

به نام خدا

## Mini project2 - ML

فاطمه صفایی: ۴۰۲۰۷۹۷۴

colab link:

<https://colab.research.google.com/drive/1SHzrXtvL80wuDbpbubVzZ4MO74IPBJzu#scrollTo=g2jwxDU7YaCY>

github: <https://github.com/fatemehsafaei/ML-miniProject1>

---

### سوال ۳:

ابتدا دیتاست دارو را از سایت مورد نظر دانلود و با استفاده از دستور gdown درون colab آپلود میکنیم. دیتاست شامل ۲۰۰ نمونه با ۵ ویژگی و یک ستون تارگت به نام Drug است. در این ستون در نهایت مشخص می شود که با توجه به علائم، هر نمونه در آخر چه دارویی دریافت می کند. کد زیر مشخص می کند که مساله، یک دسته بندی ۵ کلاسه است:

```
df3.groupby('Drug').size()
```

سپس داده ها را shuffle می کنیم.

**الف:** با استفاده از train\_test\_split ۱۵ درصد از داده ها را به بخش ارزیابی و مابقی را به بخش آموزش میدهم. راه حل دیگری که برای تقسیم داده ها به این دو بخش می توان در نظر گرفت استفاده از k fold cross validation است.

```
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle = True, random_state = 74)

for train, test in kf.split(X):

    X_train_fold, y_train_fold = X[train], y[train]
    X_test_fold, y_test_fold = X[test], y[test]

    print(X_train_fold.shape, X_test_fold.shape, y_train_fold.shape,
          y_test_fold.shape)
```

با استفاده از این روش می توان هر بار بخشی از داده ها را که با دفعه قبلی متفاوت است به بخش آموزش داد. در این صورت، مدل ما تقریباً تمام دیتاست را مشاهده میکند. دست آخر برای هر fold، یک بار مدل train و پیش بینی می شود. با استفاده از این روش می توان دقت بالاتر را بدست آورد.

پیش از اینکه از مدل درخت تصمیم استفاده کنیم، لازم است که مقادیر دیتاست را encode کنیم.

```
final_df3['BP'].unique()
```

این دستور مشخص میکند که مثلاً در ستون BP، داده ها به صورت high و low و normal دسته بندی شده اند.

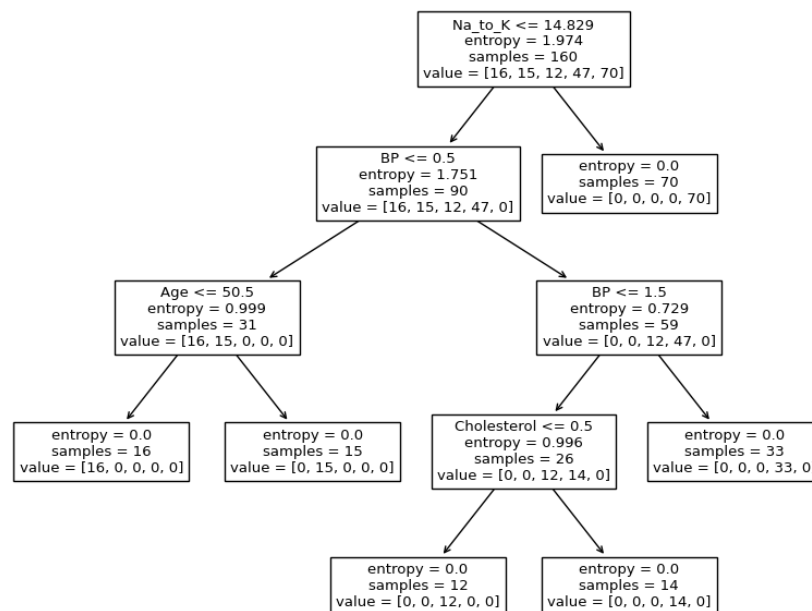
```
array(['HIGH', 'NORMAL', 'LOW'], dtype=object)
```

و مشخص است که مقادیر از نوع object هستند. بنابراین باید برنامه ای بنویسیم تا این مقادیر را به عدد تبدیل کند.

سپس باید مدل را تعریف کنیم. در این مرحله، تنها هایپر پارامتر criterion را به entropy تغییر می دهیم. یعنی بر اساس information gain بتواند node ها را قرار دهد. از امتد score استفاده کردیم و دقت برابر با ۱۰۰ درصد است. این یعنی مدل ما توانسته در عمق ۴ و با ۱۱ گره، به خوبی دیتا ها را طبقه بندی کند.

