Hossein Tavakolian ST.N: 40062353 Shiraz University FEB 2022

شرحی بر پروژه:

بعد از اینکه تصمیم گرفتیم با Input چگونه رفتار شدو ما یکسری قوانین داریم که این قوانین می بایستی انطباق داده بشوند با ورودیها.

خب حالا این قوانین و Rule ها از کجا می آیند؟

تعریف مجموعه های فازی به چه ترتیب صورت گرفته است. مثلا ما یک دامنه ای داریم بنام X و چطوری در آن ۳ مجموعه فازی قرار دهیم.

مثلا در این پروژه ۱۲ فیچر داریم.

فرض کنیم فیچر هموگلوبین دامنه های مختلفی دارد مثلا بین ۰ تا ۱۰۰۰ دو روش وجود داره که میشود روی دامنه مجموعه فازی تعریف کرد.

۱- بیاییم و از یک Expert بپرسیم.(مثلا کم بین چه فاصله قرار دارد.)
۲- وقتی که به Expert دسترسی نداریم از روی داده ها آنها را Cluster کنیم.

چه تعداد مجموعه فازی نیاز است و یا تا چه حد این مجموعه ها overlap داشته باشند وابسته میشود روی اشرافی که به موضوع داریم و هم خبرگی افرادی که با آنها مصاحبه کردیم و هم وابسته به داده های در اختیار باشد.

اگر از روی داده بدست بیاریم هم انتظار داریم که تعداد را تنظیم کنیم هم اگر داده ها جامع نباشد ممکن است اشکالاتی پیش بیاد. گپ هایی ایجاد بشه که برای آنها مقادیر فازی تعریف نشده باشد. این گپ ها دردسرآفرین خواهد بود. و ممکن است منطبق بر واقعیت نباشد و در نهایت طراحی وابسته به اپلیکیشن و مرجع خواهد بود.

از روی داده ها قوانین را بدست بیاوریم و چه قوانینی موجب چه اتفاقهایی خواهند شد.

میزان انطباق هرکدام از ورودیها با antecedent متناظر آن به چه ترتیب میشود حساب کرد؟

X’ به صورت singleton بوده یا nonsingleton

اگر singleton باشد در هر مجموعه خودش دنبالش میگردیم.

اگر nonsingleton باشد باید تطابق بزنیم. نهایت از هر ورودی به هر antecedent یک میزان تطابقی پیدا خواهد شد.

ابهام اندازه گیری را میتوان با یک مجموعه فازی مدل کرد. به همین دلیل از ورودی nonsingleton استفاده میکنیم.

قوانین می تواند zade rule یا TSK باشد.

در TSK دیگر خروجی ما مجموعه فازی نخواهد بود بلکه یک فرمول است که مارا به یک عدد می رساند.

آنچه که ما در تطابق Rule با Input به عنوان Firing level حاصل خواهد شد به عنوان وزنی برای value حاصل شده از Rule بکار گرفته خواهد شد.

کتابخانه های مورد استفاده در برنامه:

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

NumPyیک بستهٔ نرم‌افزاری قابل افزودن به پایتون است که کاربرد اصلی‌اش در مقاصد علمی و برای کار با اعداد است. پایتون به صورت پیش‌فرض تنها از آرایه‌ها و متغیرها برای عملیات ریاضی ساده پشتیبانی می‌کند. بستهٔ نام‌پای ویژه کار با اعداد از راه ماتریس‌ها و آرایه‌های چندبعدی طراحی شده است. از ویژگی‌های آرایه‌ها در نام‌پای این است که می‌توان اندازهٔ آن‌ها را به صورت پویا تغییر داد که این امر به افزایش سرعت برنامه‌نویسی کمک می‌کند. نام‌پای را می‌توان بستهٔ بنیادی پایتون برای محاسبات علمی دانست، این بسته افزون بر فراهم‌آوردن قابلیت کار با آرایه‌های اِن-بعدی، عملگرهای درایه به درایه و عملگرهای اصلی جبر خطی را ممکن می‌سازد.

