



รายงาน

Fruit Detection

จัดทำโดย

ณัฏฐร เชิงเขาว์ 6209610416

ณัฐนนท์ ราชภูริรักษ์ 6209680799

สิริวิษณุ คล้ายรัศมี 6209680831

เสนอ

ผศ.ดร. วิรัตน์ จาริวงศ์ไพบูลย์

รายงานนี้เป็นส่วนหนึ่งของวิชา คพ.345 การเรียนรู้ของเครื่องจักรและการทำเหมืองข้อมูลเชิง

ประยุกต์

สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี

มหาวิทยาลัยธรรมศาสตร์ ปีการศึกษา 2564

บทนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา CS345 การเรียนรู้ของเครื่องจักรและการทำเหมืองข้อมูลเชิง
ประยุกต์ โดยคณะผู้จัดทำมีจุดประสงค์เพื่อนำความรู้ที่ได้จากการสืบค้นเทคโนโลยี ที่มีการนำ Machine
Learning มาประยุกต์ใช้ ในเรื่องของการศึกษา

คณะผู้จัดทำ

สารบัญ

เรื่อง	หน้า
ที่มาและความสำคัญ, วัตถุประสงค์และ Features	3
Source Code	4
1. การจัดเตรียมข้อมูล	5
2. การเขียนโค้ด	9
2.1 Supervised Learning	10
2.2 Unsupervised Learning	15
2.3 Cross-Validation	23
3. การประเมินผลโมเดล	24
4. การนำไปใช้จริง	32
สรุป	35
ความเกี่ยวข้องกับปัญญาประดิษฐ์	36
บรรณานุกรม	37

ที่มาและความสำคัญ

การทำ Object Detection ส่วนใหญ่มักจะใช้ Deep Learning ในการทำนายรูปภาพที่เป็นภาพสี ที่มีความแม่นยำที่สูง ทางผู้จัดทำจึงมีแนวคิดที่ว่าถ้าหากไม่ใช้ Deep Learning ในการทำนายรูปภาพ ทางผู้จัดทำจึงมีความสนใจในการใช้ Machine Learning ในการทำนายรูปภาพแทน

คำถาม/ปัญหาที่ต้องการค้นหาคำตอบ

1. จะสามารถสร้างโมเดลที่มีความแม่นยำพอ ๆ กับ Deep Learning โดยใช้ Machine Learning ได้หรือไม่
2. เราจะใช้เกณฑ์อะไรบ้างในการจำแนกข้อมูลที่เป็นรูปภาพ หากใช้ Machine Learning ในการประมวลผล

วัตถุประสงค์

1. เพื่อศึกษาการทำงานของ Machine Learning
2. เพื่อเพิ่มความรวดเร็วในการคัดแยกผลไม้

Feature

ทำนายชนิดของผลไม้โดยการนำภาพถ่ายผลไม้มาใช้กับโปรแกรม

ข้อจำกัด

ใช้ได้แค่กับรูป กล้วย, ส้ม, แอปเปิล

Source Code

เครื่องมือที่จำเป็นต่อการรันโค้ด

1. Google colab

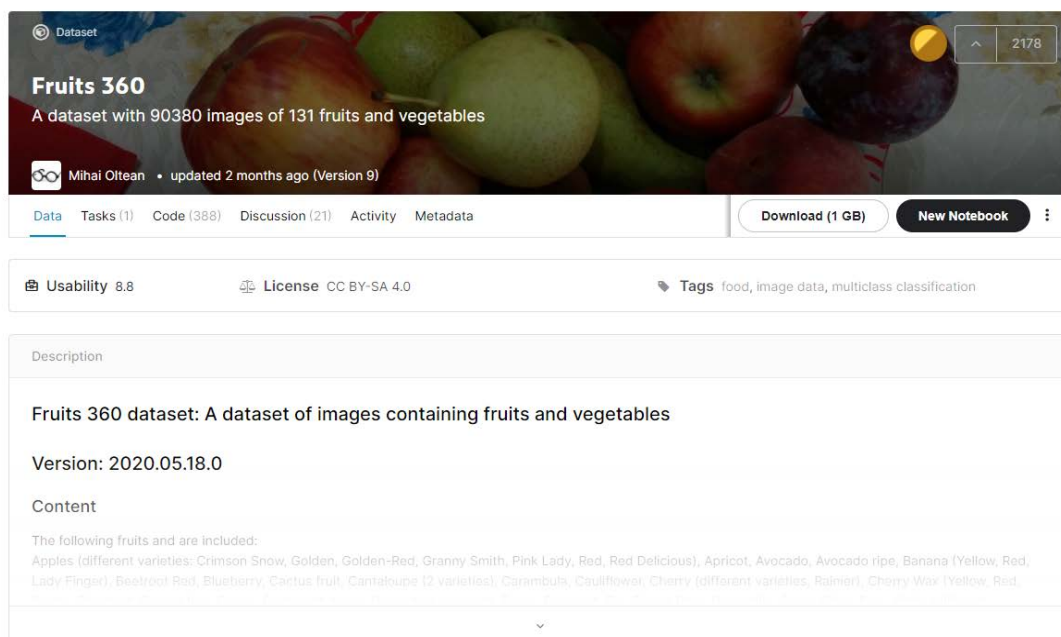
เฟรมเวิร์คที่จำเป็นต่อการรันโค้ด

1. Numpy
2. Matplotlib
3. Scikitlearn
4. opencv
5. Pickle

ชุดข้อมูลที่จำเป็นต้องใช้

จำเป็นต้องโหลด dataset ที่เป็นไฟล์รูปภาพมาเพื่อทำการสร้างโมเดลไว้ทำนายผลไม้จากภาพ โดย ชุดข้อมูลดังกล่าวมีรูปตัวอักษรทั้งหมด 90483 รูป แต่เนื่องจากชุดข้อมูลมีจำนวนมาก จนทำให้การโหลดชุดข้อมูลต้องใช้เวลานาน จึงลดเหลือ 47400 รูป โดยสามารถดาวน์โหลดได้จาก

<https://www.kaggle.com/moltean/fruits>



Fruits 360
A dataset with 90380 images of 131 fruits and vegetables

Mihai Oltean • updated 2 months ago (Version 9)

Data Tasks (1) Code (388) Discussion (21) Activity Metadata

Download (1 GB) New Notebook

Usability 8.8 License CC BY-SA 4.0 Tags food, image data, multiclass classification

Description

Fruits 360 dataset: A dataset of images containing fruits and vegetables

Version: 2020.05.18.0

Content

The following fruits and are included:

Apples (different varieties: Crimson Snow, Golden, Golden-Red, Granny Smith, Pink Lady, Red, Red Delicious), Apricot, Avocado, Avocado ripe, Banana (Yellow, Red, Lady Finger), Beetroot Red, Blueberry, Cactus fruit, Cantaloupe (2 varieties), Carambola, Cauliflower, Cherry (different varieties, Rainier), Cherry Wax (Yellow, Red,

การจัดเตรียมข้อมูล (Data Preparing)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
import os
IMG_PATH_TRAIN = "/content/drive/MyDrive/fruits_dataset/Fruit_Training"
IMG_PATH_TEST = "/content/drive/MyDrive/fruits_dataset/Fruit_Test"
IMG_HEIGHT = 100
IMG_WIDTH = 100
```

```
import cv2
```

import library ที่จำเป็นในการทำ data preparing

```
def create_image_dataset(img_folder):
    img_data_array = []
    class_name = []

    for dir1 in sorted(os.listdir(img_folder)):
        for file in sorted(os.listdir(os.path.join(img_folder, dir1))):
            image_path = os.path.join(img_folder, dir1, file)
            image = cv2.imread(image_path, cv2.COLOR_BGR2RGB)
            image = cv2.resize(image, (IMG_HEIGHT, IMG_WIDTH), interpolation = cv2.INTER_AREA)
            image = np.array(image)
            img_data_array.append(image)
            class_name.append(dir1)

    return img_data_array, class_name
```

```
img_train, class_name_train = create_image_dataset(IMG_PATH_TRAIN)
img_test, class_name_test = create_image_dataset(IMG_PATH_TEST)
```

input ชุดข้อมูลรูปภาพผลไม้ทั้งหมดจากไฟล์ที่ดาวน์โหลดจาก <https://www.kaggle.com/moltean/fruits>
ชุดข้อมูลประกอบด้วย 4 ไฟล์ โดยมีการแบ่งเป็น train และ test data มาให้ จากนั้นสร้างฟังก์ชันในการ
แปลงข้อมูลรูปภาพให้เป็น numpy array

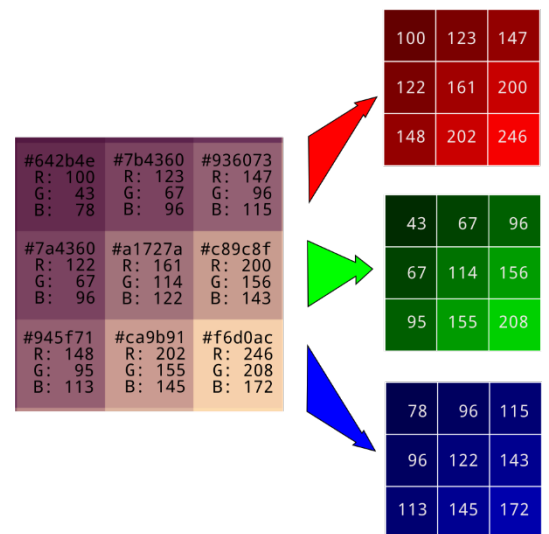
```
fruit_dict = {name: val for val, name in enumerate(np.unique(class_name_train))}
```

สร้าง dictionary ของชื่อผลไม้ในแต่ละชนิด

