MACHINE LEARNING: BÀI THỰC HÀNH SỐ 3.

K-NN & MULTINOMIAL LOGISTIC REGRESSION

A. PHƯƠNG PHÁP K-NN CHO PHÂN LOẠI NHIỀU LỚP

Tương tự trường hợp áp dụng phương pháp K-NN cho bài toán phân loại/hồi quy, ta phân tích các bước để áp dụng K-NN cho bài toán phân loại nhiều loại như sau:

(i) Với mỗi **x** – unseen ở đầu vào (các phần tử trong tập Validation)

(ii) Tính khoảng cách $\mathbf{d_i}$ từ **x** đến các $\mathbf{x_i}$ trong tập Training => Lập thành 1 mảng.

(iii) Sắp xếp các khoảng cách này theo thứ tự tăng dần, nhưng cần có tham chiếu để biết chỉ số phần tử ban đầu trong mảng $\mathbf{d_i}$

(iv) Tìm ra K chỉ số của các phần tử $\mathbf{x_i}$ trong tập Training ứng với các $\mathbf{d_i}$ nhỏ nhất. Gọi tập này là n_j , j =1, 2, … K.

(v) Xác định nhãn L có tần suất xuất hiện nhiều nhất trong tập đầu ra của các mẫu huấn luyện y_{nj}. Đầu ra dự đoán y_pred := L.

Các bước từ (i) đến (iv) đều đã có trong các phần áp dụng K-NN cho bài toán Hồi quy/Phân loại nhị phân. Ta sử dụng lại các phương thức tính khoảng cách và phương thức tìm K mẫu dữ liệu gần **x** nhất đã có:

Hàm tính khoảng cách:

Trường hợp d = 1 (dữ liệu 01 chiều – là biến đơn)

```
def distance(array, value):
    array = np.array(array)
    return np.absolute(array - value)
```

Trường hợp d > 1:

```
def distance(array, value):
    array = np.array(array)
    return np.linalg.norm(array - value, ord = 2, axis=<dim_axis>)
```

Ở đây <dim_axis> sẽ là chiều phù hợp, cần kiểm tra qua việc xét kích thước mỗi chiều (shape) như bài trước.

Phương thức để tìm K phần tử gần nhất với **x** chúng ta vẫn dùng: **indexes = np.argsort(array_D)[:k]** trong đó array_D là mảng khoảng cách từ **x** đến các mẫu dữ liệu trong tập huấn luyện (bước (ii) ở trên).

Ở đây chúng ta chỉ cần bổ sung phương thức tìm phần tử (nhãn) có tần suất xuất hiện nhiều nhất trong tập $\{y_{nj}\}$. Phương thức này có thể xây dựng như sau:

```
def highest_rank(arr):
    count_num = {}
    for i in arr:
        count_num[i] = arr.count(i)
    return max(count_num, key=count_num.get)
```

Chú ý:

- Trong phương thức này đầu vào arr phải là list
- Giá trị trả về là mẫu có tần suất xuất hiện cao nhất. Ví dụ đầu vào Arr = [1, 2, 4, 2, 5, 2, 7, 9] thì giá trị trả về là 2.

Hãy sử dụng các phần code đã có trong bài về phương pháp K-NN trước để kết hợp thành chương trình đầy đủ và thực hiện các công việc dưới đây.

Ví dụ A.1. Chúng ta sẽ bắt đầu với một tập dữ liệu tự tạo: Dữ liệu đầu vào trong không gian **d = 2** chiều (các điểm trên mặt phẳng tọa độ); gồm có **N = 1500** điểm được chia đều vào **C = 03** lớp là **c = 0**; **c =1** và c = 2 (mỗi lớp có 500 điểm) bằng cách tạo ngẫu nhiên các điểm với kỳ vọng (tâm) mỗi cụm điểm (lớp) có tọa độ

sau $C^*_1 = (2, 2)$; $C^*_2 = (8, 3)$ và $C^*_3 = (3, 6)$. Ma trận hiệp phương sai chung cho toàn bộ tập điểm là cov = [[1, 0], [0, 1]].

Đoạn lệnh dưới đây tạo dữ liệu đầu vào X, gán nhãn Y tương ứng cho 1500 mẫu:

```
import numpy as np
import matplotlib.pyplot as plt
# randomly generate data
N = 500  # number of training sample
d = 2  # data dimension
C = 3 # number of classes (for c = 0, 1, 2)
means = [[2, 2], [8, 3], [3, 6]] # centeroid of each class dataset
cov = [[1, 0], [0, 1]]
# generate 3 classes of datasets
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X2 = np.random.multivariate_normal(means[2], cov, N)
# each column is a datapoint (arrange 3 vectors X1 X2 X3 then transpose matrix)
X = np.concatenate((X0, X1, X2), axis = 0).T
# Generate label for data points of each class (0, 1, 2)
# with first 500 will belong to class 0, second 500 => 1 and last 500=> class 2
original_label = np.asarray([0]*N + [1]*N + [2]*N).T
```

Đoạn lệnh tiếp theo sẽ hiển thị trực quan các điểm dữ liệu để ta có thể quan sát phân bố của chúng:

```
def display(X, label):
    X0 = X[:, label == 0]
    X1 = X[:, label == 1]
    X2 = X[:, label == 2]
    plt.plot(X0[0, :], X0[1,:], 'b^', markersize = 4, alpha = .8)
    plt.plot(X1[0, :], X1[1,:], 'go', markersize = 4, alpha = .8)
    plt.plot(X2[0, :], X2[1,:], 'rs', markersize = 4, alpha = .8)
    plt.axis('off')
    plt.plot()
    plt.show()

display(X[0:, :], original_label)
```