با استفاده از آرایه‌های ان‌بعدی (ndarray) نام‌پای، می‌توان بر محدودیت‌های لیست‌های پایتون (list) که تنها با استفاده از حلقه‌های تکرار می‌توان بر روی آن‌ها کار کرد، غلبه نمود و بازدهی را بالا برد. تنها محدودیت مهم آرایه‌های اِن‌بعدی نام‌پای در مقایسه با لیست‌های پایتون در این است که باید حتماً نوع داده‌های موجود در درایه‌های آن یکسان باشند. در مقابل سرعت انجام عملیاتی که با استفاده از آرایه‌های ان‌بعدی اجرا می‌شود بیشتر است.

Pandas یک کتابخانه قدرتمند برای **تحلیل**، **«پیش‌پردازش (PreProcessing)»**و **«بصری‌سازی (Visualization)»**داده‌ها است .

Pandas دارای ابزارهای گوناگونی برای انجام عملیات ورودی/خروجی است و می‌تواند **داده‌ها را از فرمت‌های گوناگونی شامل MS Excel ، TSV، CSV**و دیگر موارد بخواند.

ماتریس درهم‌ریختگیاز ماتریس درهم‌ریختگی (Confusion matrix) برای دست یافتن به تصویری جامع‌تر در ارزیابی عملکرد مدل استفاده می‌شود. این ماتریس بصورت زیر تعریف می‌شود:

		دسته پیش‌بینی‌شده: +	دسته پیش‌بینی‌شده: -
دسته واقعی	+	**TP** True Positives	**FN** False Negatives Type II error
	-	**FP** False Positives Type I error	**TN** True Negatives

معیارهای اصلیمعیارهای زیر معمولا برای ارزیابی عملکرد مدل‌های دسته‌بندی بکار برده می‌شوند.

معیار	فرمول
صحت (Accuracy)	$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$
دقت (Precision)	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$
فراخوانی (Recall)	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
ویژگی (Specificity)	$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$
F1 score	$\frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$

:Train_test_split

جداسازی و تفکیک داده‌ها (تقسیم داده‌ها) به داده‌های آموزشی و داده‌های آزمایشی (Train-Test Split) روشی برای سنجش کیفیت عملکرد یک **الگوریتم یادگیری ماشین** به حساب می‌آید.

برای بخش اول فرآیند یادگیری ماشین، از داده‌های آموزشی (Train) استفاده می‌شود و برای پایش (Monitoring)و بعضاً قطع کردن یادگیری مدل، می‌توان از داده‌های اعتبارسنجی (Validation) استفاده کرد. برای بخش دوم این فرآیند نیز از داده‌های آزمایشی (Test) استفاده می‌شود. از بین این ۳ دسته داده، می‌توان داده‌های Validation را استفاده نکرد؛ هرچند وجود آن‌ها به تنظیم بهتر برخی از آَبَرپارامترها (Hyperparameters) کمک شایانی می‌کند.

:Matplotlib

برای درک داده‌های موجود به برخی از روش‌های بصری‌سازی نیاز خواهد داشت. تصاویر، معمولا بهتر و

گویاتر از خود داده‌ها هستند (به ویژه برای ذینفعان نهایی که ممکن است دارای تخصص‌های گوناگونی باشند و آمارهای عددی و تحلیل‌های متنی نمی‌توانند گزینه‌های خوبی برای ارائه خروجی به آن‌ها باشند)» .**مت‌پلات‌لیت(Matplotlib)** «، کتابخانه‌ای قدرتمند برای بصری‌سازی داده‌ها است که می‌توان با بهره‌گیری از آن، نمودارهای گوناگون را ترسیم کرد.