```
target_train = [fruit_dict[class_name_train[i]] for i in range(len(class_name_train))]  
target_test = [fruit_dict[class_name_test[i]] for i in range(len(class_name_test))]
```

สร้าง list เพื่อเก็บ target ของชื่อผลไม้

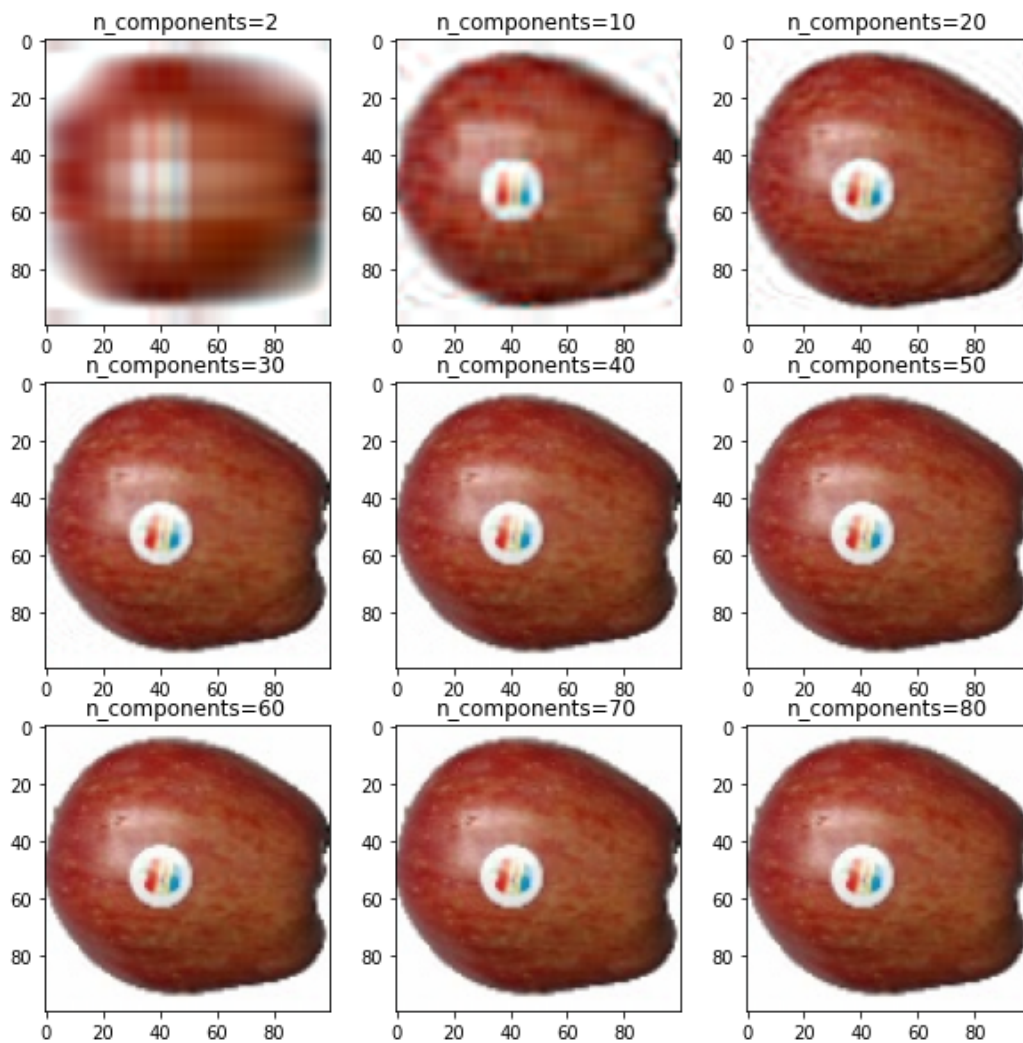
```
from sklearn.decomposition import PCA  
  
def make_pca(image,nc):  
    (R, G, B) = cv2.split(image)  
  
    rgb = [R/255, G/255, B/255]  
    arr = []  
    for i in range(3):  
        pca = PCA(n_components=nc, whiten=True)  
        trans_pca = pca.fit_transform(rgb[i])  
        arr.append(pca.inverse_transform(trans_pca))  
  
    return cv2.merge(tuple(arr)).astype("float")
```



ที่มาของรูปที่ 2: <https://www.kdnuggets.com/2020/01/convert-picture-numbers.html>

เพราะว่าในข้อมูลรูปภาพมีจำนวน feature จำนวนมาก จึงต้องทำการลดมิติข้อมูล โดยใช้ PCA เพื่อให้ลดความละเอียดของภาพ โดยใช้เทคนิคการแยกสีของรูปภาพออกมาเป็นสีแดง, สีเขียว, สีน้ำเงิน ตามลำดับเพื่อที่จะลดมิติของภาพสีทั้งสามสีได้ จากนั้นนำมาประกอบกัน

จากนั้นนำรูปภาพหลังจากทำ PCA มาแสดงผล



จากการลดมิติของข้อมูลพบว่า เมื่อเราลดจำนวน components ในรูปผลไม้น้อยเท่าไร รูปภาพจะมีความละเอียดลดลง

แต่เนื่องจากข้อมูลรูปภาพที่ได้มามีจำนวนมากๆ ทำให้ RAM ทำงานหนักมากขึ้น จึงทำให้ไม่สามารถลดมิติของรูปภาพได้ ดังนั้นสร้างชุดข้อมูลที่เก็บค่า RGB โดยทำการแยกสีของรูปภาพออกมาเป็นสีแดง, สีเขียว และ สีน้ำเงินตามลำดับ จากนั้นเก็บข้อมูลสีโดยใช้หลักสถิติต่างๆมาคำนวณ แล้วนำมา Scaling โดยใช้ Standard Scaler แทน


```

from scipy.stats import sem

def create_dataset(image):
    feature = []
    for img in image:
        (R, G, B) = cv2.split(img)
        feature.append([np.mean(R), np.mean(G), np.mean(B),
                        np.std(R), np.std(G), np.std(B),
                        np.median(R), np.median(G), np.median(B),
                        sem(R, axis=None, ddof=0), sem(G, axis=None, ddof=0), sem(B, axis=None, ddof=0),
                        np.percentile(R, 25), np.percentile(G, 25), np.percentile(B, 25),
                        np.percentile(R, 50), np.percentile(G, 50), np.percentile(B, 50),
                        np.percentile(R, 75), np.percentile(G, 75), np.percentile(B, 75)])

    return feature

```

สร้างชุดข้อมูลใหม่โดยใช้การแยกสี RGB ออกมาเป็นสีแดง, สีเขียว, สีน้ำเงิน จากนั้นนำมาสร้าง list เพื่อที่จะเก็บข้อมูลทางสถิติทั้ง 3 สี ได้แก่ mean, standard derivation, median, standard error mean และ percentile 25%, 50%, 75% ตามลำดับ

```

X_train = np.array(create_dataset(img_train))
X_test = np.array(create_dataset(img_test))

```

```

y_train = np.array(target_train)
y_test = np.array(target_test)

```

```

np.savetxt('X_train_image.txt', X_train)
np.savetxt('X_test_image.txt', X_test)

np.savetxt('y_train_image.txt', y_train)
np.savetxt('y_test_image.txt', y_test)

```

แปลงเป็น numpy array แล้ว save ชุดข้อมูลทั้งสี่ file ลงใน text file

การเขียนโค้ด (Implementation)

```
PATH = '/content/drive/MyDrive/fruits_dataset/rgb_dataset'

X_train = np.loadtxt(PATH + '/' + 'X_train_image.txt')
X_test = np.loadtxt(PATH + '/' + 'X_test_image.txt')

y_train = np.loadtxt(PATH + '/' + 'y_train_image.txt').astype('int')
y_test = np.loadtxt(PATH + '/' + 'y_test_image.txt').astype('int')
```

โหลดชุดข้อมูลทีหลังจากทำการ Data preparing แล้ว

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

เนื่องจากข้อมูลสีของรูปผลไม้ทั้งหมดเป็น continuous จึงจะต้อง scaling ข้อมูลสีเพื่อที่จะให้ข้อมูล อยู่จุดศูนย์กลาง และบาง algorithm จำเป็นต้อง scaling ข้อมูลก่อนเพราะว่ามัน sensitive มากกับข้อมูล

ขั้นตอนการเขียนโค้ดของ Supervised Learning

การสร้างโมเดลเพื่อที่จะทำนายรูปผลไม้ เราจะใช้ algorithm ในการทำโมเดลมีดังนี้ k-Nearest Neighbors, Naïve Bayes, SVC, Neural Network, Decision Tree และ Random Forest

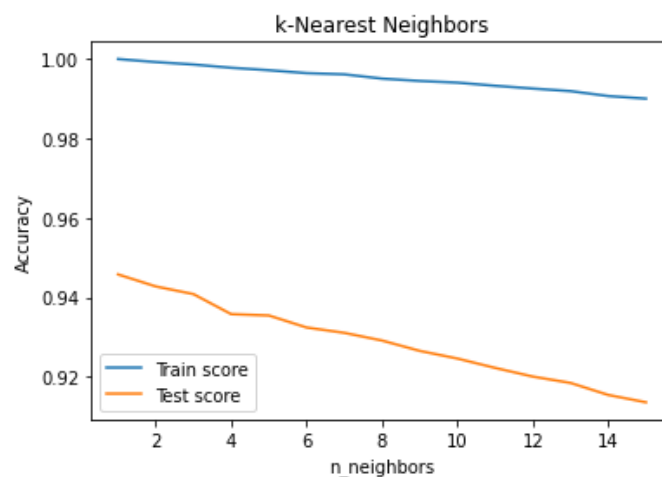
k-Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier

knn_train_score = []
knn_test_score = []
n_neighbors = np.arange(1,16)

for nb in n_neighbors:
    knn = KNeighborsClassifier(n_neighbors=nb).fit(X_train_scaled, y_train)
    knn_train_score.append(knn.score(X_train_scaled, y_train))
    knn_test_score.append(knn.score(X_test_scaled, y_test))

plt.title('k-Nearest Neighbors')
plt.plot(n_neighbors, knn_train_score, label='Train score')
plt.plot(n_neighbors, knn_test_score, label='Test score')
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



จากกราฟพบว่าค่า accuracy ทั้งสองค่าลดลงและกราฟห่างออกจากกันตามจำนวน neighbors แล้วพบว่ายิ่งค่า neighbors มากเท่าไร test score จะยิ่งน้อยและมีอาการ overfit มากขึ้น

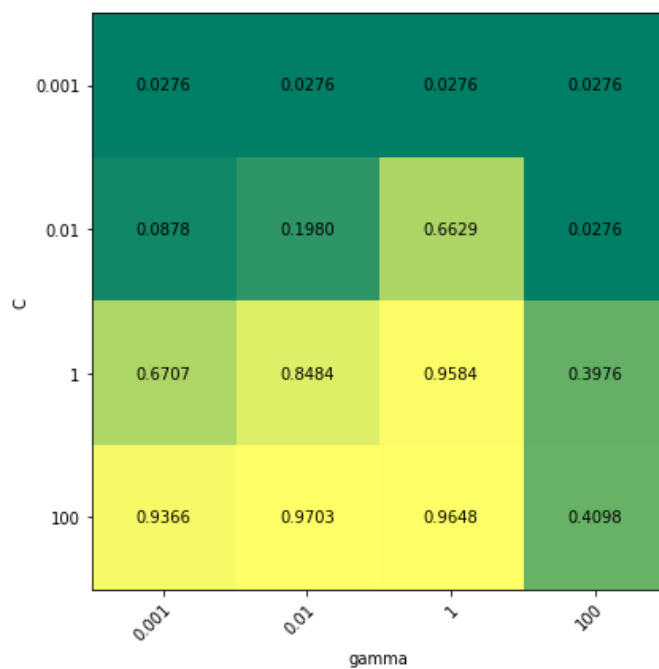
SVC