Đoạn lệnh tiếp theo, ta lấy mỗi class 400 mẫu dữ liệu đầu làm dữ liệu training và 100 mẫu còn lại làm dữ liệu validation.

```
X_train = np.concatenate((X0[:400], X1[:400], X2[:400]), axis = 0)
```

```
Y_train = np.concatenate((original_label[:400],  original_label[500:900],
original_label[1000:1400]), axis = 0)

X_val = np.concatenate((X0[400:], X1[400:], X2[400:]), axis = 0)
Y_val = np.concatenate((original_label[400:500],  original_label[900:1000],
original_label[1400:1500]), axis = 0)

print(X_train.shape, Y_train.shape, X_val.shape, Y_val.shape)
```

Trong đoạn lệnh tiếp theo, ta sẽ đặt K = 20 và sử dụng phương pháp K-NN để dự đoán phân lớp cho toàn bộ tập dữ liệu. Tất cả các phương thức/hàm cần thiết đều đã được cung cấp ở phần trên hoặc trong bài trước. Chú ý đây là dữ liệu nhiều chiều (d = 2), do đó các bạn cần chọn hàm xác định khoảng cách (distance) phù hợp. Sau khi dự đoán, chúng ta sẽ in ra màn hình phân lớp dự đoán và phân lớp đúng tương ứng của tập validation, cũng như hiển thị chúng ra màn hình để so sánh trực quan.

```
K = 20
y_pred = np.zeros(len(Y_val)).astype(int)

for j in range(len(Y_val)):
    indexes = find_nearest_index(X_train, X_val[j], K)
    y_nearest = []
    for i in range(K):
        y_nearest.append(Y_train[indexes[i] ])

    y_pred[j] =  highest_rank(y_nearest)

print(y_pred)
print(Y_val)

display(X_val.T, Y_val)
display(X_val.T, y_pred)
```

Ví dụ A.2. Trong ví dụ này chúng ta sẽ sử dụng scikit-learn sklearn với bộ dữ liệu thực phân loại hoa IRIS theo kích thước cánh và đài hoa (https://archive.ics.uci.edu/ml/datasets/iris) .

Thông tin về dữ liệu phân loại hoa IRIS như sau:

- Số chiều d = 4 bao gồm các thuộc tính: sepal length, sepal width, petal length, petal width;
- Các trường đều là giá trị thực(tính theo cm);
- Có tổng cộng 150 điểm dữ liệu;
- Có 03 loại (lớp – classes): setosa, versicolor, virginica , mỗi phân lớp đều có 50 mẫu.

Dữ liệu được phân bố đều cho mỗi loại (mỗi phân lớp có 50 mẫu dữ liệu). Đầu tiên chúng ta sẽ load dữ liệu và thử vẽ ra các điểm dữ liệu để có hình dung trực quan về các phân lớp của dữ liệu.

Do dữ liệu của ta có 04 chiều nên không thể vẽ nguyên bản lên mặt phẳng. Vì vậy trong đoạn code dưới đây chúng ta sử dụng gói PCA (Principal Components Analysis) để phân tích và chiếu dữ liệu xuống không gian hai (02) chiều thể hiện được gần đúng nhất cấu trúc dữ liệu (thành phần chủ yếu - Principal Components).

Các bạn sẽ được học nội dung này ở phần sau, ở đây ta chỉ sử dụng nó để trực quan hóa dữ liệu.

```
import pandas as pd
from sklearn.decomposition import PCA as sklearnPCA
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
```

```
X = iris.data[:, :4]  # we take full 4 features
Y = iris.target

# Normalize data
X_norm = (X - X.min())/(X.max() - X.min())

pca = sklearnPCA(n_components=2) #2-dimensional PCA
transformed = pd.DataFrame(pca.fit_transform(X_norm))

plt.axis("off")

plt.scatter(transformed[Y==0][0], transformed[Y==0][1], s=9, label='IRIS Setosa',
c='red')
plt.scatter(transformed[Y==1][0], transformed[Y==1][1], s=9, label='IRIS Versicolor',
c='green', marker="^")
plt.scatter(transformed[Y==2][0], transformed[Y==2][1], s=9, label='IRIS Virginica',
c='blue', marker="s")

plt.legend()

plt.show()
```

Tiếp theo, hãy chia dữ liệu mỗi loại hoa thành 2 phần, 40 mẫu đầu đưa vào tập Train và 10 mẫu sau đưa vào tập Validation. Có thể tham khảo đoạn code trong Ví dụ A.1., tuy nhiên cần chú ý chọn chiều (axis) phù hợp.

Sử dụng chương trình cho phương pháp K-NN đã có và dựa vào tập Train, thực hiện dự đoán phân loại các mẫu trong tập Validation. Sử dụng các độ đo như Accuracy, Precision, Confusion Matrix để đánh giá độ chính xác của phương pháp.

B. HỒI QUY SOFTMAX (MULTINOMIAL LOGISTIC REGRESSION)

Ví dụ B.1. Tương tự các phần trước, chúng ta sẽ bắt đầu với một tập dữ liệu tự tạo giống như trong Ví dụ A.1.

Trong ví dụ này, chúng ta sẽ dùng toàn bộ tập điểm như là training data và áp dụng mô hình hồi quy Logistic nhiều lớp để xác định đường phân chia các lớp của dữ liệu.