2 تا تابع داریم یکی CheckRules و یکی Tuneweight

```
def CheckRules(Input, Means):
```

```
def Tuneweights(Results, y_trn):
```

بعد از اینکه دیتا خوانده شد در خط ۵۰ و ۵۱

```
# Read data  from .csv
Initial = pd.read_csv(r'hcvdat0.csv')
data = pd.DataFrame(Initial).to_numpy()
```

لیبل ها را تخصیص دادیم.

```
# Mising value handling and handling logical features
Data = data[~pd.isnull(data).any(axis=1)]
Inputss = Data[:,4::]
Targets = Data[:,1]
Targets[Targets=='0=Blood Donor'] = 0
Targets[Targets=='0s=suspect Blood Donor'] = 4
Targets[Targets=='1=Hepatitis'] = 1
Targets[Targets=='2=Fibrosis'] = 2
Targets[Targets=='3=Cirrhosis'] = 3
```

داده ها به تست و ترین تقسیم شده اند. خط ۶۶ و ۶۷

```
# spliting train and test data
x_trn, x_tst, y_trn, y_tst = train_test_split(Inputss, Targets, test_size=0.3,
random_state=42)
```

در خط ۷۱ تا ۸۲ میانگین هر کدام از ویژگیها در هر کلاس حساب شده که این میانگین در نهایت می شود یک np 5 * 10

```
# calculating mean and standard deviation of each class
for i in range(np.unique(y_trn).shape[0]):
    if i==0:
        Means = np.mean(x_trn[y_trn==i,:],axis=0).reshape(1,-1)
        Mins = np.min(x_trn[y_trn==i,:],axis=0).reshape(1,-1)
        Maxs = np.max(x_trn[y_trn==i,:],axis=0).reshape(1,-1)
    else:
        Means = np.concatenate((Means,np.mean(np.array(x_trn[y_trn==i,:],
dtype=float), axis=0).reshape(1,-1)), axis=0)
        Mins = np.concatenate((Mins,np.min(np.array(x_trn[y_trn==i,:],
dtype=float), axis=0).reshape(1,-1)), axis=0)
        Maxs = np.concatenate((Maxs,np.max(np.array(x_trn[y_trn==i,:],
dtype=float), axis=0).reshape(1,-1)), axis=0)
```