```
from sklearn.svm import SVC

C = [0.001, 0.01, 1, 100]
gam = [0.001, 0.01, 1, 100]

test_score = np.zeros((len(C),len(gam)))

for i in range(len(C)):
    for j in range(len(gam)):
        svm = SVC(kernel='rbf', C=C[i], gamma=gam[j]).fit(X_train_scaled, y_train)
        test_score[i][j] = svm.score(X_test_scaled, y_test)
```



จาก heatmap พบว่าเมื่อเราใช้ค่า C มาก และ gamma น้อย จะทำให้ค่า test score เพิ่มขึ้น
ในทางกลับกัน หากเราใช้ค่า C น้อย และ gamma มากเท่าไร ค่า test score จะยิ่งน้อยลง

Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB().fit(X_train_scaled, y_train)
print("Naive Bayes:")
print("Naive Bayes Train score: {:.4f}".format(nb.score(X_train_scaled, y_train)))
print("Naive Bayes Test score: {:.4f}".format(nb.score(X_test_scaled, y_test)))

Naive Bayes:
Naive Bayes Train score: 0.7639
Naive Bayes Test score: 0.6666
```

จากการทำโมเดลพบว่า Train score กับ Test score มีค่าน้อย เมื่อเทียบกับ algorithm อื่นๆ

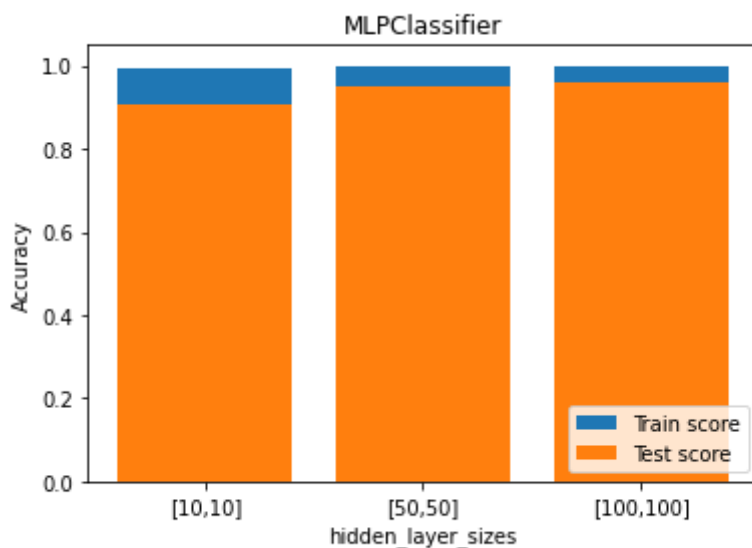
Neural Network

```
from sklearn.neural_network import MLPClassifier

hls = [[10,10],[50,50],[100,100]]
train_score_adam = []
test_score_adam = []

for h in hls:
    mlp_adam = MLPClassifier(solver='adam', max_iter=1000, random_state=0, hidden_layer_sizes=h).fit(X_train_scaled, y_train)
    train_score_adam.append(mlp_adam.score(X_train_scaled, y_train))
    test_score_adam.append(mlp_adam.score(X_test_scaled, y_test))

hs_name = ["[10,10]", "[50,50]", "[100,100]"]
plt.title('MLPClassifier')
plt.bar(hs_name, train_score_adam, label='Train score')
plt.bar(hs_name, test_score_adam, label='Test score')
plt.xlabel('hidden_layer_sizes')
plt.ylabel('Accuracy')
plt.legend(loc=4)
plt.show()
```



จากกราฟพบว่า จำนวน Hidden Layer มากเท่าไร โมเดลก็จะ train ได้ละเอียดยิ่งขึ้น โดยสรุปว่ายิ่งจำนวน Hidden Layer มากเท่า Test score จะยิ่งมากขึ้นและมีความ generalization แต่ค่า Train score ก็ยังคงที่ประมาณ 0.99

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

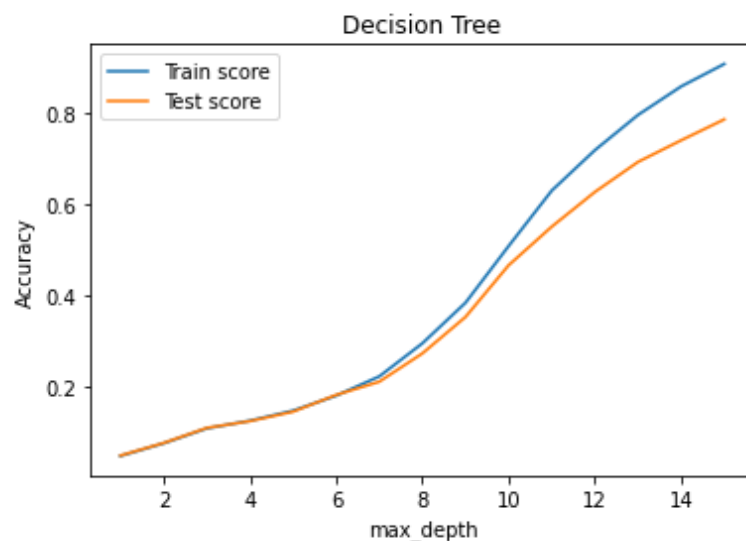
tree = DecisionTreeClassifier(random_state=0).fit(X_train_scaled, y_train)
print("Decision Tree before Pre-pruning:")
print("Train score: {:.4f}".format(tree.score(X_train_scaled, y_train)))
print("Test score: {:.4f}".format(tree.score(X_test_scaled, y_test)))
```

Decision Tree before Pre-pruning:
Train score: 1.0000
Test score: 0.8539

```
tree_train_score = []
tree_test_score = []
max_depth = np.arange(1,16)

for md in max_depth:
    tree = DecisionTreeClassifier(max_depth=md, random_state=0).fit(X_train_scaled, y_train)
    tree_train_score.append(tree.score(X_train_scaled, y_train))
    tree_test_score.append(tree.score(X_test_scaled, y_test))

plt.title('Decision Tree')
plt.plot(max_depth, tree_train_score, label='Train score')
plt.plot(max_depth, tree_test_score, label='Test score')
plt.xlabel('max_depth')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



เมื่อเรา train โมเดลก่อนที่จะทำการ Pre-pruning จะพบว่า train และ test score มีอาการ overfit และถ้าเราทำการ Pre-pruning จะพบว่าในกราฟนี้ จากค่า max_depth ตั้งแต่ 1 ถึง 6 มีอาการ underfit พอถึงค่าตั้งแต่ 6 ขึ้นไป จะพบว่าทั้งสองค่าเพิ่มขึ้นและมีความแตกต่างกัน กล่าวคือค่าทั้งสองค่ามีระยะห่างออกจากกันตาม max_depth

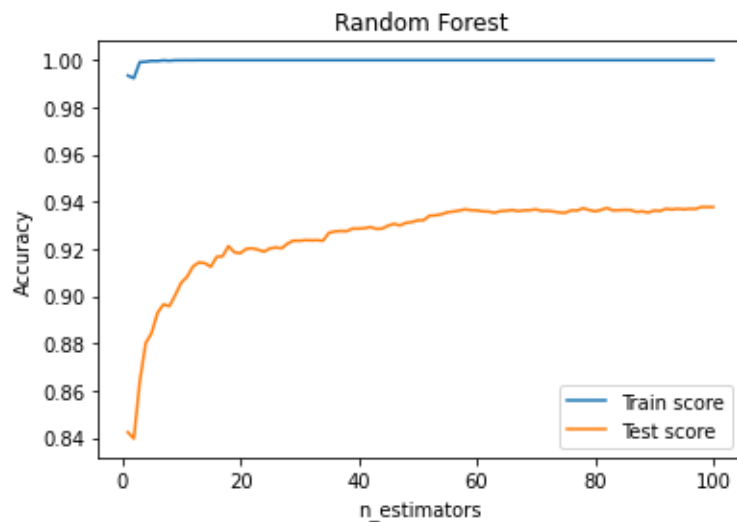
Random Forest

```
from sklearn.ensemble import RandomForestClassifier

forest_train_score = []
forest_test_score = []
n_estimators = np.arange(1,101)

for nes in n_estimators:
    forest = RandomForestClassifier(n_estimators=nes, n_jobs=-1, random_state=0).fit(X_train_scaled, y_train)
    forest_train_score.append(forest.score(X_train_scaled, y_train))
    forest_test_score.append(forest.score(X_test_scaled, y_test))

plt.title('Random Forest')
plt.plot(n_estimators, forest_train_score, label='Train score')
plt.plot(n_estimators, forest_test_score, label='Test score')
plt.xlabel('n_estimators')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



จากกราฟพบว่าในค่า test score ที่มีค่า estimators ตั้งแต่ค่า 1 ถึง 20 เป็นต้นไป มีอาการ overfit ส่วนค่า estimators ตั้งแต่ 21 ขึ้นไปค่า training และ test score จะค่อนข้างคงที่และเกิด generalization โดยสรุปว่าค่า estimators มากๆ กล่าวคือยิ่งจำนวนต้นไม้มากเท่าไร ก็ยิ่ง train ได้ดีขึ้น

ขั้นตอนการเขียนโค้ดของ Unsupervised Learning (Clustering)

การทำ Cluster ก่อนอื่น นำ X_{train} , X_{test} และ y_{train} , y_{test} มาต่อกันเป็น X และ y ตามลำดับ ส่วน X จะเอาแค่ค่า mean ของค่าสีแดง, สีเขียว และสีน้ำเงินตามลำดับ