با استفاده از کدی که در مرحله نوشته شده، یک تحلیل کلی از اینکه درخت تصمیم روی دیتاست چطور طبقه بندی انجام داده است ارائه شده است. در این قسمت، با استفاده از attribute ها، مقادیری مانند تعداد گره ها، ویژگی هر گره و یا ترشولدی که بر اساس آن split در هر گره اتفاق افتاده است استخراج شده اند.

در این تحلیل، پس از آنکه مشخص شد مدل چند گره دارد، باید عمق هر گره و اینکه آیا گره است یا برگ مشخص شود. در قسمت while، هدف این است که children سمت راست و چپ هر گره شناسایی شود. یعنی شماره گره هر کدام چند است و چه عمقی دارند. تمام این اطلاعات در ماتریس stack ذخیره می شود. سپس در هر iteration، اطلاعات هر گره بر اساس attribute هایی که در ابتدا استخراج کرده بودیم نمایش داده می شود.



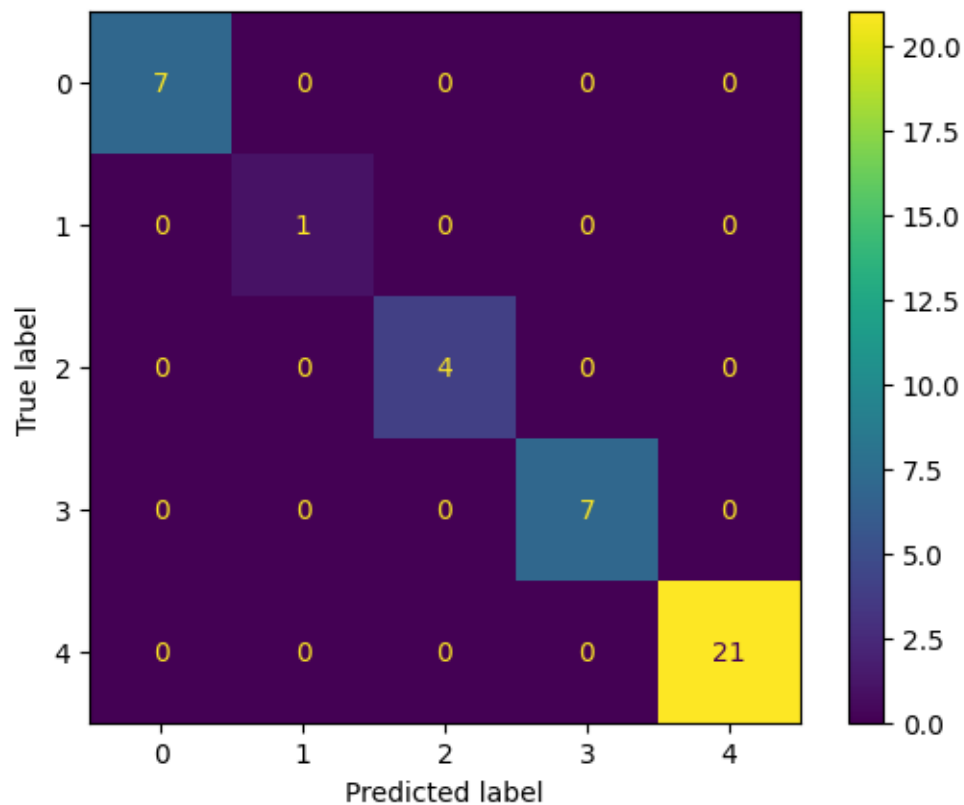
The binary tree structure has 11 nodes and has the following tree structure:

```
node=0 is a split node with value=[[16. 15. 12. 47. 70.]]: go to node 1 if X[:, 4] <= 14.828500270843506 else to node 10.  
node=1 is a split node with value=[[16. 15. 12. 47. 0.]]: go to node 2 if X[:, 2] <= 0.5 else to node 5.  
node=2 is a split node with value=[[16. 15. 0. 0. 0.]]: go to node 3 if X[:, 0] <= 50.5 else to node 4.  
node=3 is a leaf node with value=[[16. 0. 0. 0. 0.]].  
node=4 is a leaf node with value=[[0. 15. 0. 0. 0.]].  
node=5 is a split node with value=[[0. 0. 12. 47. 0.]]: go to node 6 if X[:, 2] <= 1.5 else to node 9.  
node=6 is a split node with value=[[0. 0. 12. 14. 0.]]: go to node 7 if X[:, 3] <= 0.5 else to node 8.  
node=7 is a leaf node with value=[[0. 0. 12. 0. 0.]].  
node=8 is a leaf node with value=[[0. 0. 0. 14. 0.]].  
node=9 is a leaf node with value=[[0. 0. 0. 33. 0.]].  
node=10 is a leaf node with value=[[0. 0. 0. 0. 70.]].
```

در این تحلیل داریم که هرگره، آیا یک split node است یا leaf node و دارای چه مقادیری از هر کلاس است. همچنین مشخص می شود که split بر اساس کدام ویژگی و با چه ترشولدی انجام شده است.