که آن را ترسیم کرده ایم در خط ۸۶ تا ۱۰۴

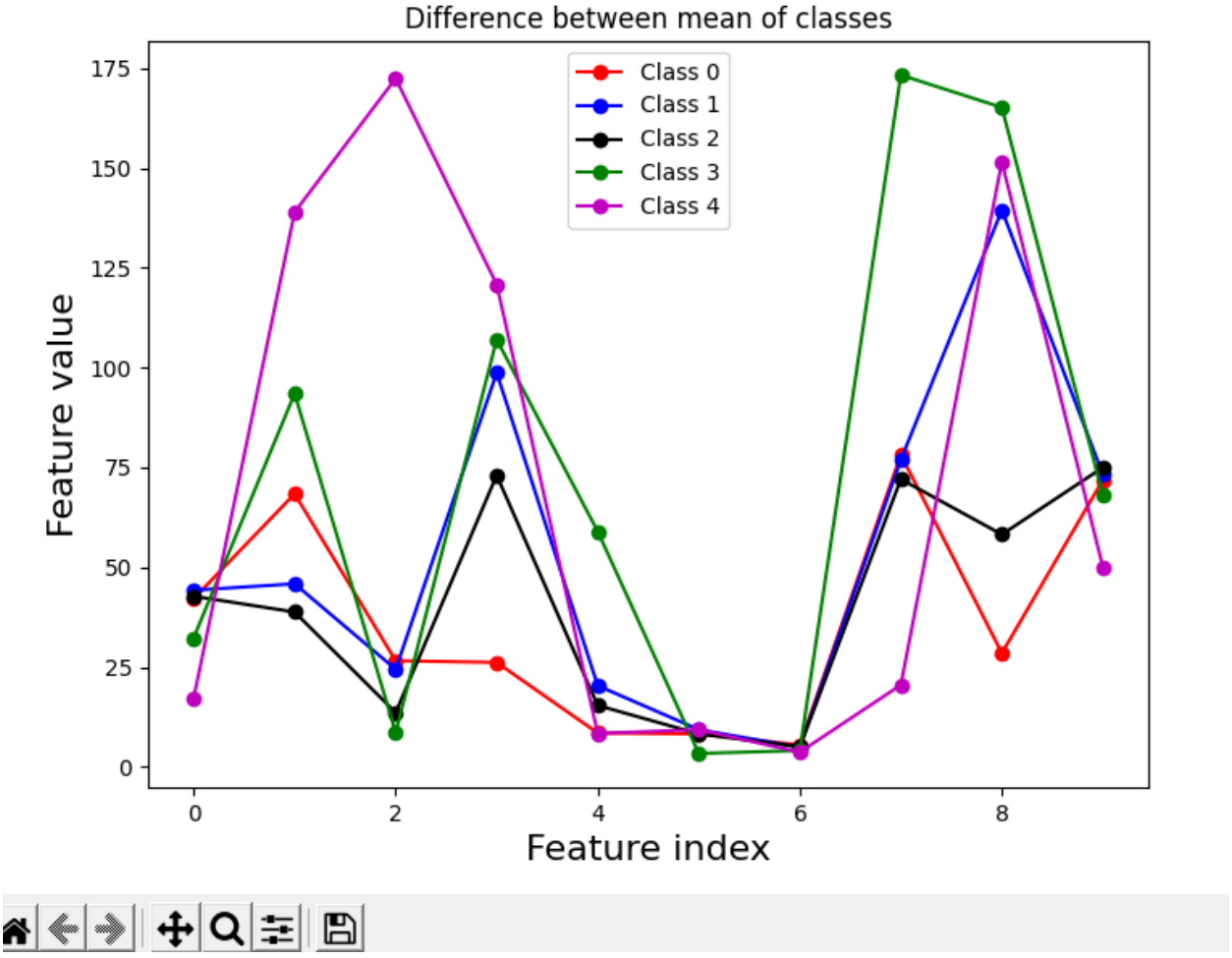
و هر کلاس با یک رنگی نمایش داده شده است و برای این رسم شده که ببنیم این با توجه به کدام فیچر می توانیم تفکیک بهتری نشان دهیم.

```
# Show differences among classes
plt.figure(figsize=[8,6])

for i in range(np.unique(y_trn).shape[0]):
    if i==0:
        plt.plot(Means[i,:], '.-r', markersize=12)
    elif i==1:
        plt.plot(Means[i,:], '.-b', markersize=12)
    elif i==2:
        plt.plot(Means[i,:], '.-k', markersize=12)
    elif i==3:
        plt.plot(Means[i,:], '.-g', markersize=12)
    elif i==4:
```

```
        plt.plot(Means[i,:], '.-m', markersize=12)

plt.xlabel('Feature index', fontsize=16)
plt.ylabel('Feature value', fontsize=16)
plt.title('Difference between mean of classes')
plt.legend(['Class 0', 'Class 1', 'Class 2', 'Class 3', 'Class 4'])
plt.show()
```



اگر شماره فیچر ها رو ۰ تا ۹ در نظر بگیریم فیچر ۱ و ۳ و ۸ خیلی بهتر تفکیک ایجاد کرده اند بین کلاسها. بخاطر همین ما بین کلاس ها این ۳ تا فیچر رو در نظر گرفتیم که اسمهای آنها هم نوشته شده است در خط –

بعد مقادیر این فیچرها رو چاپ کردیم در خط ۱۰۸ تا ۱۱۶

```
# After checking the figure, we selected Features 1,3 and 8 as discriminative
features (ALP, AST, GGT)
```

```
print("#########################################################")
print('####################### Mean of feature 1 ######################')
print(Means[:, 1])
print("#########################################################")
print('####################### Mean of feature 3 ######################')
print(Means[:, 3])
print("#########################################################")
print('####################### Mean of feature 8 ######################')
print(Means[:, 8])
```

بعد در خط ۱۱۹ تا ۱۲۶ rule ها نوشته شده است صرفا برای نمایش

Rule ها به این صورت بوده که:

اگر ورودی ما مساوی باشه با میانگین کلاس ۰ آنگاه کلاس صفر است.

