Midterm project Mangoes level classification

一、Data preprocessing

1. 壓縮圖片

```
compressImage("./data/Train/","./data/Train_compression/")     # 壓縮芒果圖片
path='./data/Train_compression/'
files=os.listdir(path)
files.sort()
```

排序檔案

```
#壓縮圖片
def compressImage(srcPath,dstPath):
    for filename in os.listdir(srcPath):
        if not os.path.exists(dstPath):
                os.makedirs(dstPath)

        #完整的檔案或資料夾路徑
        srcFile=os.path.join(srcPath,filename)
        dstFile=os.path.join(dstPath,filename)

        # 如果是檔案就處理
        if os.path.isfile(srcFile):

             if os.path.isfile(srcFile):
                try:
                    sImg=Image.open(srcFile)
                    dImg=sImg.resize((100,100),Image.ANTIALIAS)  # 設定壓縮尺寸
                    dImg.save(dstFile)
                except Exception:
                    print(dstFile+"fail")

        # 如果是資料夾就遞迴
        if os.path.isdir(srcFile):
            compressImage(srcFile, dstFile)
```

因為擔心過度壓縮的圖片，會失去一些重要的特徵，所以我盡量在滿足有限記憶體的情況下，將壓縮後的圖片大小設為 100，但同時資料處理時間就會變長。

2. 讀檔

```
#讀檔
for file in files:

    p=path+file
    img = cv2.imread(p)
    img_2D = np.reshape(img,(-1,3))
    img_1D = np.reshape(img_2D,(-1,1))

    img_1D = img_1D.flatten()
    img_1D = img_1D.tolist()

    img_list.append(img_1D)
img_arr = np.array(img_list)
```

先將每一張照片樣本的所有元素拉為一為陣列，再將代表每一張照片的 array 組成 matrix，最後一起丟入 PCA 進行處理。

3. PCA

```
pca = PCA(n_components=100)
pca.fit(img_arr)
img_arr_pca = pca.transform(img_arr)
```

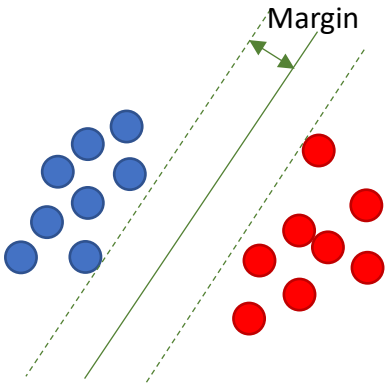
一開始做 PCA 的效果並不明顯，後來嘗試調整 n_conponent 參數，在 100 時效果最好。

二、Classifier

1.

SVC

```
clf = svm.SVC(kernel = 'poly', degree = 5, C = 0.9)
clf.fit(img_arr_pca, label_list)
```



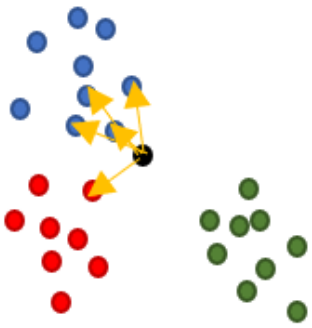
SVC 原理如上圖所示，找出分類線使其至各分類點的最短距離也就是 margin 越大越好。實做過程中發現 kernel 採用 poly 多項式的效果最好，其中 degree 參數代表的是多項式的次數，設為 5 是不希望有過擬合的情況發生，而 C 則是懲罰係數，也就是對誤差的容忍程度，這裡設為比較小的 0.9，也是為了防止過擬和現象。

2.

KNN

```
In [23]:  1 knn = KNeighborsClassifier(n_neighbors=10, weights='distance')
          2 knn.fit(img_arr_pca, label_list)

Out[23]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=1, n_neighbors=10, p=2,
                              weights='distance')
```



KNN 原理如上圖所示，對目標分類點取 k 個距離最相近的點，然後根據這 k 個點中，絕大多數分屬的類別決定目標點被歸為哪類。實做過程中發現 k 值也就是 n_neighbors 參數值調整為 10 效果最好，此外查閱了相關資料後發現 KNN 分類器缺點之一包含了當樣本分佈不均時，容易出現分類錯誤，所以將 weight 參數調成 distance 模式，讓 knn 在進行分類工作時按距離進行加權。

3.

Random Forest

```
In [25]:  1 rf1 = RandomForestClassifier(n_estimators=1000)
          2 rf1.fit(img_arr_pca, label_list)

Out[25]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=None, max_features='auto', max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=1,
                                oob_score=False, random_state=None, verbose=0,
                                warm_start=False)
```

Random Forest 的原理為從原本的資料集中隨機取 k 份 n 個資料建立 k 個

decision tree，至於 decision tree 作為分類依據的特徵是根據信息增益來判斷，當信息增益越大，代表該特徵對分類越重要。然後在建立完 k 個 decision tree 後，就採用多數決方法得到最佳的預測值。實做過程中透過調整 n_estimators 參數也就是 k 值，使模型更複雜，預測效果更好，但相對所需執行時間增長，最後我將其定為 1000，另外我也有試著增加 min_samples_leaf 參數，卻無法提升結果正確率。

三、Improvement

多數決

```
predict_final = []
for i in range(len(predict1)):
    if predict1[i] == predict3[i]:
        predict_final.append(predict1[i])
    elif predict1[i] == predict4[i]:
        predict_final.append(predict1[i])
    elif predict3[i] == predict4[i]:
        predict_final.append(predict4[i])
    else:
        predict_final.append(predict4[i])

```

參考 random forest 最後採用的多數決，我將多個 classifier 預測的結果互相比較，重複次數最高的就是多數決的結果，如果各個結果票數相同就選擇在 validation 時準確率最高的 classifier 的答案作為結果輸出，結果略有提升。

四、Difficulties

1.

```
kfold = sklearn.model_selection.KFold(n_splits=10, shuffle = True, random_state=7)
results = sklearn.model_selection.cross_val_score(rf1, img_arr_pca, label_list, cv=kfold)
print(results.mean())
```

因為每天上傳次數有限，所以必須在 training 的時候就先判斷分類器的結果好壞，利用 sklearn 建立 10 個 fold，輪流做 validation 然後取平均作為預測結果。

2.

不知道為什麼最後一天在做的時候，沒隔多久就會斷線，而且我也不是用 CPU 跑，讓我覺得很無奈，因為有的東西都已經跑很久了。