```
X = np.concatenate((X_train, X_test))
X = X[:,0:3]

y = np.concatenate((y_train, y_test))
```

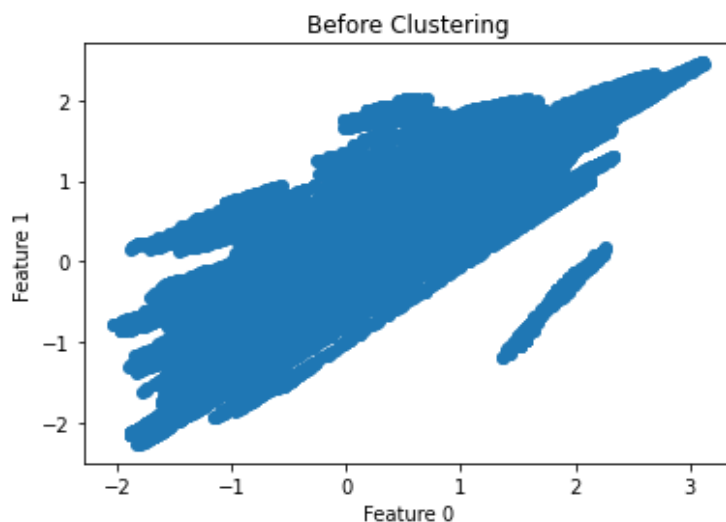
จากนั้น นำ X มาทำ preprocessing

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

กราฟก่อนทำ Clustering

```
plt.scatter(X_scaled[:, 0], X_scaled[:, 1])
plt.title('Before Clustering')
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')
plt.show()
```



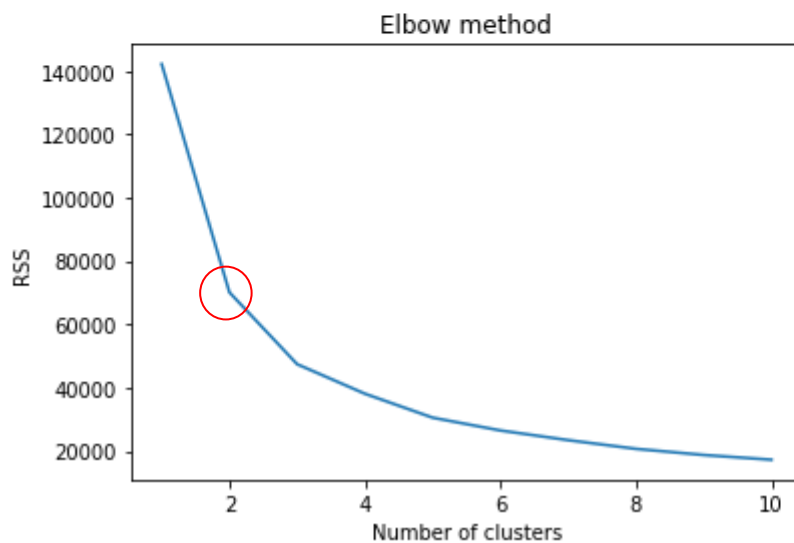
K-Means

สร้างกราฟ Elbow method เพื่อที่จะหาจุด Elbow point กล่าวคือจุดหักศอกที่มีมุมแหลมที่สุด

```
from sklearn.cluster import KMeans

rss = []
n_clusters = np.arange(1, 11)
for nc in n_clusters:
    kmeans = KMeans(n_clusters = nc, n_init = 100, random_state = 17)
    kmeans.fit(X_scaled)
    rss.append(kmeans.inertia_)

plt.plot(n_clusters, rss)
plt.title("Elbow method")
plt.xlabel("Number of clusters")
plt.ylabel("RSS")
plt.show()
```



จากกราฟพบว่าจุดที่มีการหักศอกมากที่สุดนั่นก็คือ $k = 2$

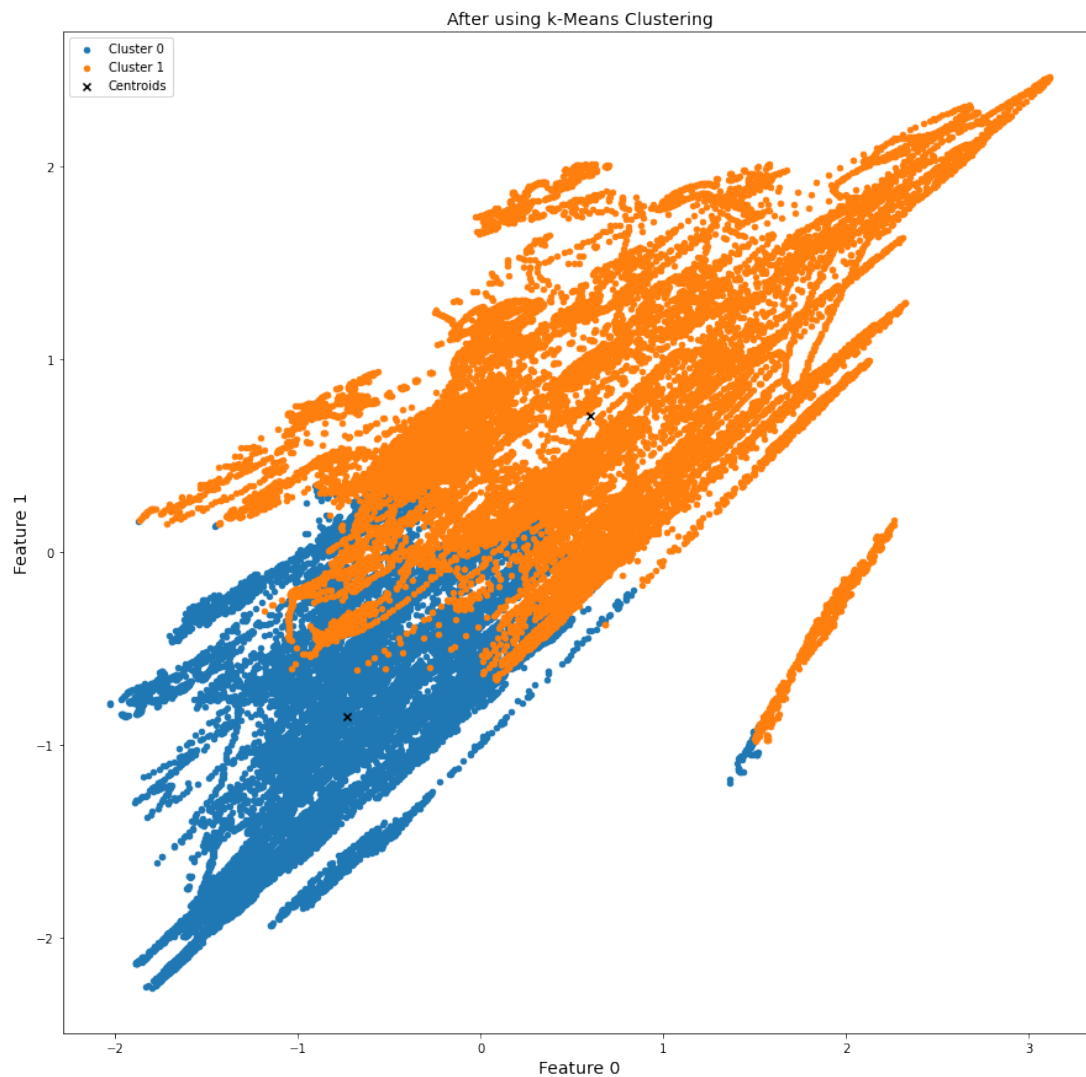
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2, n_init = 100, random_state = 17)
y_kmeans = kmeans.fit_predict(X_scaled)
```

จากนั้นนำจุด Elbow point มาใส่ในพารามิเตอร์ของ k-Means เพื่อที่จะ train โมเดล แล้วเก็บค่า label หลังจาก train ข้อมูลกับ K-Means แล้ว

สร้างกราฟ Feature เป็น 2 มิติ หลังจากทำ k-Means Clustering

```
plt.figure(figsize=(15,15))
for i in range(3):
    plt.scatter(X_scaled[y_kmeans == i, 0], X_scaled[y_kmeans == i, 1], s=20, cmap='viridis', label='Cluster '+str(i))

plt.scatter(centroid[:, 0], centroid[:, 1], marker='x', c='black', label='Centroids')
plt.title('After using k-Means Clustering')
plt.xlabel('Feature 0')
plt.ylabel('Feature 1')
plt.legend()
plt.show()
```