اگر ورودی ما مساوی باشه با میانگین کلاس ۱ آنگاه کلاس ۱ است.

اگر ورودی ما مساوی باشه با میانگین کلاس ۲ آنگاه کلاس ۲ است.

```
# With regard to dataset and mean of features in each class:
# We define our rules based on composition between value of features in new input
and mean of features in each class.
# Then, we define 5 rules as:
# 1---> IF Input==MeanClass0 THEN class==0
# 2---> IF Input==MeanClass1 THEN class==1
# 3---> IF Input==MeanClass2 THEN class==2
# 4---> IF Input==MeanClass3 THEN class==3
# 5---> IF Input==MeanClass4 THEN class==4
```

حالا این مساوی بودن برای این است که میزان شباهت سنجیده شود، که در اینجا ما میزان شباهت را با Composition اندازه میگیریم.

یعنی در واقع هر ورودی را بیایم Compose کنیم با میانگین هر کدام از اون کلاس ها، و میزان similarity را بسنجیم در نتیجه هر ورودی وقتی وارد می شود میزان شباهتش با تک تک میانگین کلاسها محاسبه می شود.

یک نوآوری هم داشتیم که آمدیم در خط ۱۲۸ تا ۱۳۹ در ابتدا که check rules میکنیم و تابع آن را اجرا میکنیم این تابع میاد Composition را محاسبه می کند بین اون داده ای که داریم ازش نام می بریم مثلا رکورد شماره ۱ از داده Train

و خروجی یک Numpy ۵ تایی خواهد بود که در واقع میزان Composition یا شباهت اون نمونه با تک تک کلاسها را دارد به ما نشان می دهد.

```
# Determine weight for each rule in regard to training data's label
# Check rules
Results = []
for k1 in range(x_trn.shape[0]):
    currentdata = (x_trn[k1, [1, 3, 8]]).reshape(1, -1)
    Composition = CheckRules(currentdata, Means[:, [1, 3, 8]])
    Results.append(Composition)

Results = np.array(Results)
```

حالا ایده جالب این خواهد بود که یک تابع Tuneweights داریم که داده را به Test و Train تقسیم کردیم پس بیایم Composition یا میزان شباهت بین یک رکورد جدید و هر کدام از اون کلاس ها را حساب کنیم که در خط ۱۳۱ تا ۱۳۴ حساب شده.

```
for k1 in range(x_trn.shape[0]):
    currentdata = (x_trn[k1, [1, 3, 8]]).reshape(1, -1)
    Composition = CheckRules(currentdata, Means[:, [1, 3, 8]])
    Results.append(Composition)
```

میزان شباهت از نظر ۳ تا فیچری که در نظر گرفته شده و با بقیه فیچرها کاری نداریم و با تک تک کلاسها محاسبه می شود.

