhất này

```
NUM_CLUSTERS = ElbowPoint( K, tss ) + 1
```

- Do chỉ số màng bắt đầu từ 0 nên phải cộng thêm 1 là vậy đó !

File: Kmeans-sklearn.py