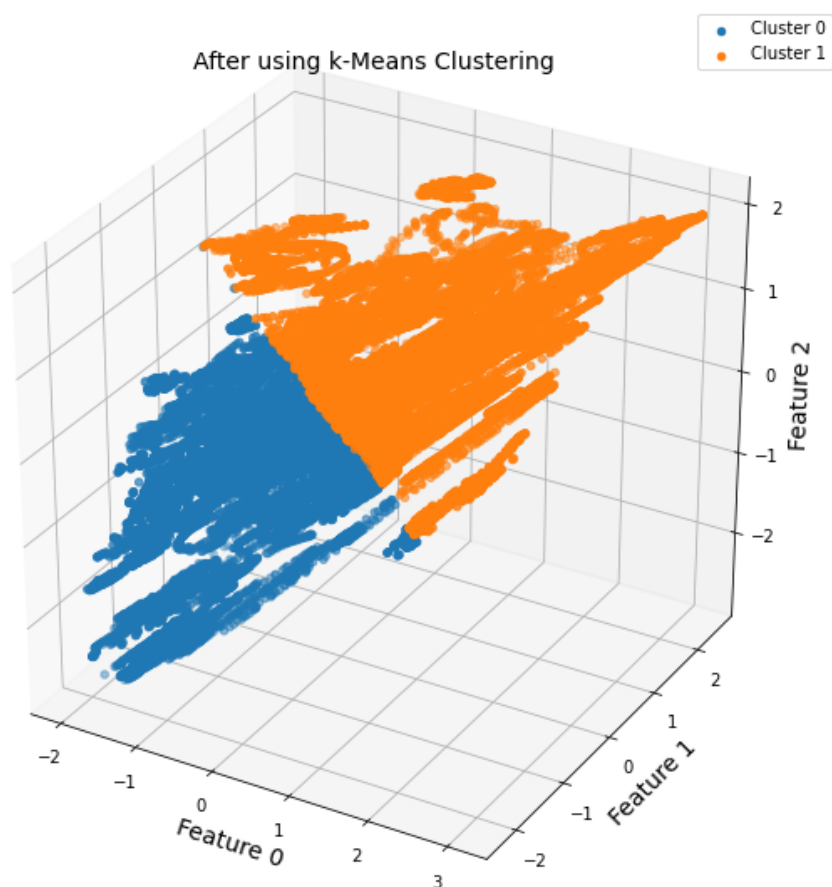


สร้างกราฟ Feature เป็น 3 มิติ หลังจากทำ k-Means Clustering

```
from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection='3d')
font_dict = {'family': 'DejaVu Sans', 'size': 14}

for i in range(3):
    ax.scatter(X_scaled[y_kmeans == i,0], X_scaled[y_kmeans == i,1], X_scaled[y_kmeans == i,2], label=f'Cluster {i}')

ax.set_title('After using k-Means Clustering', font_dict)
ax.set_xlabel('Feature 0', font_dict)
ax.set_ylabel('Feature 1', font_dict)
ax.set_zlabel('Feature 2', font_dict)
ax.legend()
plt.show()
```



จากกราฟพบว่า ในกราฟ 2 มิติ มีการจัดกลุ่มกันเป็นก้อนๆ และแบ่ง cluster ได้อย่างเห็นได้ชัด แต่บางข้อมูลมีการทับซ้อนกันกับ cluster อื่นๆ ทำให้กราฟดูซับซ้อน เมื่อเราสร้างกราฟ 3 มิติ จะพบว่ากราฟมีการจัดกลุ่มกันเป็นก้อนๆ แบ่ง cluster ได้อย่างเห็นได้ชัดที่สุด และเป็นระเบียบเรียบร้อย

สร้างตัวแปร centroid เพื่อเก็บค่าจุด Centroid (Cluster center)

```
centroid = kmeans.cluster_centers_
```

Cluster center:

```
[[-0.72716573 -0.85398774 -0.75741254]  
 [ 0.60383735  0.70915016  0.62895425]]
```

Cluster center after inverse transform:

```
[[102.66177553 115.08440791 143.41899681]  
 [144.80285425 173.86740701 194.83939186]]
```

Red : [102.66177553 144.80285425]

Green : [115.08440791 173.86740701]

Blue : [143.41899681 194.83939186]

เมื่อแสดงจุด centroid ทั้ง 3 จุด จะพบว่า ค่าเฉลี่ยทั้ง 3 สี จะจำแนกตามความเข้มของสี อย่างเช่น ค่าเฉลี่ยของสีแดงเป็น 102.66177553 คาดว่าจะเป็นจุดที่รูปผลไม้รูปนี้น่าจะใช้สีแดงค่อนข้างมาก

DBSCAN

ทำการ train ข้อมูลโดยใช้ DBSCAN

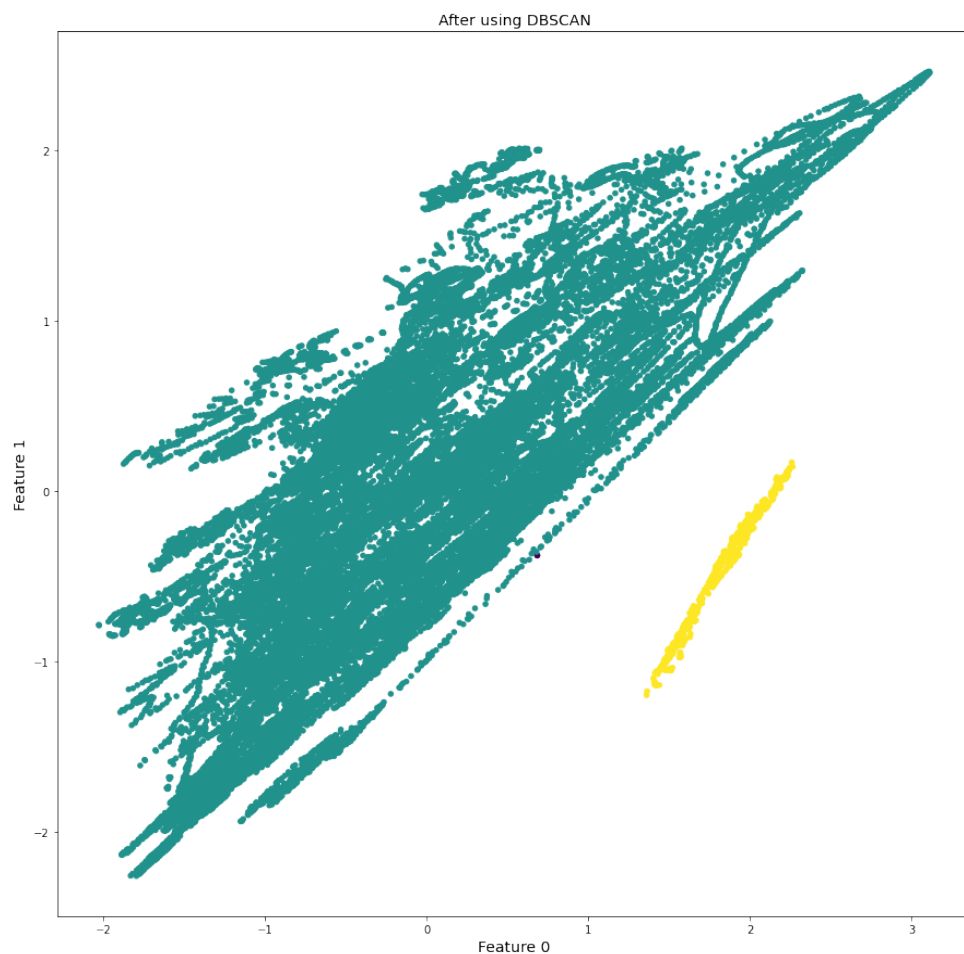
```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN()
y_dbscan = dbscan.fit_predict(X_scaled)
```

สร้างกราฟ Feature เป็น 2 มิติ หลังจากทำ DBSCAN

```
plt.figure(figsize=(15,15))
font_dict = {'family': 'DejaVu Sans', 'size': 14}

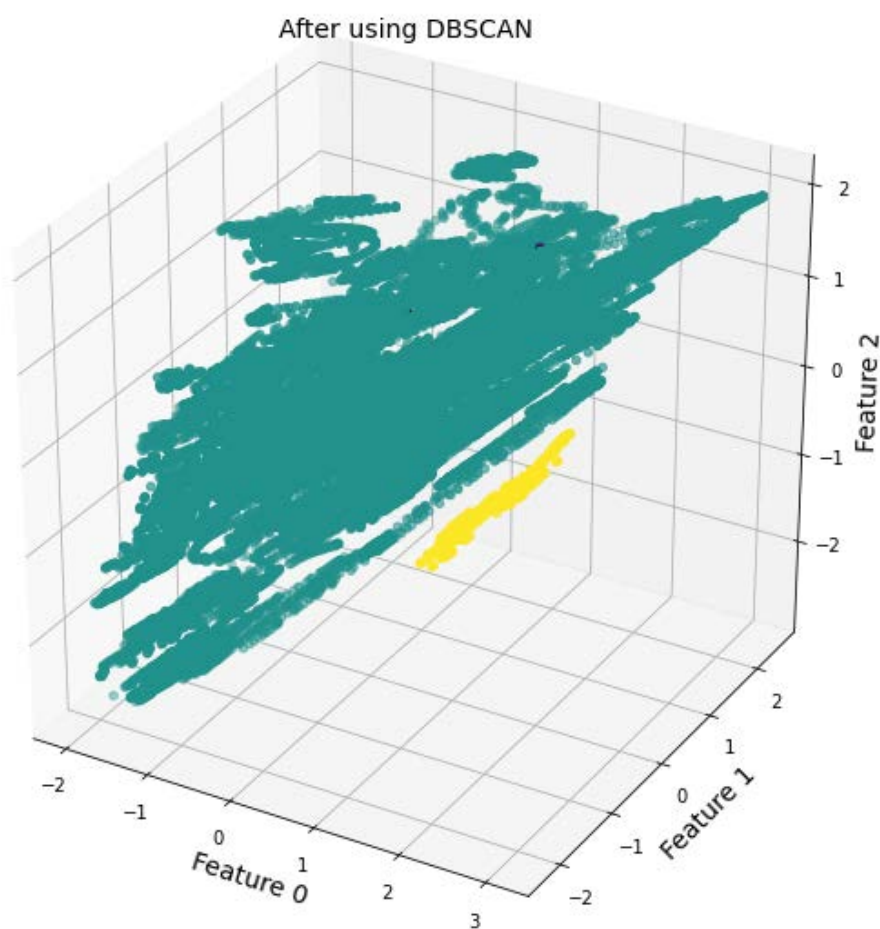
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y_dbscan, s=20, cmap='viridis')
plt.title('After using DBSCAN', font_dict)
plt.xlabel('Feature 0', font_dict)
plt.ylabel('Feature 1', font_dict)
plt.show()
```



สร้างกราฟ Feature เป็น 3 มิติ หลังจากทำ k-Means Clustering

```
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection = '3d')
font_dict = {'family': 'DejaVu Sans', 'size': 14}

ax.scatter(X_scaled[:, 0], X_scaled[:, 1], X_scaled[:, 2], c=y_dbscan, s=20, cmap='viridis')
ax.set_title('After using DBSCAN', font_dict)
ax.set_xlabel('Feature 0', font_dict)
ax.set_ylabel('Feature 1', font_dict)
ax.set_zlabel('Feature 2', font_dict)
plt.show()
```



