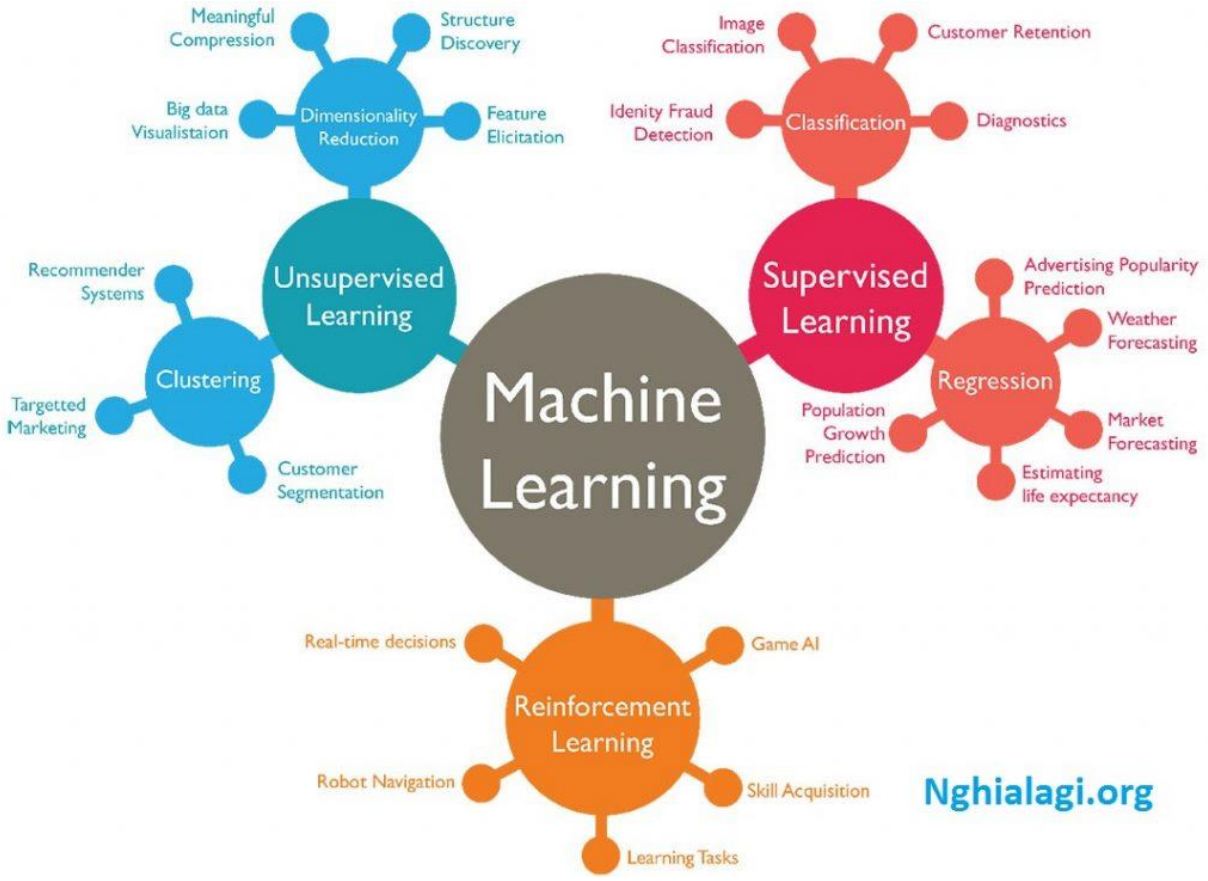
BÁO CÁO BÀI TẬP

MACHINE LEARNING CƠ BẢN



Sinh viên thực hiện : Trần Đặng Trung Đức

TPHCM, ngày 16 tháng 09 năm 2020

MỤC LỤC

BÀI TẬP

Đề bài 1:
Cho dữ liệu iris.data (chiều dài của cánh hoa, chiều rộng của cánh hoa, chiều dài của đài hoa, chiều rộng của đài hoa, phân lớp)
Bộ dữ liệu 150 mẫu, với 3 loài hoa, mỗi loại 50 mẫu
Hãy dự đoán trên bộ dữ liệu này
Yêu cầu
1- Chuẩn hóa dữ liệu về [0,1]
2- Chia dữ liệu train/test theo tỉ lệ 8:2
3- Sử dụng 4 giải thuật: Linear, Bayes, KNN và SVM
4- Đánh giá 4 giải thuật đó bằng các độ đo: Accuracy, Confusion Matrix và F1 score, giải thích kết quả đạt được
5- Sử dụng giải thuật cho kết quả cao nhất để dự đoán từ 1 dữ liệu

Trích ví dụ mẫu từ tập dữ liệu cung cấp

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
```

Đề bài 2:
Cho dữ liệu ảnh như ở file đính kèm (với 2 người Bill Gate và Mark Zuckerberg)
Vào 1 ảnh ra ảnh đó là ai
Yêu cầu
+ Chia dữ liệu thành 2 phần: Train/Test
+ Trích xuất đặc trưng
+ Đưa vào mô hình huấn luyện SVM
+ Đánh giá kết quả bằng accuracy
+ Lưu mô hình huấn luyện
+ Dự đoán từ 1 ảnh vào bất kỳ

Trích ví dụ ảnh từ dữ liệu cung cấp

GIẢI BÀI 1

Tham khảo lý thuyết và code có chỉnh sửa trong tài liệu [1]

Bước 1: Khai báo các thư viện cần sử dụng.

