Hossein Tavakolian 40062353 HW04
Fuzzy sets and systems 1400
Shiraz University

همانند پروژه قبل در بخش اول کتابخانه ها فراخوانی می شوند که عبارتند از:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import confusion_matrix
```

توابع مورد استفاده نیز به شرح زیر است:

```
def normalize(X):
```

نرمال کردن مقادیر جهت اینکه در یک رنج باشند. رنج ۰ و ۱

این تابع بر اساس عملیات composition کار classification را برای ما انجام میدهد.

```
def Composition_based_Classification(x_tst, R0, Rc):
```

بدنه اصلی برنامه:

```
# Load data
```

خواندن فایل CSV

```
Raw_data = pd.read_csv(r'hcvdat0.csv')
Temp = pd.DataFrame(Raw_data).to_numpy()
```

```
# Remove Nans
```

حذف مقادیر Nans با استفاده از کتابخانه pandas و دستور isnull سطرهایی که در آن مقدار Nan بوده کل سطر را حذف کردم.

```
# Convert logical values to numerical
```

مانند تمرین قبلی کلاسها را به صورت عددی تبدیل کردیم به کلاس ۰، ۱، ۲، ۳ و ۴
از ویژگیها فقط ۱۰ تا ستون را استفاده کرده این و ویژگی سن (Age) و جنسیت (Sex) را استفاده نکرده ایم.

```
# Convert logical values to numerical
X = Data[:, 4::]
Y = Data[:, 1]
Y[Y == '0=Blood Donor'] = 0
Y[Y == '0s=suspect Blood Donor'] = 4
Y[Y == '1=Hepatitis'] = 1
Y[Y == '2=Fibrosis'] = 2
Y[Y == '3=Cirrhosis'] = 3
```

```
#X[X[:,1]=='f',1] = 1
#X[X[:,1]=='m',1] = 2
```

نرمال کردن را انجام داده ایم

```
# Normalize
X_normalize, minX, maxX = normalize(X)
```

دیتا به دو قسمت تست و آموزش تقسیم شده ۰٫۳ برای تست و بقیه برای آموزش

```
# Train Test split
X_trn, x_tst, y_trn, y_tst = train_test_split(
    X_normalize, Y, test_size=0.3, random_state=42)
```

برای محاسبه مفهوم Rc که همان توان تفکیک پذیری ویژگیها برای هر کلاس است که ابتدا یک R0 حساب شده است که در واقع میانگین درون کلاسی است. یعنی اگر برای کلاس صفر ۱۰۰۰ تا نمونه داریم میانگین اینها را حساب میکنی. مثلا اگر ۱۰ تا ویژگی داریم میانگین اینها رو حساب میکنیم که میشه یک ماتریس ۱۰*۱ یعنی مثلا ماتریس ۱۰*۱۰۰۰ تبدیل میشه به۱۰*۱
پس R0 درنهایت میشود یک ماتریس ۱۰*۵ به صورت تعداد کلاس در تعداد ویژگیها

```
or i in range(np.unique(y_trn).shape[0]):
    if i == 0:
        R0 = np.mean(X_trn[y_trn == i, :], axis=0).reshape(1, -1)
    else:
        R0 = np.concatenate(
            (R0, np.mean(X_trn[y_trn == i, :], axis=0).reshape(1, -1)), axis=0)
```

بعد ۲ مقدار را حساب میکنیم

۱- interClassVariance
۲- BetweenClassVariance

interClassVariance می شود واریانس هرکدام از ویژگیها در هر کلاس چقدر است. پس می شود یک ماتریس به اندازه تعداد کلاس در تعداد ویژگیها

```
for i in range(np.unique(y_trn).shape[0]):
    if i == 0:
        InterClassVariance = np.var(
            X_trn[y_trn == i, :], axis=0).reshape(1, -1)
    else:
        InterClassVariance = np.concatenate((InterClassVariance, np.var(
            X_trn[y_trn == i, :], axis=0).reshape(1, -1)), axis=0)
```

BetweenClassVariance می شود واریانس بین کلاسها است یعنی واریانس میانگین ها را حساب میکنیم. می خواهیم ببینیم هر ویژگی بین این کلاس های مختلف چگونه تغییر میکند و توی هر کلاس چقدر تغییر میکند.