Sau đó các bạn có thể sinh ngẫu nhiên một số điểm dữ liệu và sử dụng các tham số tối ưu đã tìm được để dữ đoán xem các điểm dữ liệu tương ứng thuộc lớp nào.

a. Chương trình Python với các hàm/phương thức tự xây dựng

- Trong ví dụ này, chúng ta sẽ tự xây dựng các phương thức chính của mô hình hồi quy Logistic nhiều lớp, chỉ dựa vào các công cụ cơ bản trong thư viện Numpy.
- Chúng ta sử dụng lại đoạn lệnh tạo dữ liệu như trong Ví dụ A.1, tuy nhiên ta cần thêm dòng lệnh bổ sung chiều x_0 (đồng nhất bằng 1) vào mỗi mẫu dữ liệu.

```
import numpy as np
import matplotlib.pyplot as plt
# randomly generate data
N = 500  # number of training sample
d = 2  # data dimension
C = 3 # number of classes (for c = 0, 1, 2)
means = [[2, 2], [8, 3], [3, 6]] # centeroid of each class dataset
cov = [[1, 0], [0, 1]]
# generate 3 classes of datasets
```

```
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X2 = np.random.multivariate_normal(means[2], cov, N)
# each column is a datapoint (arrange 3 vectors X1 X2 X3 then transpose matrix)
X = np.concatenate((X0, X1, X2), axis = 0).T
# extended data by add row ONES (equivalent to 1) at first place
X = np.concatenate((np.ones((1, 3*N)), X), axis = 0)

# Generate label for data points of each class (0, 1, 2)
# with first 500 will belong to class 0, second 500 => 1 and last 500=> class 2
original_label = np.asarray([0]*N + [1]*N + [2]*N).T
```

- Đoạn lệnh in dữ liệu ra màn hình để hình dung trực quan chúng ta sử dụng lại như của Ví dụ A.1.

- Xây dựng các phương thức cần cho mô hình hồi quy logistic nhiều lớp:
- Hàm chuyển từ một vector dữ liệu sang dạng one-hot-coding tức là các vector chỉ gồm 0/1/2 ứng với các nhãn của điểm dữ liệu:

```
from scipy import sparse
def convert_labels(y, C = C):
    """
    convert 1d label to a matrix label: each column of this
    matrix coresponding to 1 element in y. In i-th column of Y,
    only one non-zeros element located in the y[i]-th position,
    and = 1 ex: y = [0, 2, 1, 0], and 3 classes then return

            [[1, 0, 0, 1],
             [0, 0, 1, 0],
             [0, 1, 0, 0]]
    """
    Y = sparse.coo_matrix((np.ones_like(y),
        (y, np.arange(len(y)))), shape = (C, len(y))).toarray()
    return Y

# Y = convert_labels(y, C)
```

-

```
def softmax_stable(Z):
```

```
    """
    Compute softmax values for each sets of scores in Z.
    each column of Z is a set of score.
    """
    e_Z = np.exp(Z - np.max(Z, axis = 0, keepdims = True))
    A = e_Z / e_Z.sum(axis = 0)
    return A
def softmax(Z):
    """
    #Compute softmax values for each sets of scores in V.
    #each column of V is a set of score.
    """
    e_Z = np.exp(Z)
    A = e_Z / e_Z.sum(axis = 0)
    return A
def softmax_regression(X, y, W_init, eta, tol = 1e-4, max_count = 10000):
    W = [W_init]
    C = W_init.shape[1]
    Y = convert_labels(y, C)
    it = 0
    N = X.shape[1]
    d = X.shape[0]

    count = 0
    check_w_after = 20
    while count < max_count:
        # mix data
        mix_id = np.random.permutation(N)
        for i in mix_id:
            xi = X[:, i].reshape(d, 1)
            yi = Y[:, i].reshape(C, 1)
            ai = softmax(np.dot(W[-1].T, xi))
            W_new = W[-1] + eta*xi.dot((yi - ai).T)
            count += 1
            # stopping criteria
            if count%check_w_after == 0:
                if np.linalg.norm(W_new - W[-check_w_after]) < tol:
                    return W
```

```
            W.append(W_new)
    return W


# cost or loss function
def cost(X, Y, W):
    A = softmax(W.T.dot(X))
    return -np.sum(Y*np.log(A))


# Predict that X belong to which class (1..C now indexed as 0..C-1 )
def pred(W, X):
    """
    predict output of each columns of X
    Class of each x_i is determined by location of max probability
    Note that class are indexed by [0, 1, 2, ...., C-1]
    """
    A = softmax_stable(W.T.dot(X))
    return np.argmax(A, axis = 0)


# W[-1] is the solution, W is all history of weights
```

- In ra kết quả (bộ hệ số w)

```
eta = .05
d = X.shape[0]
W_init = np.random.randn(X.shape[0], C)
W = softmax_regression(X, original_label, W_init, eta)
print(W[-1])
```

- Phần code in kết quả trực quan (phần này các bạn thực hiện chỉ để quan sát)

```
#Visualize

xm = np.arange(-2, 11, 0.025)
xlen = len(xm)
ym = np.arange(-3, 10, 0.025)
ylen = len(ym)
xx, yy = np.meshgrid(xm, ym)

print(np.ones((1, xx.size)).shape)
xx1 = xx.ravel().reshape(1, xx.size)
yy1 = yy.ravel().reshape(1, yy.size)

XX = np.concatenate((np.ones((1, xx.size)), xx1, yy1), axis = 0)

print(XX.shape)

Z = pred(W[-1], XX)
```