```
import numpy as np                              # Thư viện tính toán với ma trận
import matplotlib.pyplot as plt                 # Thư viện trực quan hóa dữ liệu
from sklearn import neighbors                   # Thư viện KNN
import pandas as pd                             # Thư viện đọc dữ liệu
from sklearn.model_selection import train_test_split  # Hàm tách dữ liệu
from sklearn.metrics import accuracy_score      # Hàm đánh giá độ chính xác
from sklearn import linear_model                # Linear model
from sklearn.svm import SVC                     # SVM
from sklearn.naive_bayes import MultinomialNB # Bayes
from scipy import sparse                        # Lưu ma trận dạng Sparse
from sklearn.naive_bayes import GaussianNB      # Phân phối GaussianNB
from sklearn import metrics                     # Thư viện metrics đánh giá model
import matplotlib.pyplot as plt                 # Thư viện trực quan hóa
from sklearn import svm                         # SVM
```

Bước 2: Đọc dữ liệu hoa Iris bằng hàm pd từ thư viện pandas, đặt tên cho các chiều dữ liệu và nhãn hoa từ file iris.data.

```
# Load data
iris_cols= ['CDCH','CRCH' ,'CDDH','CRDH','PLH']
iris_sam = pd.read_csv('iris.data', sep=',', names=iris_cols, encoding='latin-1')
X0 = iris_sam.to_numpy()                    # Chuyển dữ liệu thành matrix
N = X0.shape[0]                             # Số lượng mẫu
iris_feature = X0[:,0:4]                    # Ma trận chứa feature vector của hoa
d = iris_feature.shape[1]                   # Số chiều dữ liệu
labels_count= X0[:,-1]                      # Nhãn hoa Iris
C=3
labels_name= ['Iris-setosa','Iris-versicolor','Iris-virginica']
```

Bước 3: Gán các loại hoa Iris tương ứng vào các giá trị 0, 1, 2 để dễ phân loại.

```
# Gán nhãn cho hoa Iris là 0, 1, 2
labels_feature = []
for lb in labels_count:
    if lb == 'Iris-setosa':
        labels_feature.append(0)
    elif lb == 'Iris-versicolor':
        labels_feature.append(1)
    else:
        labels_feature.append(2)
labels_feature = np.array(labels_feature)
```

Bước 4: Chuẩn hóa dữ liệu bằng phương pháp Rescaling, sau đó dùng train_test_split để tách dữ liệu thành 2 tập train/test theo tỷ lệ 8:2. Viết hàm Confusion matrix thủ công.

```
# Chuẩn hóa dữ liệu
iris_feature_nor = np.copy(iris_feature)
Min_col = np.min(iris_feature,0)                    # Max của các feature
Max_col = np.max(iris_feature,0)                    # Min của các feature
Max_Min = Max_col-Min_col                           # Max trừ Min
for i in range(N):                                  # Chuẩn hóa theo Rescaling
    for j in range(d):
        iris_feature_nor[i,j] = (iris_feature_nor[i,j]-Min_col[j])/Max_Min[j]
iris_feature_nor=np.round(iris_feature_nor.tolist(),3) # Làm tròn dữ liệu
# Tách dữ liệu thành 2 tập train và set tỉ lệ 8:2
X_train, X_test, y_train, y_test = train_test_split(
     iris_feature_nor, labels_feature, test_size=30)
# Hàm tạo confusion matrix
def confusion_matrix(y_true, y_pred):
    N = np.unique(y_true).shape[0]    # Số lượng Class
    cm = np.zeros((N, N))             # Khởi tạo ma trận Confusion matrix
    for n in range(y_true.shape[0]):
        cm[y_true[n], y_pred[n]] += 1
    return cm
```

Bước 5: Giải bài toán phân loại hoa Iris bằng các phương pháp khác nhau.

1. *Model K-NN*
   - Thuật toán: K-NN là phương pháp dự đoán nhãn của một điểm dữ liệu dựa trên K điểm gần nó nhất đã được phân loại.
   - Cách làm: sử dụng thư viện có sẵn của scikit-learn để train.
   - Đánh giá kết quả bằng Accuracy, Confusion matrix và F1 – Score