حالا کاری که کردیم در این تابع میزان شباهت بین دیتا جدید و کلاس ها (یعنی یک داده جدید داریم مثلا رکورد شماره ۰ از دیتا Train که یک عدد 10 * 1 است و از آن ۱۰ تا فیچر هم ۳ تا را انتخاب کردیم.)
۱، ۳ و ۸ که هرکدام یک عدد دارد که میانگین هر کدام از آن کلاسها هم برای ما یک عدد است شباهت اینها حساب میشه که خروجی میشود یک ماتریس n * 5 که شباهت این داده جدید با هر کدام از آن کلاس ها است.
حالا یک ماتریس وزن هم به اندازه همین 5 * 1 یا یک Numpy 1*5 در نظر گرفته می شود که وزن هرکدام از این ۵ تا می تواند بین ۰ تا ۱ نوسان کند.
و در Tuneweights آمدیم گفتیم ۵ تا کلاس داریم در ۵ تا حلقه به نام i1 تا i5 بیاد وزن ها را تغییر دهد از ۰ تا ۱ بافاصله۰٫۱ یعنی ترکیبات مختلف وزن ها را در نظر بگیرد بعد این ترکیبات مختلف وزن ها یعنی i1 تا i5 یعنی وزن مختص هرکدام از کلاس ها را ضرب در آن کامپوزیشن هایی کند که برای داده Train بدست آمده و سعی کند وزن ها را به صورتی Tune کند که بهترین مقدار از ACCURACY را داشته باشد.

```
def Tuneweights(Results, y_trn):
    Accuracies = []
    Weights = []
    for i1 in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
        for i2 in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
            for i3 in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
                for i4 in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
                    for i5 in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
1]:
                        Weights.append([i1, i2, i3, i4, i5])
                        temp = Results * np.array([i1, i2, i3, i4, i5])
                        Out = np.argmax(temp, axis=1)
                        accuracy = (np.where(y_trn == Out)[
                                    0].shape[0]) / (y_trn.shape[0])
                        Accuracies.append(accuracy)

    Best = np.argmax(Accuracies)
    OptimizedWeights = Weights[Best]

    return np.array(OptimizedWeights)
```

و بعد از اجرا آن Best همان بهترین ترکیب است و optimizedweight هم همان وزنهای بهینه ای خواهد بود که ما در نظر گرفته ایم که وزن های بهینه شده ۱ ، ۰٫۵، ۰، ۰٫۸ و ۰ که اینها همان وزنهایی هستند که به کلاس ها داده شده است یعنی به کلاس ۱ وزن ۱ و به کلاس ۲ وزن ۰٫۵ و ...

بعد آمدیم برای تست از ۱۴۳ تا ۱۵۱ checkrules کردیم با این تفاوت که وقتی کامپوزیشن برای هر کدام

```
# Check rules
Results = []
for k1 in range(x_tst.shape[0]):
    currentdata = (x_tst[k1, [1, 3, 8]]).reshape(1, -1)
    Composition = CheckRules(currentdata, Means[:, [1, 3, 8]])
    temp = Composition*OptimizedWeights
    Out = np.argmax(temp)
    Results.append(Out)

Results = np.array(Results)
```

از رکوردها در دیتا تست مشخص شد ما در خط ۱۴۷ آمدیم کامپوزیشن را ضربدر وزنهای بهینه میکنیم تا

```
    temp = Composition*OptimizedWeights
```

یک کامپوزیشن ریفرم شده داشته باشیم که با این کار ما Accuracy خیلی خوبی را در حدود ۸۶ بدست آوردیم.

۲ تا ایده خیلی خوب که استفاده شده یکی اینکه بر اساس اون نمودار، فیچر های خیلی خوب رو انتخاب کردیم و کار دوم هم اینکه وزنها را مناسب کردیم و کار را زیاد پیچیده نکردیم که برای هر کدام از ورودی ها بیایم اعداد فازی تعریف کنیم و Rule ها را ساده تر در نظر گرفتیم و بر حسب کامپوزیشن داده ورودی با میانگین کلاس ها گذاشتیم و یک optimize weight هدفمند هم انتخاب کردیم بر اساس TUNE رو داده Train.

نتایج:

```
############################################################
####################### Mean of feature 1 ######################
[68.40853333333337 45.86666666666667 38.7875 93.54 139.0]
############################################################
####################### Mean of feature 3 ######################
[26.15466666666667 98.75 73.0625 107.07333333333332 120.9]
############################################################
####################### Mean of feature 8 ######################
[28.49039999999999 139.50833333333335 58.225 165.23999999999998
 151.60000000000002]
############################################################
####################### Accuracy ######################
Accuracy:  0.864406779661017
############################################################
############## Confusion matrix ######################
[[149   0   0   2   0]
 [  7   1   0   0   0]
 [  1   3   0   0   0]
 [  5   1   0   3   0]
 [  2   0   0   3   0]]

[Done] exited with code=0 in 33.542 seconds
```