```
BetweenClassVariance = np.var(R0, axis=0).reshape(1, -1)
```

بعد یک شاخص محاسبه کردیم بنام Rc که در واقع حاصل تقسیم واریانس بین کلاسی به واریانس درون کلاسی است که هرچه واریانس بین کلاس ها بیشتر باشد و واریانس درون کلاس ها کمتر باشد یعنی آن ویژگی ما دارد تفاوت بیشتری بین این کلاس ها را دارد برای ما توضیح میدهد.

```
Temp = BetweenClassVariance/InterClassVariance
Rc = Temp/((np.sum(Temp, axis=1)).reshape(-1, 1))
```

حالا اون تابع اصلی یعنی def Composition_based_Classification
با کمک R0 که ماتریس میانگین ها است و Rc و x-tst که میخواهیم ازش خروجی بگیریم.
ماتریسهایمان را از داده آموزشی ساختیم و می خواهیم با داده تست آن را تست کنیم.

در تابع

```
def Composition_based_Classification(x_tst, R0, Rc):
    Forecast = []
    for k1 in range(x_tst.shape[0]):
        temp = (x_tst[k1, :]).reshape(1, -1)
        Dist = (np.abs(temp-R0)) / \
           (np.max((np.abs(temp-R0)), axis=0).reshape(1, -1))
        Similarity = 1-Dist
        Out = np.sum(Similarity+Rc, axis=1)
        Out = np.argmax(Out)
        Forecast.append(Out)

    return np.array(Forecast)
```

ورودی ها دریافت می شوند.
یک ماتریس Forecast خالی تعریف میکنیم.
بعد با ازای تک تک داده های تست فاصله آنها را با هرکدام از اعضای اون ماتریس R0 حساب میکنیم. مثلا اگر این ویژگی من ۱۰*۱ است میرم با ماتریس R0 که ۱۰*۵ بود که هر سطر آن مربوط می شود به میانگین آن ویژگیها در یک کلاس، فاصله ها را حساب میکنم.

بعد 1-Distance یعنی similarity را حساب می کنیم. هرچه قدر شباهت دیتا جدید ما به یکی از آن کلاس ها نزدیک تر باشه یعنی بیشتر به آن تعلق دارد. که همان مفهوم composition می شود.
در واقع این شباهت یا Similarity همان R0 ما می شود.
بعد با Rc جمع می کنیم و

ماکسیمم مقداری که آنجا وجود دارد را بدست می آوریم. در واقع برای هر ویژگی که ۱*۱۰ است اونی که
حاصل Similarity + Rc می شود یک ماتریس به اندازه تعداد کلاس در ویژگی و وقتی که Sum را حساب کنیم می شود به اندازه تعداد کلاس در یک و اونی که بیشترین مقدار را داردم می شود کلاس انتخابی ما.
به عنوان خروجی به Forecast اضافه می شود
ودر نهایت خروجی گزاش می شود.

ومقادیر precision , accuracy و Fscore، Recall نیز محاسبه می شود

```
# Evaluation
result = precision_recall_fscore_support(
    np.array(y_tst, dtype=float), Outputs)
result = np.mean(result, axis=1)
Fscore = result[2]
Precision = result[0]
Recall = result[1]
Accuracy = accuracy_score(np.array(y_tst, dtype=float), Outputs)
print("Fscore: ", Fscore)
print("Precision: ", Precision)
print("Recall: ", Recall)
print("Accuracy: ", Accuracy)
```

```
Fscore:  0.49533826775743417
Precision:  0.6145313295975547
Recall:  0.5389330389992641
Accuracy:  0.8870056497175142
```

و ماتریس کامپوزیشن کشیده می شود و گزارش می شود.

```
# confusion chart
cm = confusion_matrix(np.array(y_tst, dtype=float), Outputs)
print("confusion matrix: ")
print(cm)
```

```
confusion matrix:
[[146   2   3   0   0]
 [  3   0   5   0   0]
 [  0   1   3   0   0]
 [  0   1   1   7   0]
 [  2   0   1   1   1]]
```