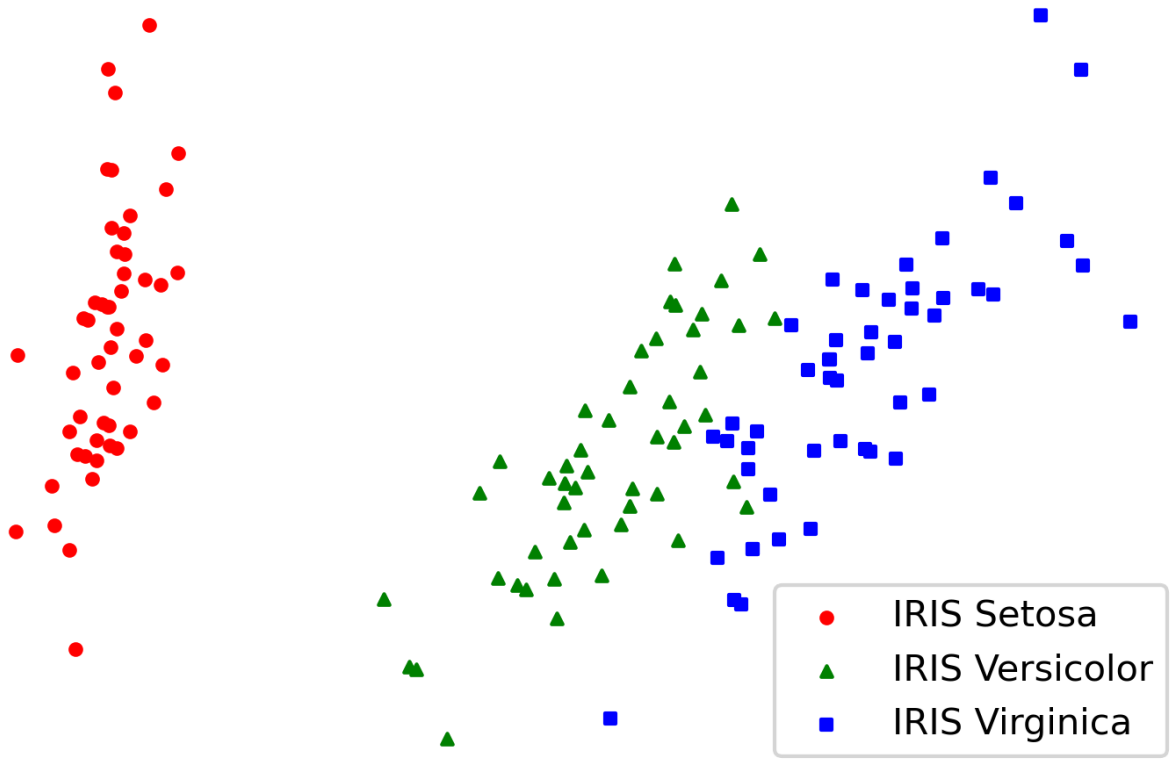
```
Z = Z.reshape(xx.shape)

CS = plt.contourf(xx, yy, Z, 200, cmap='jet', alpha = .1)

plt.xlim(-2, 11)
plt.ylim(-3, 10)
plt.xticks(())
plt.yticks(())

display(X[1:, :], original_label)
plt.savefig('ex1.png', bbox_inches='tight', dpi = 300)
plt.show()
```

Trong **Ví dụ B.1** – trên đây, chúng ta đã sử dụng các phương thức tự xây dựng (từ thư viện cơ bản numpy) để tiến hành minh họa ví dụ nhân tạo với 500 điểm dữ liệu 2 chiều thuộc 03 phân lớp khác nhau.

Dưới đây chúng ta sẽ xét một ví dụ với dữ liệu thực tế từ bộ dữ liệu phân loại hoa IRIS.

Ví dụ B.2. Trong ví dụ này chúng ta sẽ sử dụng scikit-learn sklearn với bộ dữ liệu thực phân loại hoa IRIS theo kích thước cánh và đài hoa (https://archive.ics.uci.edu/ml/datasets/iris) .

Phần đọc và biểu diễn trực quan các mẫu dữ liệu các bạn tham khảo **Ví dụ A.3**. Chương trình sẽ cho kết quả như sau:



a) <u>Bài tập tự thực hành 1</u>: Áp dụng chương trình tự xây dựng các hàm/phương thức từ numpy:

Các bạn hãy sử dụng những công cụ đã có ở **Ví dụ B.1**, sau đó tham khảo lại cấu trúc dữ liệu cvs về hoa IRIS đã có ở phần nhóm các phương pháp Naïve Bayes (Gaussian Naïve Bayes), tìm các đọc dữ liệu và có các thao tác phù hợp để sử dụng các hàm/phương thức đã xây dựng trong Ví dụ 1, thực hiện quá trình training và prediction với dữ liệu được chia theo tỉ lệ: train:validation = 4:1 (phân chia đều theo mỗi loại hoa – tự các bạn viết đoạn code để phân chia). Các bạn hoàn thành phần này như là bài tập.

b) Chương trình Python với các hàm/phương thức từ thư viện Scikit-Learn

- Trong mục này, chúng ta sử dụng mô hình xây dựng từ lớp **LogisticRegression** trong gói **sklearn.linear_model** (chú ý mô hình của chúng ta vẫn dựa trên tổ hợp tuyến tính và vẫn là hồi quy logistic).