```
# KNN
# Chọn 3 điểm gần nhất và tính khoảng cách theo chuẩn 2
def KNN_lib(X_train,y_train):
    # Sử dụng thư viện scikit-learn KNN để phân loại
    clf = neighbors.KNeighborsClassifier(n_neighbors = 3, p = 2, weights = 'distance')
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    # Kết quả 30 điểm dữ liệu lấy từ tập test
    print("Print results for 30 test data points:")
    print("Predicted labels: ", y_pred[0:30])
    print("Ground truth    : ", y_test[0:30])
    # Đánh giá kết quả bằng hàm accuracy_score
    print("Accuracy of 3NN: %.2f %%" %(100*accuracy_score(y_test, y_pred)))
    cnf_matrix = confusion_matrix(y_test, y_pred)    # Tạo CM
    print('Confusion matrix:')
    print(cnf_matrix)
    print('Accuracy:', 100*(np.diagonal(cnf_matrix).sum()/cnf_matrix.sum()),'%')
    normalized_confusion_matrix = cnf_matrix/cnf_matrix.sum(axis = 1, keepdims = True)
    print('\nConfusion matrix (with normalization):')
    print(normalized_confusion_matrix)
    print('Precision Recall and F1-Score')
    print(metrics.classification_report(y_test, y_pred, digits=3))
```

- Nhận xét: kết quả thu được sau khi chạy code 30 điểm dữ liệu từ tập test.
  + Độ chính xác là 90% với các giá trị điểm F1-Score khá tốt (lớn hơn 0.85).

```
Print results for 30 test data points:
Predicted labels:  [1 1 2 1 2 1 0 1 1 2 2 1 0 2 0 2 2 1 1 0 1 2 0 1 0 1 0 2 2 2]
Ground truth    :  [1 1 2 1 2 2 0 1 2 2 2 1 0 2 0 2 2 1 1 0 1 2 0 1 0 1 0 1 2 2]
Accuracy of 3NN: 90.00 %
Confusion matrix:
[[ 7.  0.  0.]
 [ 0. 10.  1.]
 [ 0.  2. 10.]]
Accuracy: 90.0 %

Confusion matrix (with normalization):
[[1.         0.         0.        ]
 [0.         0.90909091 0.09090909]
 [0.         0.16666667 0.83333333]]
Precision Recall and F1-Score
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         7
           1      0.833     0.909     0.870        11
           2      0.909     0.833     0.870        12

    accuracy                          0.900        30
   macro avg      0.914     0.914     0.913        30
weighted avg      0.903     0.900     0.900        30
```

2. *Model Linear*

- Thuật toán Linear Regression: là thuật toán phân loại dùng với các dữ liệu có các dấu hiệu linear.
- Cách làm: sử dụng thư viện có sẵn của scikit-learn để train.
- Đánh giá kết quả bằng Accuracy, Confusion matrix và F1 – Score.

```
# Linear Regression
def LR_vec(X_train,y_train):
    # Sử dụng công thức từ công thức rút ra từ tính đạo hàm trực tiếp
    X_train_lm = np.concatenate((np.ones((X_train.shape[0],1)),X_train),axis=1)
    A = X_train_lm.T@X_train_lm                 # Nhân 2 ma trận X.T và X
    b = X_train_lm.T@y_train.reshape(120,1)     # Nhân 2 ma trận X.T và y
    w = np.linalg.pinv(A)@b                 # Tính w bằng nhân 2 ma trận A-1 và b
    # Sử dụng thư viện scikit-learn
    regr = linear_model.LinearRegression(fit_intercept=False)  # Gồm bias
    regr.fit(X_train_lm, y_train.reshape(X_train_lm.shape[0],1))
    print( 'Solution found by scikit-learn: ', regr.coef_ )
    print( 'Solution by vectorized: ', w.T)
    # Kết quả 30 điểm dữ liệu lấy từ tập test
    X_test_lm =  np.concatenate((np.ones((X_test.shape[0],1)),X_test),axis=1)
    y_pred =  np.int32(np.round(X_test_lm@w))
    print("Print results for 30 test data points:")
    print("Predicted labels: ", y_pred.T[0])
    print("Ground truth    : ", y_test)
    print("Accuracy of LM_vec: %.2f %%" %(100*accuracy_score(y_test,y_pred.T[0])))
    cnf_matrix = confusion_matrix(y_test, y_pred.T[0])
    print('Confusion matrix:')
    print(cnf_matrix)
    print('Accuracy:', 100*(np.diagonal(cnf_matrix).sum()/cnf_matrix.sum()),'%')
    normalized_confusion_matrix = cnf_matrix/cnf_matrix.sum(axis = 1, keepdims = True)
    print('\nConfusion matrix (with normalization:)')
    print(normalized_confusion_matrix)
    print('Precision Recall and F1-Score')
    print(metrics.classification_report(y_test, y_pred, digits=3))
```

- Nhận xét: kết quả thu được sau khi chạy code 30 điểm dữ liệu từ tập test.
  + Độ chính xác là 96,67% với các giá trị điểm F1-Score khá tốt (lớn hơn 0.95).