**پ:**

ماتریس در هم ریختگی برای مدل ما به صورت زیر است:



با توجه به این ماتریس نتیجه می شود که هیچ کدام از ابعاد به یکدیگر همبسته نیستند و با افزایش یکی، دیگری کاهش یا افزایش نمی یابد.

```
Accuracy : 1.0
Precision_micro : 1.0
Recall_micro : 1.0
F1 score_micro : 1.0
```

با ۴ شاخص ارزیابی، ماتریس در همریختگی را با داده های تست ارزیابی میکنم. مشاهده می شود که خطا وجود ندارد.

برای هرس کردن، از ۳ هاپیر پارامتر استفاده شده است. دقت هر کدام نیز محاسبه شده است. مشاهده می شود که هاپیر پارامتر max\_features تاثیر زیادی روی دقت نهایی دارد. بطوری که اگر روی ۳ تنظیم شود می تواند دقت را ۳۰ درصد کاهش دهد. پارامتر هایی مانند عمق، مشخص می کنند که عمق درخت تا چه سطحی پایین بیاید. طبیعتا اگر این عمق از عمق واقعی درخت کمتر باشد روی ارزشیابی نهایی تاثیر میگذارد چراکه باعث حذف شدن گره می شود. پارامتر ccp\_alpha نیز مشخص میکند پیچیدگی مدل تا چه حد است. این عددی بین ۰ تا ۱ است و مقدار پیش فرض آن صفر است. یعنی درخت واقعی. اما هرچه به ۱ نزدیک میشود درخت کوتاهتر و ساده تر می شود. یکی از مزایای استفاده از این هاپیر پارامتر ها، کاهش over fit شدن مدل است.

## ج:

روش هایی مانند جنگل تصادفی و adaboost یک meta-estimator هستند. این روش ها تعداد زیادی از مدل های کلاسیفایر مانند درخت تصمیم را روی زیرمجموعه های مختلف دیتاست منطبق می کنند و در نهایت با میانگین گیری، دقت پیش بینی را بالا می برند. یعنی یک نمونه را به چند کلاسیفایر میدهم و در نهایت از بین پیش بینی ها رای گیری می کنیم و بعد دیتا را به کلاسی که بیشترین رای را آورده نسبت می دهیم. الگوریتم جنگل تصادفی تنها برای کلاسیفایر درخت تصمیم است.

در این بخش از جنگل تصادفی استفاده شده است. کد این برنامه که در بخش 3-3 قابل مشاهده است مشخص میکند که استفاده از این روش با وجود اینکه عمق درخت ها را ماکزیموم ۲ گرفته (در حالت عادی ۴ است) دقتی مشابه حالتی که درخت تصمیم ما عمق ۳ داشت دارد.

#### سوال ۴:

دیتاست را به شیوه ای که در سوال قبل هم گفته شد داندلود می کنیم. سپس از `train test split` برای اسپلیت کردن دیتا به دو بخش آموزش و تست استفاده می کنیم. قبل از اینکه از بیز برای کلاسیفای کردن استفاده کنیم، لازم است که `data shuffling` انجام دهیم. این کار را پیش از اسپلیت کردن انجام می دهیم. سپس داده ها را نرمالایز می کنیم. در این عملیات نباید از داده های آزمون استفاده شود، چراکه باعث نشت اطلاعات از داده های آزمون به مدل یادگیری می شود و مدل، تعمیم پذیری و عملکرد خوبی نخواهد داشت. بنابراین ابتدا `scaler` را روی داده های آموزش فیت کرده و سپس از آن برای مقیاس بندی داده آزمون استفاده می کنیم.