จากกราฟพบว่า กราฟมีการจัดกลุ่มกันแบบแบ่งอาณาเขตได้อย่างชัดเจน แต่อาจมีข้อมูลบางส่วนหลงเหลืออยู่ด้วย

Clustering Evaluation

```
from sklearn.metrics.cluster import adjusted_rand_score

print("ARI k-Means: {:.4f}".format(adjusted_rand_score(y, y_kmeans)))
print("ARI DBSCAN: {:.4f}".format(adjusted_rand_score(y, y_dbscan)))
```

```
ARI k-Means: 0.0319
ARI DBSCAN: 0.0008
```

```
from sklearn.metrics.cluster import silhouette_score

print("Silhouette Score k-Means: {:.4f}".format(silhouette_score(X_scaled, y_kmeans)))
print("Silhouette Score DBSCAN: {:.4f}".format(silhouette_score(X_scaled, y_dbscan)))
```

```
Silhouette Score k-Means: 0.3615
Silhouette Score DBSCAN: 0.0064
```

จากการประเมินโมเดลในการทำ clustering จะพบว่าค่า ARI ของ k-Means กับ DBSCAN มีค่าน้อยมาก ๆ ส่วน Silhouette Score ก็มีค่าน้อยมาก ๆ เช่นกัน ทั้งสองโมเดลไม่ค่อยดีนัก จึงทำให้ไม่เหมาะแก่การใช้งานจริง

Cross-Validation

การทำ Cross-Validation เราจะใช้ GridSearchCV ที่จะหาค่าพารามิเตอร์ในแต่ละ algorithm ที่ดีที่สุด แล้วนำมาทำ Model Evaluation เพื่อหาค่าพารามิเตอร์ที่หลังจากทำ GridSearchCV ที่เหมาะสมแก่การทำโมเดลที่ดีที่สุด แล้วนำมาใช้งานจริง

```
name_cv = ["KNeighborsClassifier", "SVC", "Neural Network", "Decision Tree", "Random Forest"]
```

```
from sklearn.model_selection import GridSearchCV
```

```
param_knn = {'n_neighbors':np.arange(1, 16)}
```

```
param_svc = {'kernel':['rbf'], 'C':[0.001, 0.01, 1, 100], 'gamma':[0.001, 0.01, 1, 100]}
```

```
param_neural_network = {'max_iter':[1000], 'hidden_layer_sizes':[[10,10], [50, 50], [100, 100]]}
```

```
param_tree = {'max_depth':np.arange(1,16)}
```

```
param_forest = {'n_estimators':np.arange(1,101), 'n_jobs':[-1]}
```

```
param = [param_knn, param_svc, param_neural_network, param_tree, param_forest]
```

```
algo_cv = [KNeighborsClassifier(), SVC(), MLPClassifier(), DecisionTreeClassifier(), RandomForestClassifier()]
```

```
cv = []
```

```
for i in range(len(algo_cv)):
```

```
    cv.append(GridSearchCV(algo_cv[i], param[i]).fit(X_train_scaled, y_train))
```

```
best_param = []
```

```
for i in range(len(algo_cv)):
```

```
    print(cv[i].best_params_)
```

```
    best_param.append(cv[i].best_params_)
```

```
{'n_neighbors': 1}
```

```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

```
{'hidden_layer_sizes': [100, 100], 'max_iter': 1000}
```

```
{'max_depth': 15}
```

```
{'n_estimators': 100, 'n_jobs': -1}
```

สร้างฟังก์ชันของ GridSearchCV เพื่อใช้ในการหาค่าพารามิเตอร์แต่ละ Algorithm ที่ดีที่สุด จากนั้นแสดงค่าพารามิเตอร์ที่ดีที่สุดในแต่ละ algorithm หลังจากทำ GridSearchCV

Model Evaluation

ในการทำ Model Evaluation เพื่อที่จะประเมินว่า algorithm ไหนเหมาะสมในการใช้งานจริง แต่เนื่องจาก target ของชุดข้อมูลนี้มีจำนวน target 70 ตัว ทำให้มีความยุ่งยากในการประเมินโมเดล ดังนั้นจึงจะต้องลดจำนวน target ให้เหลือสองตัว โดยการทำเป็น negative class ช่วยให้ประเมินได้ง่ายขึ้น และเข้าใจง่าย

ขั้นตอนการสร้างโค้ดในการประเมินโมเดล

นำ X_train_scaled, X_test_scaled และ y_train, y_test มาต่อกันเป็น X และ y ตามลำดับ เพื่อที่จะสร้างชุดการทดสอบใหม่ในการประเมิน

```
X_new = np.concatenate((X_train_scaled, X_test_scaled))
y_new = np.concatenate((y_train, y_test))
```

เก็บค่า y_new เพื่อที่จะเก็บค่า boolean เมื่อ target เป็น 0 (Apple Braeburn) โดยกำหนดให้ Apple Braeburn เป็นค่า true และ not Apple Braeburn เป็น false

```
y_new = y_new == 0
```

ทำการแยกข้อมูลใหม่ออกเป็น train และ test

```
from sklearn.model_selection import train_test_split
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X_new, y_new, random_state = 21)
```

สร้างตัวแปรต่างๆ เพื่อเก็บค่าพารามิเตอร์หลังจากทำ GridSearchCV เสร็จแล้ว

```
best_n_neighbors = best_param[0]['n_neighbors']

best_c = best_param[1]['C']
best_gamma = best_param[1]['gamma']

best_hidden_layer_sizes = best_param[2]['hidden_layer_sizes']

best_max_depth = best_param[3]['max_depth']

best_n_estimators = best_param[4]['n_estimators']
```

นำพารามิเตอร์ที่ดีที่สุดนำมาใส่ใน Algorithm ต่างๆ เพื่อที่จะ train โมเดล และหาค่า test score และ ค่าความน่าจะเป็นของ test data

```
knn = KNeighborsClassifier(n_neighbors=best_n_neighbors).fit(X_train_new, y_train_new)
knn_pred = knn.predict(X_test_new)
knn_pred_proba = knn.predict_proba(X_test_new)

nb = GaussianNB().fit(X_train_new, y_train_new)
nb_pred = nb.predict(X_test_new)
nb_pred_proba = nb.predict_proba(X_test_new)

svm = SVC(kernel='rbf', C=best_c, gamma=best_gamma, probability=True).fit(X_train_new, y_train_new)
svm_pred = svm.predict(X_test_new)
svm_pred_proba = svm.predict_proba(X_test_new)

mlp_adam = MLPClassifier(solver='adam', max_iter=1000, random_state=0, hidden_layer_sizes=best_hidden_layer_sizes).fit(X_train_new, y_train_new)
mlp_pred = mlp_adam.predict(X_test_new)
mlp_pred_proba = mlp_adam.predict_proba(X_test_new)

tree = DecisionTreeClassifier(max_depth=best_max_depth, random_state=0).fit(X_train_new, y_train_new)
tree_pred = tree.predict(X_test_new)
tree_pred_proba = tree.predict_proba(X_test_new)

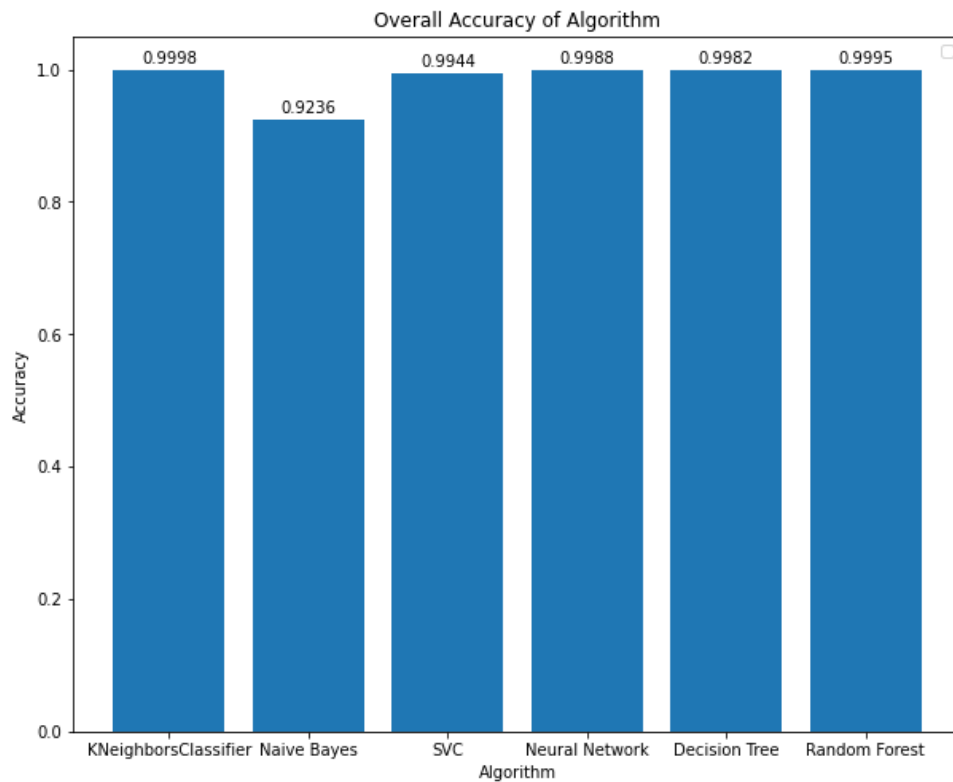
forest = RandomForestClassifier(n_estimators=best_n_estimators, n_jobs=-1, random_state=0).fit(X_train_new, y_train_new)
forest_pred = forest.predict(X_test_new)
forest_pred_proba = forest.predict_proba(X_test_new)
```

สร้าง list y_pred กับ y_pred_proba เพื่อเก็บ list ของ target และค่าความน่าจะเป็น หลังจากทำนายโมเดลต่างๆ ตามลำดับ