```
Solution found by scikit-learn:  [[-0.0362519  -0.06202817 -0.2479548   0.9653295
1.56651057]]
Solution by vectorized:  [[-0.0362519  -0.06202817 -0.2479548   0.9653295   1.56651057]]
Print results for 30 test data points:
Predicted labels:  [1 1 1 2 1 1 1 2 1 0 2 0 2 2 0 1 1 1 2 2 2 0 0 1 2 2 0 2 2 0]
Ground truth    :  [1 1 1 2 1 1 1 2 1 0 2 0 2 2 0 1 2 1 2 2 2 0 0 1 2 2 0 2 2 0]
Accuracy of LM_vec: 96.67 %
Confusion matrix:
[[ 7.  0.  0.]
 [ 0. 10.  0.]
 [ 0.  1. 12.]]
Accuracy: 96.66666666666667 %

Confusion matrix (with normalization:)
[[1.         0.         0.        ]
 [0.         1.         0.        ]
 [0.         0.07692308 0.92307692]]
Precision Recall and F1-Score
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         7
           1      0.909     1.000     0.952        10
           2      1.000     0.923     0.960        13

    accuracy                          0.967        30
   macro avg      0.970     0.974     0.971        30
weighted avg      0.970     0.967     0.967        30
```

3. *Model multiclass dùng Support Vector Machine*

- Thuật toán Support Vector Machine: là thuật toán phân loại dùng với các dữ liệu Linear seperable hoặc gần Linear seperable để tìm được biên có Margin lớn nhất.
- Cách làm: viết hàm Vectorized và sử dụng Mini batch để cập nhật W.
- Đánh giá kết quả bằng Accuracy, Confusion matrix và F1 – Score.

Hàm `svm_loss_vectorized(W, X, y, reg)` để tính hàm mất mát và đạo hàm của nó theo phương pháp vectorized

```
# MulticLass SVM dùng vectorized
def svm_loss_vectorized(W, X, y, reg):
    d, C = W.shape                  # Chiều và số Class
    _, N = X.shape                  # N là tập mẫu
    loss = 0                        # Khởi tạo giá trị hàm mất mát bằng 0
    dW = np.zeros_like(W)           # Khởi tạo ma trận đạo hàm hàm mất mát
    Z = W.T@X                       # Score matrix
    correct_class_score = np.choose(y, Z).reshape(N,1).T
    margins = np.maximum(0, Z - correct_class_score + 1) # Higne loss
    margins[y, np.arange(margins.shape[1])] = 0
    loss = np.sum(margins, axis = (0, 1))
    loss /= N
    loss += 0.5 * reg * np.sum(W * W)

    F = (margins > 0).astype(int)
    F[y, np.arange(F.shape[1])] = np.sum(-F, axis = 0)
    dW = X@F.T/N + reg*W
    return loss, dW
```

Hàm cập nhật ma trận thông số W theo Mini-Batch

```
# Mini-Batch
def multiclass_svm_GD(X, y, Winit, reg, lr=.1, \
        batch_size = 100, num_iters = 100):
    W = Winit                                # Giá trị đầu tiên ma trận W
    loss_history = np.zeros((num_iters))     # Khởi tạo hàm mất mát
    for it in range(num_iters):
        # Ngẫu nhiên chọn 1 batch của X
        idx = np.random.choice(X.shape[1], batch_size)
        X_batch = X[:, idx]
        y_batch = y[idx]
        # Tính hàm mất mát và đạo hàm của nó bằng vectorized
        loss_history[it], dW =svm_loss_vectorized(W, X_batch, y_batch, reg)
        W -= lr*dW                           # Công thức cập nhật W
    return W, loss_history
```

Hàm chính cho thuật toán multi_SVM và dự đoán kết quả từ 30 điểm trong tập test

```
def multi_SVM_vec(X_test,y_test):
    C = 3                                   # Số class
    reg = .1                                # Hệ số regularization
    Winit = np.random.randn(d, C)           # Khởi tạo ma trận W ngẫu nhiên
    W, loss_history = multiclass_svm_GD(X_train.T, y_train, Winit, reg)
    Z=W.T@X_test.T                          # Score matrix
    y_pred = np.argmax(Z, axis = 0)         # Tìm chỉ số có giá trị lớn nhất
    print("Print results for 30 test data points:")
    print("Predicted labels: ", y_pred)
    print("Ground truth    : ", y_test)
    print("Accuracy of multi_SVM_vec: %.2f %%" %(100*accuracy_score(y_test,y_pred)))
    cnf_matrix = confusion_matrix(y_test, y_pred)
    print('Confusion matrix:')
    print(cnf_matrix)
    print('Accuracy:', 100*(np.diagonal(cnf_matrix).sum()/cnf_matrix.sum()),'%')
    normalized_confusion_matrix = cnf_matrix/cnf_matrix.sum(axis = 1, keepdims = True)
    print('\nConfusion matrix (with normalization:)')
    print(normalized_confusion_matrix)
    print('Precision Recall and F1-Score')
    print(metrics.classification_report(y_test, y_pred, digits=3))
    # Thư viện scikit-learn
    svc = svm.SVC(kernel='linear').fit(X_train, y_train)
    lin_svc = svm.LinearSVC(C=C).fit(X_train, y_train)
    y_pred1 = svc.predict(X_test)
    y_pred2 = lin_svc.predict(X_test)
    print("SVC with linear kernel : ", y_pred1)
    print("LinearSVC linear kernel: ", y_pred2)
    print("Accuracy of multi_SVM_vec: %.2f %%" %(100*accuracy_score(y_test,y_pred1)))
    print("Accuracy of multi_SVM_vec: %.2f %%" %(100*accuracy_score(y_test,y_pred2)))
```

- Nhận xét: kết quả thu được sau khi chạy code 30 điểm dữ liệu từ tập test.
  + Độ chính xác là 80% với các giá trị điểm F1-Score ở mức khá, tuy nhiên kết quả này không ổn định sau mỗi lần chạy chương trình dao động từ 50-80%
  + Nếu sử dụng thư viện có sẵn của scikit-learn thì cho kết quả rất tốt >80%