دیتاست بیماری قلبی در بردارنده ۱۴ ستون و ۱۰۲۵ داده است. ستون آخر با عنوان `target` بیانگر سالم یا بیمار بودن است (سالم=۰ و بیمار=۱). ۱۳ ستون دیگر فاکتور هایی مانند سن، جنسیت (زن=۰ و مرد=۱)، نوع درد قفسه سینه، فشارخون، کلسترول، قندخون و ... را نشان می دهد. همچنین تمام داده ها عددی هستند.

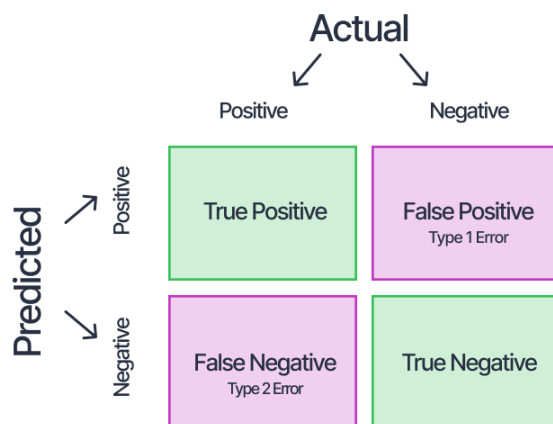
برای فهمیدن اینکه مقدار `Null` یا `NaN` داریم، از دستور `df.isna()` استفاده می شود که برای هر سلول از دیتافریم مقدار `True` (دارای مقدار خالی) یا `False` (دارای مقدار غیر خالی) را بر می گرداند. برای شناسایی تعداد مقادیر خالی در هر ستون دیتافریم از دستور زیر استفاده می کنیم:

```
df.isna().sum()
```

از مدل بیز برای کلاسیفای کردن داده ها استفاده می کنیم.

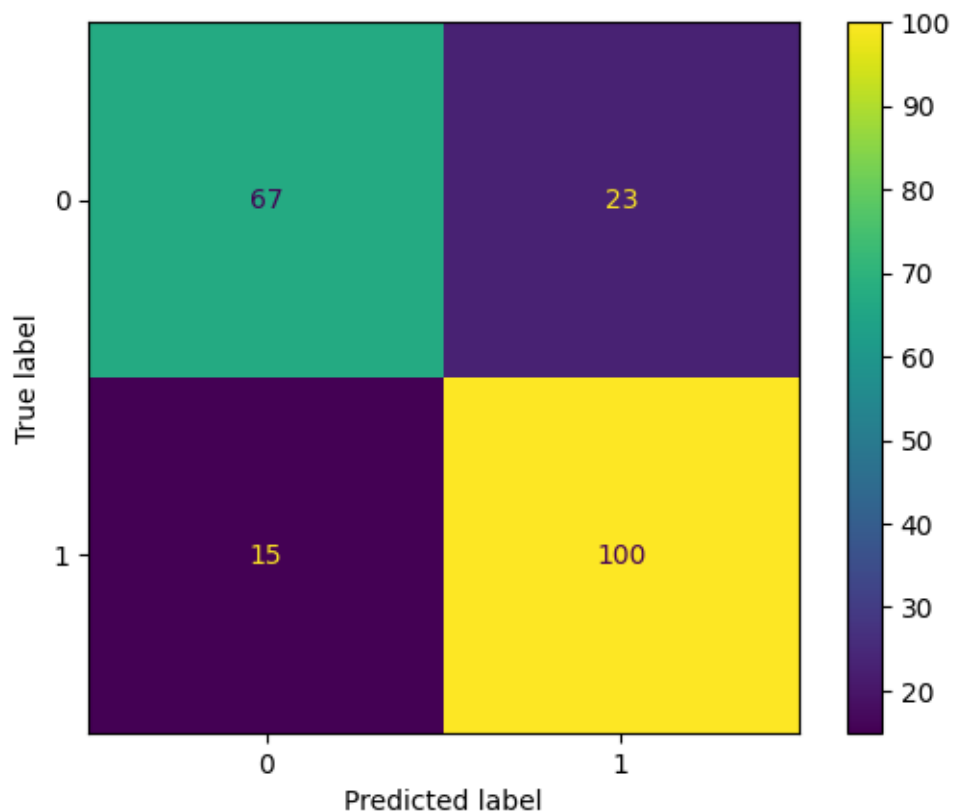
ماتریس درهم ریختگی، نتایج حاصل از تست مدل طبقه بند را بر اساس اطلاعات واقعی موجود، نمایش می دهد. بر اساس این مقادیر می توان معیارهای مختلف ارزیابی و اندازه گیری دقت را تعریف کرد.