```
y_pred = [knn_pred, nb_pred, svm_pred, mlp_pred, tree_pred, forest_pred]
```

```
y_pred_proba = [knn_pred_proba, nb_pred_proba, svm_pred_proba, mlp_pred_proba, tree_pred_proba, forest_pred_proba]
```

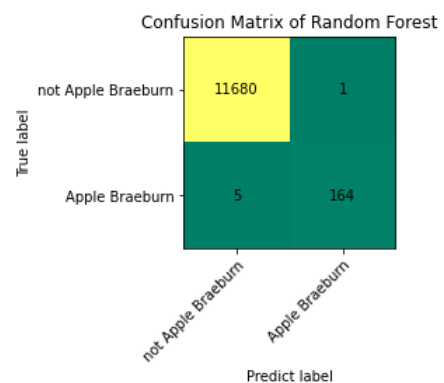
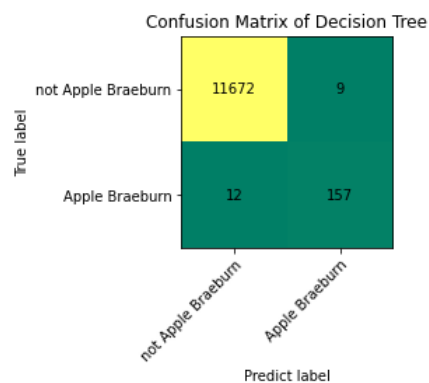
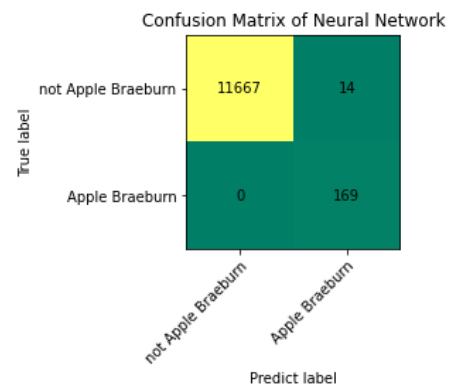
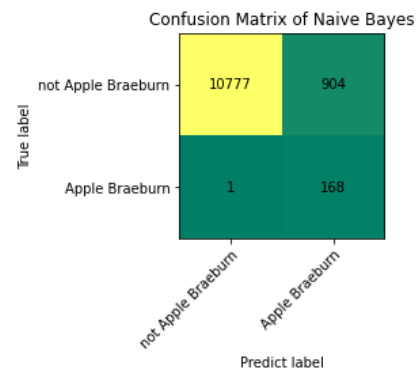
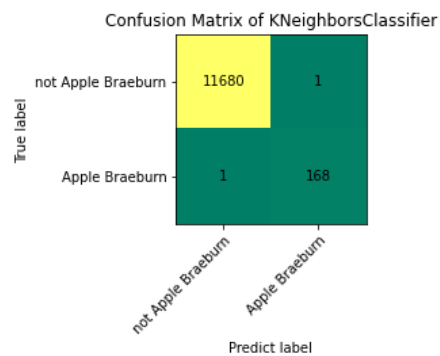
แสดงกราฟค่า Accuracy ของ test score



จากการพบว่า k-Nearest Neighbor มีค่า accuracy มากที่สุด รองลงมาจะเป็น Neural Network และ Algorithm ที่มีค่า accuracy น้อยที่สุดนั่นก็คือ Naïve Bayes ที่มีค่า 0.9236

ค่า accuracy ที่กล่าวไปข้างต้น เรามักจะไม่ประเมินผลจากค่า accuracy เนื่องจากค่า accuracy สามารถปรับเปลี่ยนได้เอง จึงทำให้ไม่สามารถวัดประสิทธิภาพของโมเดลได้ จำเป็นต้องใช้ Confusion metrix, F1-score, precision, recall ในการประเมินผลโมเดลได้แม่นยำที่สุด

สร้าง Confusion Metrix ในทุกๆ Algorithm



จาก confusion matrix จะพบว่าโมเดลของ k-Nearest Neighbor มีการทายถูกมากที่สุด รองลงมาจะเป็น Random Forest ที่มีการทายผิดไป 6 รูป ส่วนโมเดลที่การทายผิดมากที่สุดคือ Naïve Bayes มีการทายรูปผิดมากถึง 905 รูป โดยสรุปว่า โมเดลที่มีการทายผิดยิ่งมากเท่าไร โมเดลนั้นจะมีประสิทธิภาพที่ไม่ดีนัก

F1-score

Algorithm	F1-score
k-Nearest Neighbor	1.00
Naïve Bayes	0.92
SVC	0.99
Neural Network	1.00
Decision Tree	1.00
Random Forest	1.00

จากตารางจะพบว่า ทั้ง Algorithm มีค่า F1-score มากที่สุดยกเว้น SVC กับ Naïve Bayes ที่มีค่า 0.99 กับ 0.92 ตามลำดับ ทำให้ทั้ง 4 Algorithm มีประสิทธิภาพมากที่สุด

Precision & Recall

k-Nearest Neighbor

	Precision	Recall	F1-score	Support
not Apple Braeburn	1.00	1.00	1.00	11681
Apple Braeburn	0.99	0.99	0.99	169

Naïve Bayes

	Precision	Recall	F1-score	Support
not Apple Braeburn	1.00	0.92	0.96	11681
Apple Braeburn	0.16	0.99	0.27	169

SVC

	Precision	Recall	F1-score	Support
not Apple Braeburn	1.00	1.00	1.00	11681
Apple Braeburn	0.83	0.76	0.80	169

Neural Network

	Precision	Recall	F1-score	Support
not Apple Braeburn	1.00	1.00	1.00	11681
Apple Braeburn	0.92	1.00	0.96	169

Decision Tree

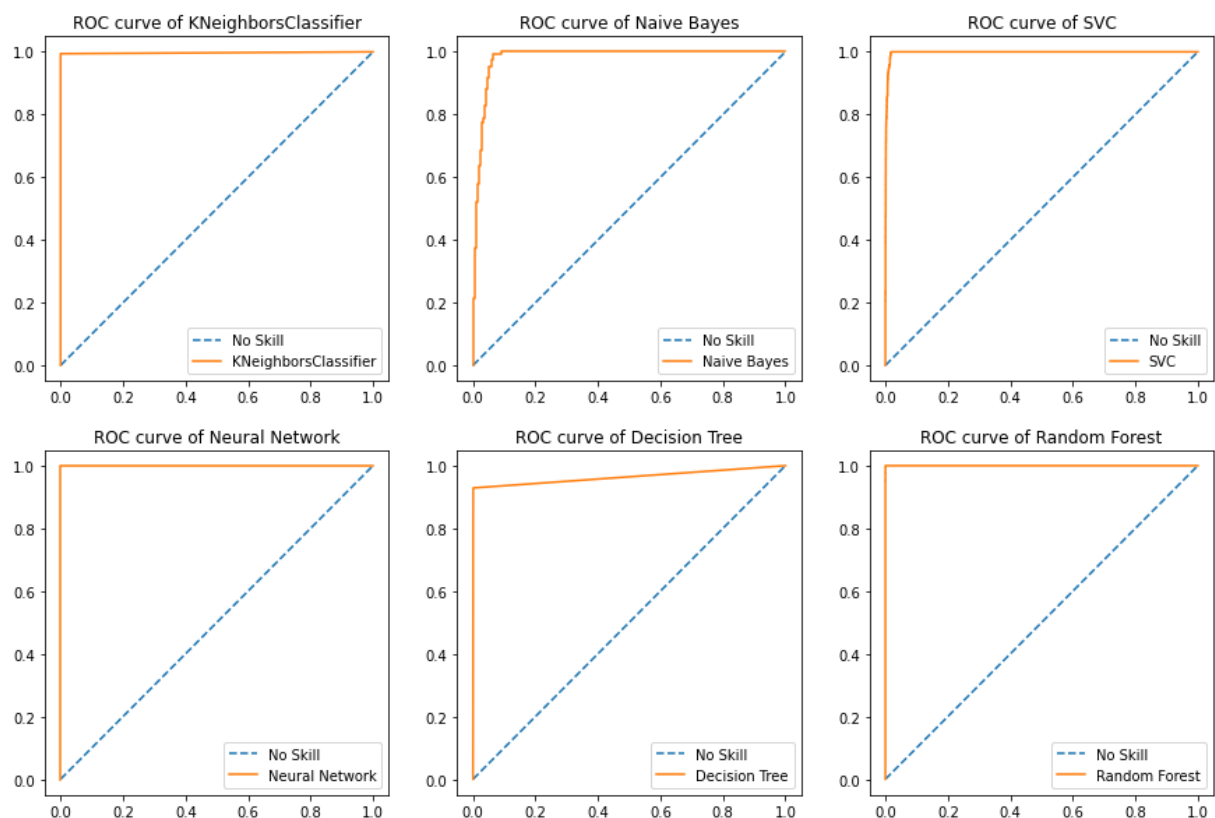
	Precision	Recall	F1-score	Support
not Apple Braeburn	1.00	1.00	1.00	11681
Apple Braeburn	0.95	0.93	0.94	169

Random Forest

	Precision	Recall	F1-score	Support
not Apple Braeburn	1.00	1.00	1.00	11681
Apple Braeburn	0.99	0.97	0.98	169

จากตารางพบว่า k-Nearest Neighbor มี Precision กับ Recall มากที่สุด กล่าวคือ Algorithm นี้มีการตรวจจับข้อมูลที่ทำนายเป็น Apple Braeburn และไม่เป็น Apple Braeburn ได้ดีมาก ส่วน Naïve Bayes ที่มี Precision กับ Recall มากที่สุดใน class ของ not Apple Braeburn แต่มีค่า Precision น้อยมาก ๆ ใน class ของ Apple Braeburn แสดงว่า Algorithm นี้มีการตรวจจับข้อมูลที่ทำนายว่าเป็น Apple Braeburn ไม่ดีนัก ถึงแม้ว่าใน class ของ not Apple Braeburn มีความ bias อยู่

ROC curve



จากกราฟพบว่าในกราฟของ k-Nearest Neighbor, Neural Network, Decision Tree และ Random Forest จุด ideal point จะเข้าหามุมบนขวาสุด แสดงว่า True positive มีค่าสูงที่สุด

Save โมเดล

จากการประเมินของโมเดลทุกๆ ตัวจะพบว่า k-Nearest Neighbor เป็นโมเดลที่เหมาะสมแก่การใช้งานจริงมากที่สุด จากนั้นนำมา save โมเดลโดยใช้ pickle แล้ว save เป็น my_model.pkl