```
Print results for 30 test data points:
Predicted labels:  [2 2 2 1 0 0 0 2 1 1 2 1 0 2 0 0 0 1 0 0 1 0 0 2 1 1 2 2 2 0]
Ground truth    :  [2 1 1 1 0 0 0 2 1 2 2 2 0 2 0 0 0 1 0 0 1 0 0 2 1 2 1 2 2 0]
Accuracy of multi_SVM_vec: 80.00 %
Confusion matrix:
[[12.  0.  0.]
 [ 0.  5.  3.]
 [ 0.  3.  7.]]
Accuracy: 80.0 %

Confusion matrix (with normalization:)
[[1.    0.    0.   ]
 [0.    0.625 0.375]
 [0.    0.3   0.7  ]]
Precision Recall and F1-Score
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      0.625     0.625     0.625         8
           2      0.700     0.700     0.700        10

    accuracy                          0.800        30
   macro avg      0.775     0.775     0.775        30
weighted avg      0.800     0.800     0.800        30

SVC with linear kernel :  [2 1 1 1 0 0 0 2 1 2 2 2 0 2 0 0 0 1 0 0 1 0 0 2 1 2 1 2 2 0]
LinearSVC linear kernel:  [2 1 1 1 0 0 0 2 1 2 2 2 0 2 0 0 0 1 0 0 1 0 0 2 1 2 1 2 2 0]
Accuracy of multi_SVM_vec_svm.SVC: 100.00 %
Accuracy of multi_SVM_vec_svm.LinearSVC: 100.00 %
```

Vẫn dùng Model trên nhưng ta chỉ phân loại dựa trên 2 đặc tính chiều dài và chiều rộng của đài hoa, do thấy dữ liệu của 2 thuộc tính này có sự phân biệt khá rõ ràng.

Tham khảo code có chỉnh sửa trong tài liệu [2]

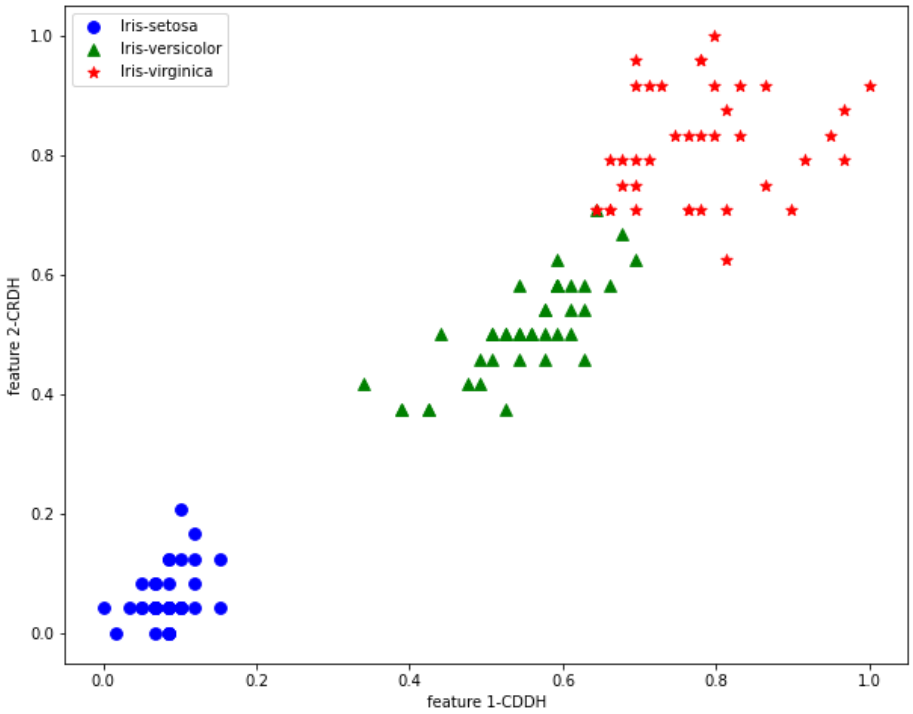
5.1,3.5,**1.4,0.2**,Iris-setosa
4.9,3.0,**1.4,0.2**,Iris-setosa
4.7,3.2,**1.3,0.2**,Iris-setosa
4.6,3.1,**1.5,0.2**,Iris-setosa
7.0,3.2,**4.7,1.4**,Iris-versicolor
6.4,3.2,**4.5,1.5**,Iris-versicolor
6.5,2.8,**4.6,1.5**,Iris-versicolor
5.7,2.8,**4.5,1.3**,Iris-versicolor
6.3,3.3,**6.0,2.5**,Iris-virginica
5.8,2.7,**5.1,1.9**,Iris-virginica
6.8,3.0,**5.5,2.1**,Iris-virginica
5.7,2.5,**5.0,2.0**,Iris-virginica

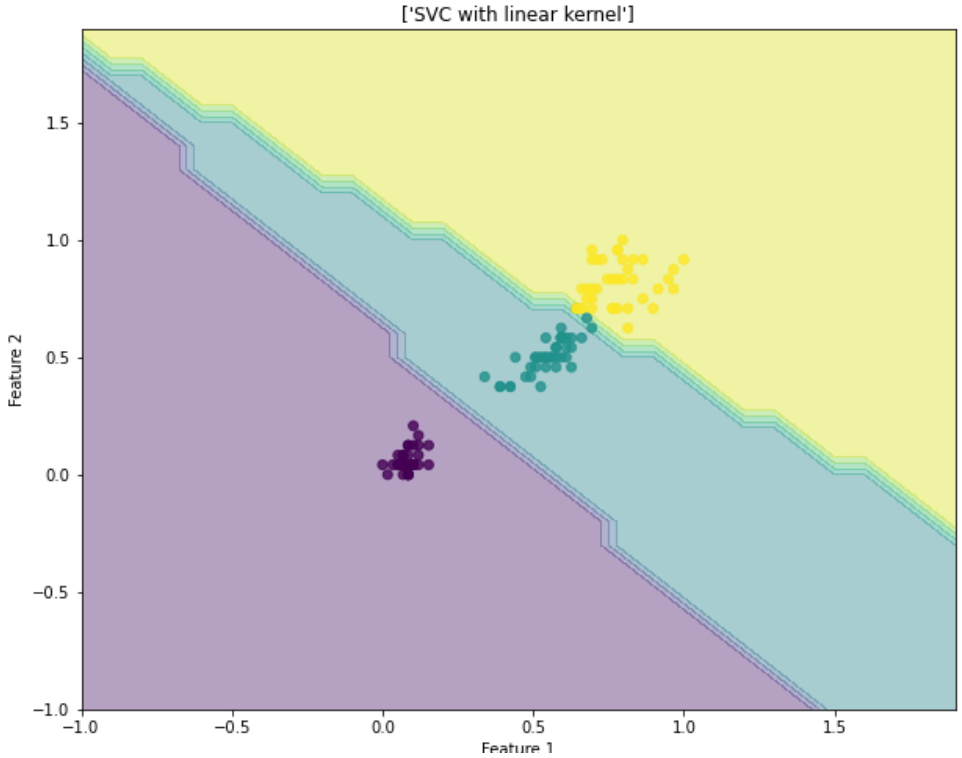
Tách 2 feature cuối ra để train, tạo figure và trực quan hóa bằng mathplotlib