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

import sklearn
#from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris

iris=load_iris()
# print(iris)
X=iris.data # Observed variable
Y=iris.target # Dependent variable (label)

#print(X.shape)
#print(Y.shape)

# Splitting Train and test Data
X_train,X_test,Y_train,Y_test=sklearn.model_selection.train_test_split(X,Y,test_size=0.2,
                                                                       random_state=2)
#sc=StandardScaler()
#X_train=sc.fit_transform(X_train)
#X_test=sc.transform(X_test)

# Call to Logistic Regression Model - SAG: solving is based on Stochastic Average Gradient
lorg=LogisticRegression(multi_class='multinomial',solver='sag', max_iter=5000)
# and train model by Training Dataset
lorg.fit(X_train,Y_train)

# Then Predict the Test data
Y_pred=lorg.predict(X_test)

# for accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,Y_pred))

# for confusion matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(Y_test,Y_pred)
print(cm)
```

Từ ví dụ mẫu ở trên ta có thể thấy cách sử dụng mô hình hồi quy logistic của thư viện scikit-learn trong trường hợp phân loại nhiều lớp tương tự như phân loại 2 lớp. Các bước thực hiện sẽ như sau:

i. Load dữ liệu (tùy theo việc dữ liệu dạng gì và ở đâu)

ii. Gán dữ liệu cho biến quan sát X và biến dự báo Y

iii. Chia tập training và tập test nếu cần. Chú ý trường hợp dữ liệu do chúng ta tự tạo thì cần tự gán tập dữ liệu test (không có nhãn).

iv. Tạo đối tượng **LogisticRegression** của gói **sklearn.linear_model** (đã khai báo từ thư viên scikit-learn)

v. Tiến hành "huấn luyện" trên tập dữ liệu training để tìm hệ số tối ưu thông qua phương thức fit của đối tượng nói trên:

```
<LogisticRegression object>.fit(X_train, Y_train)
```

vi. Nếu có dữ liệu test, tiến hành dự đoán cho bộ dữ liệu test với mô hình và bộ tham số đã được tối ưu theo dữ liệu huấn luyện

```
Y_predict = <LogisticRegression object>.predict(X_test)
```

vii. Hiển thị kết quả/đánh giá độ chính xác/tính confusion matrix …

Ví dụ B.3 (Bài tập tự thực hành 2). Vận dụng kiến thức từ ví dụ trên, ta xây mô hình hồi quy Logistic phân loại tập dữ liệu các đoạn văn bản ngắn lấy từ các bản tin được tổng hợp trong bộ dữ liệu có tên 20 NewsGroups. Đây là bộ dữ liệu với trên 20000 bản tin ngắn (văn bản) được phân chia gần như đồng đều trong 20 lĩnh vực thuộc 6 nhóm khác nhau như sau:

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Ví dụ, mẫu văn bản có số thứ tự 5 trong tập dữ liệu, được xếp vào phân loại chủ đề tôn giáo (talk.religion.misc) có nội dung như sau:

```
From: dmcgee@uluhe.soest.hawaii.edu (Don McGee)
Subject: Federal Hearing
Originator: dmcgee@uluhe
Organization: School of Ocean and Earth Science and Technology
Distribution: usa
Lines: 10

Fact or rumor....?  Madalyn Murray O'Hare an atheist who eliminated the use of the bible rea
ding and prayer in public schools 15 years ago is now going to appear before the FCC with a
petition to stop the reading of the Gospel on the airways of America.  And she is also campa
igning to remove Christmas programs, songs, etc from the public schools.  If it is true then
mail to Federal Communications Commission 1919 H Street Washington DC 20054 expressing your
opposition to her request.  Reference Petition number 2493.
```

Thông tin thêm về tập dữ liệu này có thể được tham khảo tại link: http://qwone.com/~jason/20Newsgroups /. Trong ví dụ này, chúng ta không sử dụng văn bản gốc mà sẽ sử dụng dữ liệu đã được xử lý/số hóa (theo kỹ thuật Bag-Of_Words) của nó, tức là dữ liệu có dạng vector như ví dụ trước.

Hơn nữa, cũng tương tự như bộ dữ liệu IRIS, bộ dữ liệu này đã được cung cấp sẵn trong gói scikit-learn, chúng ta có thể gọi và sử dụng nó bằng các lệnh import thư viện (cho dữ liệu đã vector hóa):

```
from sklearn.datasets import fetch_20newsgroups_vectorized
```

Sau đó, đoạn lệnh sau cho phép các bạn lấy tập dữ liệu và tách thành các phần training data và test data:

```
n_samples = 20000

X, y = fetch_20newsgroups_vectorized(subset='all', return_X_y=True)
X = X[:n_samples]
y = y[:n_samples]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42,
                                                    stratify=y, test_size=0.1)
train_samples, n_features = X_train.shape
n_classes = np.unique(y).shape[0]
```

Ở đây, n_samples là số mẫu dữ liệu các bạn sẽ sử dụng (chú ý tổng số mẫu là 20000). Nếu chương trình chạy chậm chúng ta có thể thu nhỏ kích thước dữ liệu (đặt lại n_samples). Trong đoạn lệnh trên tỷ lệ train:test đang là 0.9:0.1, ta có thể nâng số mẫu dữ liệu test lên bằng cách đặt test_size = 0.25. Chú ý số chiều dữ liệu khá lớn nên chúng ta chỉ nên thực hiện khoảng 5 đến 8 epochs.

Hãy hoàn thiện mô hình hồi quy Logistic nhiều lớp để dự đoán cho mẫu dữ liệu này. Sau đó in ra độ chính xác, ma trận Confusion Matrix của mô hình.

Ví dụ B.4 (Bài tập tự thực hành 3). Ta xét ví dụ phân loại các loại kính trong tập dữ liệu nhận dạng kính (Glass Identification Database) có thể download tại link: https://raw.githubusercontent.com/jbrownlee/Datasets/master/glass.csv .

Chúng ta có thể mô tả sơ bộ như sau: Dữ liệu gồm 214 mẫu kính, được đặc trưng bởi các thông số kỹ thuật/thành phần như sau:

1. RI: refractive index (chỉ số khúc xạ, tức chiết suất)
2. Na: Hàm lượng Natri (đơn vị đo: phần trăm hàm lượng theo khối lượng trong Oxide tương ứng. Chú ý kính được làm chủ yếu từ Oxide Silicat. Tương tự cho các thuộc tính từ 4 đến 10)
3. Mg: Hàm lượng Ma-gê (Magnesium)
4. Al: hàm lượng nhôm (Aluminum)
5. Si: Hàm lượng silic (Silicon)
6. K: Hàm lượng Kali (Potassium)
7. Ca: Hàm lượng Can-xi (Calcium)
8. Ba: Hàm lượng Bari (Barium)
9. Fe: Hàm lượng sắt

Mô tả chi tiết hơn có trong https://raw.githubusercontent.com/jbrownlee/Datasets/master/glass.names .

Tệp dữ liệu có 11 cột: Cột 0 ứng với chỉ số mẫu IdNumber – đánh số từ 1 đến 214; Các cột 1 đến 9 ứng với các trường thuộc tính nói trên. Cột 10 là phân loại của mẫu kính. Có 7 loại kính như sau:

- 1 building_windows_float_processed (kính cửa sổ CT xây dựng bằng thủy tinh đã xử lý nổi[(1)])
- 2 building_windows_non_float_processed (kính cửa sổ CT xây dựng bằng thủy tinh chưa xử lý nổi)
- 3 vehicle_windows_float_processed (kính cửa phương tiện bằng thủy tinh đã xử lý nổi)
- 4 vehicle_windows_non_float_processed (Không có trong bộ dữ liệu này)
- 5 containers (Thủy tinh cho các loại hộp đựng)
- 6 tableware (Kính phủ mặt bàn/tủ)
- 7 headlamps (Thủy tinh làm các loại bóng đèn có tỏa nhiệt)

Chỉ số cho các phân lớp được đánh số như trên. Các cột dữ liệu có dòng tiêu đề (header) và sau đó là số liệu.

Đoạn chương trình đọc dữ liệu từ file CSV và phân vào các tập training – test như sau:

```
from matplotlib import pyplot as plt
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# change file_data to where did you put it!
file_data = 'D:\\Teach_n_Train\\Machine Learning\\code\\exam2\\glass.csv'

glass_df = pd.read_csv(file_data)
print(glass_df.info())

glass_types = glass_df['Type'].unique()
print(glass_types)
```

```
print(glass_df['Type'].value_counts())

X_1 = glass_df[glass_df.columns[:-1]]
y_1 = glass_df['Type']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_1, y_1,
                                                    test_size=0.25,
                                                    random_state=42)
```

Các bạn tự hoàn thiện việc xây dựng mô hình hồi quy Logistic nhiều lớp để dự đoán loại kính cho tập dữ liệu test trong ví dụ này, sau đó so sánh với dữ liệu gốc để đưa ra độ chính xác, ma trận confusion matrix.

---

(1) Trong quy trình làm thủy tinh: quá trình xử lý thủy tinh nổi là việc cho một dải thủy tinh nóng chảy liên tục, được nung nóng đến hơn 1000 độ C. được đổ từ lò nung sang một bể nông lớn bằng kim loại nóng chảy, thường là thiếc. Thủy tinh nổi và nguội trên hộp thiếc và trải ra để tạo thành một bề mặt phẳng..

Trong **Ví dụ 1** chúng ta đã sử dụng các phương thức tự xây dựng (từ thư viện cơ bản numpy) để tiến hành minh họa ví dụ nhân tạo với 500 điểm dữ liệu 2 chiều thuộc 03 phân lớp khác nhau.

Trong **Ví dụ 2, Ví dụ 3** và **Ví dụ 4**, bài tự thực hành, chúng ta đã sử dụng thư viện scikit-learn, với mô hình LogisticRegression trong gói linear_ model để thực hiện quá trình training và predict.

Trong Ví dụ tiếp theo, chúng ta sẽ sử dụng lại các phần code đã xây dựng từ numpy trong **Ví dụ 1** để áp dụng vào một bài toán phân loại thực tế: Phân loại các chữ số viết tay.

Ví dụ 4. Dữ liệu của ví dụ này lấy từ: http://yann.lecun.com/exdb/mnist/ , các tệp tin được nén nên cần đoạn chương trình giải nén. Dữ liệu được để trong 4 tệp với thông tin như dưới đây:

```
train-images-idx3-ubyte:    training set images (dữ liệu ảnh train)
train-labels-idx1-ubyte:    training set labels (dữ liệu nhãn ứng với ảnh train)
t10k-images-idx3-ubyte:     test set images (dữ liệu ảnh test)
t10k-labels-idx1-ubyte:     test set labels  (dữ liệu nhãn ứng với ảnh test)
```

Tập huấn luyện chứa 60000 mẫu và tập test chứa 10000 mẫu. Chú ý dữ liệu ảnh các chữ số viết tay ở đây được lưu liên tiếp nhau và không theo định dạng ảnh, cụ thể trong cấu trúc file như sau:

Cấu trúc file **`train-images-idx3-ubyte`** chứa dữ liệu ảnh training:

```
[thứ tự byte] [type]          [value]          [description]
     0000     32 bit integer  0x00000803(2051) magic number
     0004     32 bit integer  60000            number of images (số ảnh = 60000)
     0008     32 bit integer  28               number of rows (số dòng mỗi ảnh)
     0012     32 bit integer  28               number of columns (số cột mỗi ảnh)
     0016     unsigned byte   ??               cường độ pixel thứ nhất
     0017     unsigned byte   ??               cường độ pixel thứ hai
     ........
     xxxx     unsigned byte   ??               cường độ pixel cuối cùng
```

Cường độ Pixels được sắp xếp cạnh nhau thành dòng. Giá trị cường độ Pixel là từ 0 đến 255 (1byte) nhưng để ngược: 0 là background (trắng),và 255 là foreground (đen), tuy nhiên khi in ảnh ra màn hình thì điều này không quan trọng.

Cấu trúc file **`train-labels-idx1-ubyte`** chứa nhãn của các ảnh training:

```
[thứ tự byte] [type]          [value]          [description]
     0000     32 bit integer  0x00000801(2049) magic number (MSB first)
     0004     32 bit integer  60000            number of items (số nhãn ảnh = 60000)
     0008     unsigned byte   ??               label cho ảnh 1
     0009     unsigned byte   ??               label cho ảnh 2
```

```
.....
xxxx     unsigned byte   ??               label cho ảnh cuối
```

Ở đây nhãn cho ảnh là số nguyên từ 0 đến 9 (ứng với chữ số trong ảnh).

Cấu trúc của tệp dữ liệu ảnh test (train-images-idx3-ubyte) và nhãn ảnh test (t10k-labels-idx1-ubyte) tương tự như với dữ liệu training, số lượng ảnh là 10000.

Trong ví dụ này chúng ta sẽ sử dụng 5000 ảnh ở tập training để huấn luyện, sau đó kiểm tra kết quả với 10000 ảnh ở tập test. Toàn bộ phần các phương thức/hàm đã được xây dựng cho mô hình Multinomial Logistic Regression trong **Ví dụ 1** sẽ được sử dụng lại trong ví dụ này. Chúng ta chỉ bổ sung phần đọc dữ liệu từ các file nén, xuất ra dạng ảnh (ma trận – phục vụ việc show ảnh để kiểm tra bằng mắt thường) và chuyển sang dạng vector để tính toán.

Thông tin chung về dữ liệu:

- Số chiều: d = 28 x 28 = 784 chiều;
- Số phân lớp: C = 10 (từ 0 đến 9);
- Số mẫu dữ liệu training: N = 5000.

Bộ dữ liệu có thể download từ link trên (http://yann.lecun.com/exdb/mnist/) hoặc lấy từ tệp đính kèm theo bài thực hành.

Code minh họa trong Python:

1. Đoạn code đọc tệp dữ liệu, giải nén và chuyển các đoạn ứng với mỗi ảnh sang một ma trận số:

Khai báo thư viện và đường dẫn đến tệp dữ liệu. Cần sửa đường dẫn này đến vị trí đặt dữ liệu cụ thể.

```
import os
import numpy as np

# set names to the paths because they're too long
data_path = 'D:\\Teach_n_Train\\Machine Learning\\exam_n_practice\\handwritting'
# train path
train_images_path = os.path.join(data_path, 'train-images-idx3-ubyte.gz')
train_labels_path = os.path.join(data_path, 'train-labels-idx1-ubyte.gz')
# test path
test_images_path = os.path.join(data_path, 't10k-images-idx3-ubyte.gz')
test_labels_path = os.path.join(data_path, 't10k-labels-idx1-ubyte.gz')
```

Xây dựng phương thức đọc dữ liệu từ tệp gzip, giải nén và đưa về định dạng là một dãy ảnh (một dãy ma trận nguyên)

```
def get_mnist_data(images_path, labels_path, num_images
                   , shuffle=False, _is=True, image_size=28):
    """
        This shuffle param is active when .gz is downloaded at:
            - 'http://yann.lecun.com/exdb/mnist/'
            - This function return random num_images in 60000 or 10000
    """
    # read data
    import gzip           # to decompress gz (zip) file

    # open file training to read training data
    f_images = gzip.open(images_path,'r')

    # skip 16 first bytes because these are not data, only header infor
    f_images.read(16)

    # general: read num_images data samples if this parameter is set;
    # if not, read all (60000 training or 10000 test)
```

```
    real_num = num_images if not shuffle else (60000 if _is else 10000)

    # read all data to buf_images (28x28xreal_num)
    buf_images = f_images.read(image_size * image_size * real_num)

    # images
    images = np.frombuffer(buf_images, dtype=np.uint8).astype(np.float32)
    images = images.reshape(real_num, image_size, image_size,)

    # Read labels
    f_labels = gzip.open(labels_path,'r')
    f_labels.read(8)

    labels = np.zeros((real_num)).astype(np.int64)

    # rearrange to correspond the images and labels
    for i in range(0, real_num):
        buf_labels = f_labels.read(1)
        labels[i] = np.frombuffer(buf_labels, dtype=np.uint8).astype(np.int64)