```
best_model = KNeighborsClassifier(n_neighbors=1).fit(X_train_scaled, y_train)
```

```
import pickle  
pickle.dump(best_model, open('my_model.pkl', 'wb'))
```


การนำไปใช้จริง

โหลดตัวโมเดลหลังจาก train กับชุดข้อมูลไว้แล้ว

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
import cv2
from google.colab.patches import cv2_imshow
```

```
import pickle

FILE_PATH = '/content/drive/MyDrive/fruits_dataset/rgb_dataset/my_model.pkl'
model = pickle.load(open(FILE_PATH, 'rb'))
```

ทำการครอบรูปเพื่อที่จะครอบรูปเฉพาะรูปผลไม้ที่ต้องการที่ใช้ และจะไม่เลือกรูปที่ตรวจจับผิดพลาด

```
def crop_image(img):
    result = img.copy()
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    _, binary = cv2.threshold(gray, 225, 255, cv2.THRESH_BINARY_INV)
    contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    arr = []
    res = []
    for c in contours:
        x,y,w,h = cv2.boundingRect(c)
        res.append(x+y+w+h)
        arr.append([x,y,w,h])

    for i in range(len(contours)):
        if sum(arr[i]) == max(res):
            x,y,w,h = arr[i]

    cv2.rectangle(img, (x, y), (x + w, y + h), (36,255,12), 3)
    img_result = result[y:y+h, x:x+w]
    return img_result
```

สร้าง dictionary ของชื่อผลไม้ต่างๆ

```
fruit_dict = {0: 'Apple Braeburn', 1: 'Apple Crimson Snow', 2: 'Apple Golden 1', 3: 'Apple Golden 2', 4: 'Apple Golden 3',
5: 'Apple Granny Smith', 6: 'Apple Pink Lady', 7: 'Apple Red 1', 8: 'Apple Red 2', 9: 'Apple Red 3',
10: 'Apple Red Delicious', 11: 'Apple Red Yellow 1', 12: 'Apple Red Yellow 2', 13: 'Apricot', 14: 'Avocado',
15: 'Banana', 16: 'Banana Lady Finger', 17: 'Blueberry', 18: 'Cantaloupe 1', 19: 'Cantaloupe 2',
20: 'Carambola', 21: 'Cherry 1', 22: 'Cherry 2', 23: 'Cherry Rainier', 24: 'Cherry Wax Black',
25: 'Cherry Wax Red', 26: 'Cherry Wax Yellow', 27: 'Clementine', 28: 'Dates', 29: 'Ginger Root',
30: 'Grape Blue', 31: 'Grape Pink', 32: 'Grape White', 33: 'Grape White 2', 34: 'Grape White 3',
35: 'Grape White 4', 36: 'Guava', 37: 'Kiwi', 38: 'Lemon', 39: 'Lemon Meyer',
40: 'Limes', 41: 'Lychee', 42: 'Mandarine', 43: 'Mango', 44: 'Mango Red',
45: 'Mangostan', 46: 'Mulberry', 47: 'Orange', 48: 'Papaya', 49: 'Passion Fruit',
50: 'Peach', 51: 'Peach 2', 52: 'Peach Flat', 53: 'Pear', 54: 'Pear 2',
55: 'Pear Kaiser', 56: 'Pear Monster', 57: 'Pear Red', 58: 'Pear Stone', 59: 'Pear Williams',
60: 'Pineapple', 61: 'Pineapple Mini', 62: 'Plum', 63: 'Plum 2', 64: 'Pomegranate',
65: 'Raspberry', 66: 'Salak', 67: 'Strawberry', 68: 'Strawberry Wedge', 69: 'Watermelon'}
```

จากนั้นนำมาแยกสีของรูปภาพออกมาเป็นสีแดง, สีเขียว, สีน้ำเงินตามลำดับ แล้วใช้หลักสถิติต่างๆมาคำนวณค่าสีทั้ง 3 สี จากนั้น Scaling โดยใช้ Standard Scaler สุดท้ายนำมาทำนายผลลัพธ์

```
def image_predict(image):
    from scipy.stats import sem
    from sklearn.preprocessing import StandardScaler

    cp_img = crop_image(image)
    (R, G, B) = cv2.split(cp_img)
    X = np.array([np.mean(R), np.mean(G), np.mean(B),
                  np.std(R), np.std(G), np.std(B),
                  np.median(R), np.median(G), np.median(B),
                  sem(R, axis=None, ddof=0), sem(G, axis=None, ddof=0), sem(B, axis=None, ddof=0),
                  np.percentile(R, 25), np.percentile(G, 25), np.percentile(B, 25),
                  np.percentile(R, 50), np.percentile(G, 50), np.percentile(B, 50),
                  np.percentile(R, 75), np.percentile(G, 75), np.percentile(B, 75)])

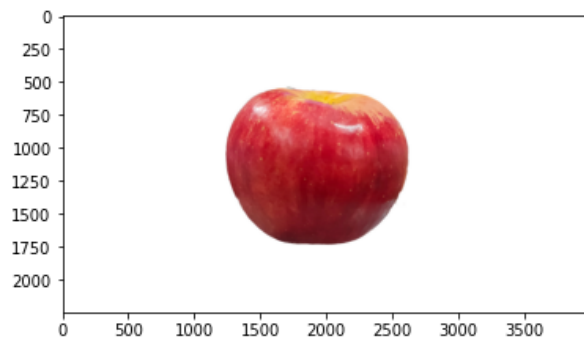
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X.reshape(1, -1))

    y_pred = model.predict(X_scaled)
    img_result = fruit_dict[y_pred[0]]

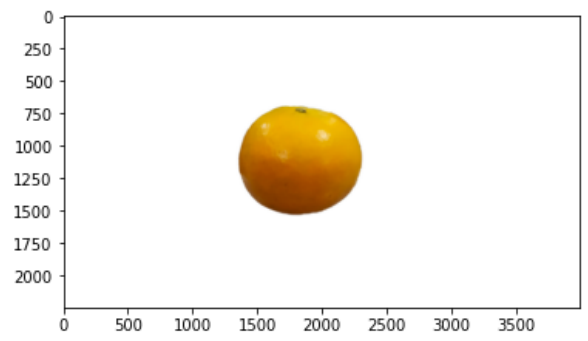
    return img_result
```

แสดงผลลัพธ์หลังจากทำนายกับโมเดล

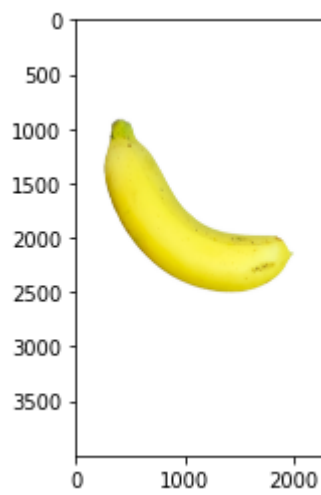
Apple Red 3



Apple Red 3



Apple Red 3



สรุป

จากการทดลองกับข้อมูลรูปภาพจริง จะพบว่าเมื่อเรานำข้อมูลไปให้โมเดลทำนาย โมเดลจะทำนายรูปภาพสีไม่ค่อยดีนักใน Machine Learning เพราะว่า Machine Learning มักจะ train กับข้อมูลที่ไม่ซับซ้อนหรือรูปภาพที่เป็นภาพระดับสีเทา (Greyscale) จึงทำให้ผลลัพธ์ไม่ค่อยดีเท่าที่ควร ซึ่งไม่เหมือนกับ Deep Learning ที่สามารถทำนายภาพสีโดยตรงได้หรือทำงานกับข้อมูลที่ซับซ้อนได้

ความเกี่ยวข้องกับปัญญาประดิษฐ์

การจำแนกประเภทผลไม้มักใช้ความรู้ด้าน k-Nearest Neighbor, Neural Network และ SVC ซึ่งจำเป็นต่อการเรียนรู้ของโปรแกรม และเกี่ยวข้องกับปัญญาประดิษฐ์ ที่จะทำให้โปรแกรมสามารถจำแนกชนิดผลไม้ได้อย่างถูกต้องและแม่นยำ แล้วสามารถนำไปต่อยอดไปทำเป็น Deep Learning ได้

บรรณานุกรม

Renu Khandelwal, "Loading Custom Image Dataset for Deep Learning Models: Part 1," Accessed November 10, 2021, <https://towardsdatascience.com/loading-custom-image-dataset-for-deep-learning-models-part-1-d64fa7aeca6>.

Iqbal Hussain, "A quick guide to color image compression using PCA in python," Accessed November 10, 2021, <https://towardsdatascience.com/dimensionality-reduction-of-a-color-photo-splitting-into-rgb-channels-using-pca-algorithm-in-python-ba01580a1118>.

Adrian Rosebrock, "Machine Learning in Python," Accessed November 10, 2021, <https://www.pyimagesearch.com/2019/01/14/machine-learning-in-python/>.

Matplotlib, "Creating annotated heatmaps," Accessed November 17, 2021, https://matplotlib.org/stable/gallery/images_contours_and_fields/image_annotated_heatmap.html.

Zach, "How to Plot a ROC Curve in Python (Step-by-Step)," Accessed November 17, 2021, <https://www.statology.org/plot-roc-curve-python/>.

Jason Brownlee, "How to Use ROC Curves and Precision-Recall Curves for Classification in Python," Accessed November 18, 2021, <https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/>.

Jason Brownlee, "Save and Load Machine Learning Models in Python with scikit-learn," Accessed November 16, 2021, <https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/>.

Praveen, "Edge Detection with OpenCV," Accessed November 19, 2021,
https://github.com/cloudxlab/opencv-intro/blob/master/6_edge_detection.ipynb.

Abdou Rockikz, "How to Detect Contours in Images using OpenCV in Python,"
Accessed November 19, 2021, <https://www.thepythoncode.com/article/contour-detection-opencv-python>.