```
# Dùng 2 feature 3 và 4 để train
def multi_SVM_2(X_train,y_train):
    X = X_train[:,2:4]              # Chọn 2 feature cuối của X để train
    plt.figure(figsize=(10, 8))     # Tạo một figure có kích thước 10,8
    # Tạo biểu đồ phân tán với 3 màu và 3 ký hiệu khác nhau
    for i, c, s in (zip(range(C), ['b', 'g', 'r'], ['o', '^', '*'])):
        ix = y_train == i
        plt.scatter(X[:, 0][ix], X[:, 1][ix], color=c, marker=s, s=60,
label=labels_name[i])
    plt.legend(loc=2, scatterpoints=1)          # Tạo legend (nhãn)
    plt.xlabel("feature 1-" + iris_cols[2])     # Tạo label cho trục x
    plt.ylabel("feature 2-" + iris_cols[3])     # Tạo label cho trục y
    plt.show()                                  # Show tất cả
```

Train và hiển thị dữ liệu

```
Predicted labels:  [1 1 0 0 1 0 2 0 0 0 1 2 0 2 2 1 1 2 2 1 1 1 0 1 1 1 0 1 0 2]
Ground truth    :  [2 1 0 0 2 0 2 0 0 0 1 2 0 2 2 2 1 2 2 1 1 1 0 1 1 2 0 1 0 2]
Accuracy of multi_SVM_vec: 86.67 %
```





4. *Model Gaussian Naive Bayes*
   - Thuật toán: Mô hình này được sử dụng chủ yếu trong loại dữ liệu mà các thành phần là các biến liên tục.
   - Cách làm: sử dụng thư viện scikit-learn có sẵn để train.
   - Đánh giá kết quả bằng Accuracy, Confusion matrix và F1 – Score.

```
# Bayes
def Bayes_Gauss(X_train,y_train):
    gnb = GaussianNB()                                    # Thư viện Gauss
    y_pred = gnb.fit(X_train, y_train).predict(X_test)  # Dự đoán tập test
    print("Print results for 30 test data points:")
    print("Predicted labels: ", y_pred)
    print("Ground truth    : ", y_test)
    print("Accuracy of Bayes_Gauss: %.2f %%" %(100*accuracy_score(y_test,y_pred)))
    cnf_matrix = confusion_matrix(y_test, y_pred)
    print('Confusion matrix:')
    print(cnf_matrix)
    print('Accuracy:', 100*(np.diagonal(cnf_matrix).sum()/cnf_matrix.sum()),'%')
    normalized_confusion_matrix = cnf_matrix/cnf_matrix.sum(axis = 1, keepdims = True)
    print('\nConfusion matrix (with normalization:)')
    print(normalized_confusion_matrix)
    print('Precision Recall and F1-Score')
    print(metrics.classification_report(y_test, y_pred, digits=3))
```

GIẢI BÀI 2

Mẫu dữ liệu được cho gồm các hình ảnh có những điểm khác nhau: kích thước (pixel), màu (big depth). Nhận thấy đa số ảnh có size lớn và có màu nên ta xử lý thô bằng cách resize lại ảnh về cùng một kích thước 110x110 pixel (kích thước trung bình) và đưa về cùng 24 big depth.

Kết quả dữ liệu sau khi xử lý dùng https://www.iloveimg.com/vi/thay-doi-kich-thuoc-anh

Tham k

Bước 1: Khai báo các thư viện cần sử dụng và tạo random