    # shuffle to get random images data
    if shuffle is True:
        rand_id = np.random.randint(real_num, size=num_images)

        images = images[rand_id, :]
        labels = labels[rand_id,]

    # change images data to type of vector 28x28 dimentional
    images = images.reshape(num_images, image_size * image_size)
    return images, labels
```

Gọi phương thức đọc dữ liệu để kiểm tra xem đọc đúng hay không:

```
train_images, train_labels = get_mnist_data(
    train_images_path, train_labels_path, 5000, shuffle=True)

test_images, test_labels = get_mnist_data(
    test_images_path, test_labels_path, 10000, _is=False, shuffle=True)

print(train_images.shape, train_labels.shape)
print(test_images.shape, test_labels.shape)
```

Đoạn chương trình chứa các phương thức tự xây dựng cho hồi quy logistic nhiều lớp đã có từ Ví dụ 1

```
# Convert matrix to image
def get_image(image):
    return image.reshape(28, 28)

# These methods are from Vi du 1
def convert_labels(y, C):
    from scipy import sparse
    Y = sparse.coo_matrix((np.ones_like(y),
        (y, np.arange(len(y)))), shape = (C, len(y))).toarray()
    return Y

def softmax(Z):
    e_Z = np.exp(Z)
    A = e_Z / e_Z.sum(axis = 0)
    return A

def softmax_stable(Z):
    e_Z = np.exp(Z - np.max(Z, axis = 0, keepdims = True))
    A = e_Z / e_Z.sum(axis = 0)
```

```
    return A

def pred(W, X):
    A = softmax_stable(W.T.dot(X))
    return np.argmax(A, axis = 0)
```

Đoạn chương trình chứa thủ tục tính toán theo phương pháp lặp Gradient Descent ngẫu nhiên trong mô hình hồi quy logistic cho dữ liệu nhiều lớp.

```
def _softmax_regression(X, Y, theta, lambda_=0.5,
                        iterations=20, learning_rate=1e-5, batch_size=200):
    from sklearn.metrics import log_loss

    losses = []
    _theta = theta
    d, N = X.shape

    for iter_ in range(iterations):
        shuffle_index = np.random.permutation(N)
        for i in shuffle_index:
            xi = X[:, i].reshape(d, 1)
            yi = Y[:, i].reshape(10, 1)
            ai = softmax_stable(np.dot(_theta.T, xi))
            _theta += learning_rate * xi.dot((yi - ai).T)
            if (iter_ * N + i) % batch_size == 0:
                Y_hat = np.dot(_theta.T, X)
                losses.append(log_loss(Y, Y_hat))

        Y_hat = np.dot(_theta.T, X)
        print(f"epoch {iter_} - cost {log_loss(Y, Y_hat) / N}")

    return _theta, losses
```

In thử một vài ảnh để kiểm tra dữ liệu

```
# for display and test digit :D
import random
import matplotlib.pyplot as plt

index = random.randint(0, 1000)
print(train_labels[index], test_labels[index])


train_image = np.asarray(get_image(train_images[index])).squeeze()
test_image = np.asarray(get_image(test_images[index])).squeeze()

plt.figure()

#subplot(r,c) provide the no. of rows and columns
f, axarr = plt.subplots(1, 2)

# use the created array to output your multiple images. In this case I have
stacked 4 images vertically
axarr[0].imshow(train_image)
axarr[1].imshow(test_image)
plt.show()
```

Chuẩn bị dữ liệu huấn luyện

```
X_train = np.concatenate((np.ones((1, train_images.shape[0])), train_images.T),
                                                                axis = 0)
Y_train = convert_labels(train_labels, 10)
```

```
print(X_train.shape)
print(Y_train.shape)

train_image = np.asarray(get_image(train_images[index])).squeeze()
test_image = np.asarray(get_image(test_images[index])).squeeze()

plt.figure()

#subplot(r,c) provide the no. of rows and columns
f, axarr = plt.subplots(1, 2)

# use the created array to output your multiple images. In this case I have
stacked 4 images vertically
axarr[0].imshow(train_image)
axarr[1].imshow(test_image)
plt.show()
```

Chạy quá trình training và in ra loss function

```
theta = np.zeros((X_train.shape[0], 10))

opt_theta, losses = _softmax_regression(X_train, Y_train, theta)
print('training success: ', opt_theta.shape, len(losses))
```

Tính các thông số về độ chính xác (gọi thư viện để chương trình không quá dài)

```
from sklearn.metrics import accuracy_score
print('accuracy training data: ', accuracy_score(train_labels,
                                        pred(opt_theta, X_train)))

# test_images, test_labels = get_mnist_data(
#     test_images_path, test_labels_path, 1000, _is=False, shuffle=False)

X_test = np.concatenate((np.ones((1, test_images.shape[0])), test_images.T),
                                                              axis = 0)
print(X_test.shape)
print('accuracy testing data: ', accuracy_score(test_labels,
                                          pred(opt_theta, X_test)))
```

Yêu cầu thực hành (Bài tập tự thực hành 4):

1) Sử dụng đoạn code chọn số chiều chính trong ví dụ 2, đưa tập dữ liệu đã đọc về còn 2 chiều, sau đó hiển thị lên màn hình để xem quan hệ giữa các lớp dữ liệu.
2) Với đoạn chương trình đọc dữ liệu đã có, hãy chạy lại ví dụ này với các thư viện của gói linear_model, lớp LogisticRegression và so sánh kết quả.