ماتریس در هم ریختگی به صورت زیر است:



- نمونه عضو کلاس مثبت باشد و عضو همین کلاس تشخیص داده شود (مثبت صحیح یا True Positive)
  - نمونه عضو کلاس مثبت باشد و عضو کلاس منفی تشخیص داده شود (منفی کاذب یا False Negative)
  - نمونه عضو کلاس منفی باشد و عضو همین کلاس تشخیص داده شود (منفی صحیح یا True Negative)
  - نمونه عضو کلاس منفی باشد و عضو کلاس مثبت تشخیص داده شود (مثبت کاذب یا False Positive)
- برای رسم ماتریس درهم ریختگی از کتابخانه sklearn استفاده می‌شود. داده‌های واقعی و داده‌های پیش‌بینی شده را به `confusion_matrix` می‌دهیم و نمایش ماتریس را بگونه‌ای تنظیم می‌کنیم تا ماتریس برای ارزیابی پیش‌بینی `target` ترسیم شود.

ماتریس در هم‌ریختگی مدل ما بصورت زیر است:



این ماتریس نشان می‌دهد که 67 مورد به درستی و 15 مورد به غلط، سالم تشخیص داده شده است. همچنین 100 مورد به درستی و 23 مورد به غلط بیمار تشخیص داده شده است.

معیارهای قابل استخراج از ماتریس درهم ریختگی عبارتند از:

۱- معیار صحت:  $Accuracy = (TP+TN) / (TP+FN+FP+TN)$

۲- معیار دقت:  $Precision = TP / (TP+FP)$

۳- معیار حساسیت یا Recall:  $Recall = TP / (TP+FN)$

۴- معیار F1 Score:  $F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$

```
# how to evaluate a confusion maatrix:
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('Accuracy :',accuracy_score(y_test,y_pred))
print('Precision :',precision_score(y_test,y_pred))
print('Recall :',recall_score(y_test,y_pred))
print('F1 score :',f1_score(y_test,y_pred))
```

```
Accuracy : 0.8146341463414634
Precision : 0.8130081300813008
Recall : 0.8695652173913043
F1 score : 0.8403361344537815
```

با استفاده از نتایج بدست آمده، مدل معیار صحت عملکرد ۸۰ درصد دارد.

### بررسی حالت Macro و Micro در sklearn

کتابخانه sklearn در معیارهای ارزیابی نظیر معیارهای `precision_score`, `recall_score`, `f1_score`، پارامتر `average` را به منظور تعیین نوع میانگین‌گیری روی داده‌ها در نظر گرفته است.

اگر `average='micro'` به این معناست که معیار مورد نظر بطور کلی با مقادیر مثبت واقعی، منفی کاذب و مثبت کاذب محاسبه می‌شود.

اگر `average='macro'` به این معناست که معیار مورد نظر برای هر برچسب محاسبه می‌شود و میانگین بدون وزن پیدا در نظر گرفته می‌شود. در اینجا عدم تعادل برچسب‌ها نادیده گرفته می‌شود.

برای مقایسه هر دو رویکرد `micro` و `macro` معیارها را با در نظر گرفتن پارامتر `average` مجدداً محاسبه می‌کنیم:

```
# average = micro vs. average = macro:

print('Precision_micro :',precision_score(y_test,y_pred,average='micro'))
print('Precision_macro :',precision_score(y_test,y_pred,average='macro'))

print('Recall_micro :',recall_score(y_test,y_pred,average='micro'))
print('Recall_macro :',recall_score(y_test,y_pred,average='macro'))

print('F1 score_micro :',f1_score(y_test,y_pred,average='micro'))
print('F1 score_macro :',f1_score(y_test,y_pred,average='macro'))

Precision_micro : 0.8146341463414634
Precision_macro : 0.815040650406504
Recall_micro : 0.8146341463414634
Recall_macro : 0.8070048309178743
F1 score_micro : 0.8146341463414634
F1 score_macro : 0.8097029509478211
```

که مشاهده می شود در هر دو حالت نتایج نزدیک به هم هستند.

در نهایت پنج داده بصورت تصادفی از بین داده های آزمون انتخاب شده و خروجی واقعی با پیش بینی شده و قایسه شده است:

```
ytests = []
ypreds = []
for i in range(5):
    # print(i)
    num = np.random.randint(0, X_test_scaled.shape[0])
    ytests.append(y_test[num])
    ypreds.append(y_pred[num])

print(ytests, ypreds)

[1, 0, 0, 1, 1] [1, 0, 0, 1, 0]
```