```
import numpy as np                              # Thư viện tính toán với ma trận
from sklearn import linear_model                # Thư viện linear_model
from sklearn.metrics import accuracy_score      # Hàm đánh giá độ chính xác
from sklearn import svm                         # SVM
from matplotlib.pyplot import imread            # Load ảnh
import matplotlib.pyplot as plt
from sklearn import metrics
np.random.seed(1)
```

Bước 2: Tải ảnh sau khi đã xử lý ảnh thô, tạo ma trận giảm chiều dữ liệu ngẫu nhiên

```
# Load ảnh
# BG(187L+73S=260) MZ(175L+45S=220)
pathBG = 'C:/Users/15108/Desktop/New data/Train Resized/mauBG/'
pathMZ = 'C:/Users/15108/Desktop/New data/Train Resized/mauMZ/'
train_BG = np.arange(1, 141)                # Chọn 140 ảnh đầu là tập train BG
test_BG = np.arange(141, 169)               # Chọn 28 ảnh còn lại tập test BG
train_MZ= np.arange(1, 141)                 # Chọn 140 ảnh đầu là tập train MZ
test_MZ= np.arange(141, 167)                # Chọn 26 ảnh còn lại tập test MZ
D = 110*110                                 # Kích thước sau khi resize
d = 1000                                    # Kích thước mới dùng projection
ProjectionMatrix = np.random.randn(D, d)    # Khởi tạo ma trận chiếu ngẫu nhiên
```

Viết các hàm cần thiết để đọc ảnh, chuyển ảnh thành thang xám và vector hóa ảnh, sau đó giảm chiều dữ liệu ảnh, mỗi vector đại diện cho một ảnh. Sau bước này ta bắt đầu train

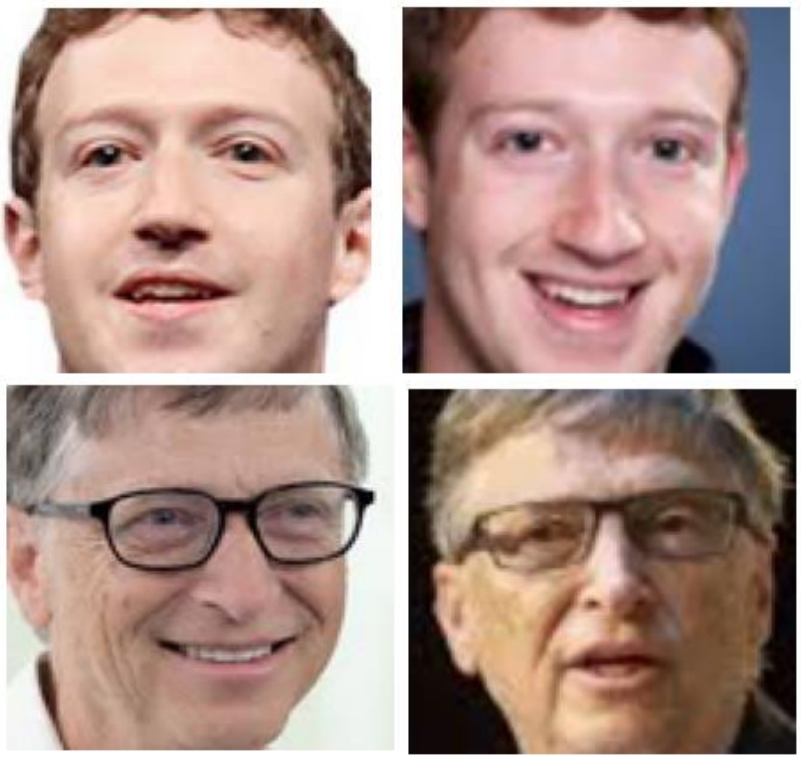
```
# Hàm đọc tên ảnh và lưu vào danh sách
def build_list_fn(pre,imgs,path):
    list_fn = []                            # Khởi tạo list rỗng
    for im in imgs:                         # Thêm ảnh vào list
        fn = path + pre  + ' '+ str(im) + '.png'
        list_fn.append(fn)
    return list_fn
# Hàm chuyển đổi ảnh màu thành ảnh xám
def rgb2gray(rgb):
    # Y' = 0.299R + 0.587G + 0.114B công thức chuyển đổi ảnh màu sang xám
    return rgb[:,:,0]*.299 + rgb[:,:,1]*.587 + rgb[:, :,2]*.114
# Hàm vetorize ảnh
def vectorize_img(filename):
    rgb = imread(filename)      # Load ảnh
    gray = rgb2gray(rgb)        # Chuyển đổi sang thang màu xám
    im_vec = gray.reshape(1, D) # Chuyển thành vector hàng
    return im_vec
# Xây dựng ma trận dữ liệu và chiếu xuống PM để giảm chiều dữ liệu
def build_data_matrix(imgs_BG,imgs_MZ):
    total_imgs = imgs_BG.shape[0] + imgs_MZ.shape[0]    # Tổng số lượng ảnh
    X_full = np.zeros((total_imgs, D))                  # Khởi tạo dữ liệu X
    y = np.hstack((np.zeros((int(total_imgs/2), )), np.ones(int((total_imgs/2), ))))
    list_fn_BG = build_list_fn('Bglri',imgs_BG,pathBG)  # Build ảnh Billgate
    list_fn_MZ = build_list_fn('Mzlri',imgs_MZ,pathMZ)  # Buil ảnh Mac Zuckerberg
    list_fn = list_fn_BG + list_fn_MZ
    # Chuyển đổi sang thang màu xám từng ảnh và giảm chiều dữ liệu
    for i in range(len(list_fn)):
        X_full[i, :] = vectorize_img(list_fn[i])
    X = X_full@ProjectionMatrix
    return (X, y)
```

Build các dữ liệu trong tập train khi đã tách và xử lý bằng các hàm ở trên, chuẩn hóa dữ liệu và trích 4 ảnh trong tập test để kiểm tra độ chính xác của thuật toán, viết hàm để show ảnh

```
# Build tập train
(X_train_full, y_train) = build_data_matrix(train_BG,train_MZ)
x_mean = X_train_full.mean(axis = 0)
x_var  = X_train_full.var(axis = 0)
def feature_extraction(X):
    return (X - x_mean)/x_var
# Chuẩn hóa dữ liệu tập train và tập test
X_train = feature_extraction(X_train_full)
X_train_full = None
(X_test_full, y_test) = build_data_matrix(test_BG, test_MZ)
X_test = feature_extraction(X_test_full)
X_test_full = None
# 4 ảnh để test
fn1 = pathMZ + 'Mzlri 158.png'
fn2 = pathMZ + 'Mzlri 160.png'
fn3 = pathBG + 'Bglri 155.png'
fn4 = pathBG + 'Bglri 153.png'
# Hàm xử lý dữ liệu đầu vào (chuyển thành ảnh xám và giảm chiều dữ liệu)
def feature_extraction_fn(fn):
    im = vectorize_img(fn)
    im1 = np.dot(im, ProjectionMatrix)
    return feature_extraction(im1)
# Hàm trực quan
def display_result_SVM(fn):
    rgb = imread(fn)          # Đọc ảnh
    plt.axis('off')           # Tắt các trục tung hoành
    plt.imshow(rgb)           # Show ảnh
    plt.show()                # Show tất cả
```

```python
# Train với SVM
def BG_MZ_SVM(X_train,y_train):
    y1 = y_train.reshape((X_train.shape[0],))     # Reshape y_train đúng chiều
    X1 = X_train                                  # Tập dữ liệu train
    clf = svm.SVC(kernel='linear')                # Dùng thư viện SVM linear
    clf.fit(X1, y1)
    # Đánh giá bằng accuracy 30 ảnh trong tập test
    y_pred = clf.predict(X_test)
    print("Accuracy:",100*metrics.accuracy_score(y_test, y_pred),'%')
    # Load 4 ảnh từ tập test để thử
    x1 = feature_extraction_fn(fn1)
    p1 = clf.predict(x1)
    if p1 == [0]:
        print('BILL GATE')
    else:
        print('MARK ZUCKERBERG')
    x2 = feature_extraction_fn(fn2)
    p2 = clf.predict(x2)
    if p2 == [0]:
        print('BILL GATE')
    else:
        print('MARK ZUCKERBERG')
    x3 = feature_extraction_fn(fn3)
    p3 = clf.predict(x3)
    if p3 == [0]:
        print('BILL GATE')
    else:
        print('MARK ZUCKERBERG')
    x4 = feature_extraction_fn(fn4)
    p4 = clf.predict(x4)
    if p4 == [0]:
        print('BILL GATE')
    else:
        print('MARK ZUCKERBERG')
    # Trực quan hóa ảnh test
    display_result_SVM(fn1)
    display_result_SVM(fn2)
    display_result_SVM(fn3)
    display_result_SVM(fn4)
```

4 ảnh từ tập test được lấy ra để kiểm tra độ chính xác



- Nhận xét: Kết quả theo thứ tự của 4 ảnh là chính xác, thuật toán có độ chính xác 88,89% theo Accuracy

```
Accuracy: 88.88888888888889 %
MARK ZUCKERBERG
MARK ZUCKERBERG
BILL GATE
BILL GATE
```

TÀI LIỆU THAM KHẢO

[1] [Online]. Available: https://machinelearningcoban.com

[2] [Online]. Available: https://viblo.asia/p/ung-dung-support-vector-machine-trong-bai-toan-phan-loai-hoa-PdbGnLXBkyA