بنام خدا

دانشکده‌ی مهندسی برق و کامپیوتر

یادگیری ماشین

استاد : ابولقاسمی

پروژه‌ی نهایی درس

اعضای گروه :

- عرفان حاجی هاشمی – 810197489
- کیانا لطفی – 810197574
- شایان واصف_ 810197603

: Topics

هدف از این پروژه تشخیص جنسیت و حالت چهره افراد از روی داده های تصویر و هم چنین کلاس بندی و خوشه بندی می باشد. که با کمک مفاهیم یادگیری ماشین و شناسایی تصویر به این هدف نائل می شویم.

➢ Dataset :

داده هایی که در این پروژه مورد استفاده قرار گرفته ، تصاویر2605 فرد مختلف با سه حالت چهره (happy,neutral,sad) در دو جنس زن و مرد است، که با استفاده از روش استخراج ویژگی (feature extraction)، تعدادی مجموعه ویژگی (feature set) به داده های تصویر اضافه شده ، در قالب فایل pkl. ذخیره شده است.

➢ Pre-proccessing:

در ابتدا داده ها را لود کرده و پیش پردازش (pre-proccessing) انجام می دهیم.

پیش پردازش داده ها دارای مراحل زیر می‌باشد :

✓ یکدست و تمیز کردن داده ها : در این قسمت داده های نامناسب در ستون "status" را شناسایی کرده و آن داده ها را حذف می کنیم. این داده ها شامل مقادیر garbage تصادفی می‌باشد که در هنگام ساخت Dataframe ایجاد شده است . در کد زیر این بخش عملی می‌شود :

Drop all the garbage value inside the Status column

```
In [437]:
def Final_Data(new_data):
    final_data=new_data
    for item in new_data.loc[:,"Status"]:
        if len(item)>=10:
            final_data=final_data.drop(new_data.loc[:,"Status"][new_data.loc[:,"Status"]==item].index[0])
    final_data=final_data.reset_index(drop=True)
    return final_data
```

همچنین به دلیل اینکه اسامی وارد شده در ستونهای Gender و Status متنوع می‌باشد ، (برای مثال Happy یا happy ، مجبوریم طبق فرض خودمان عبارت های داخا این ستونها را را با اسم قابل شناسایی (H,S,N برای ستون Status و female,male برای ستون Gender) جایگزین کنیم .

Gender	Status
female	H
male	S
male	N

✓ جایگزین کردن Missing value ها : قبل نرمالایز کردن داده ها ، داده ها را آماده سازی می کنیم ، یعنی به دنبال پیدا کردن Nan های داخل ستون‌های ویژگی هستیم . در ادامه سه رویکرد اساسی را پیشنهاد می‌دهیم :

1. حذف تمامی Nan های داخل Dataframe موجود : در این روش در هر سطری که Nan شناسایی کنیم ، آن سطر را دور می‌ریزیم . طبیعی است که این روش به دلیل اینکه تعداد داده‌ی زیادی را از دست می‌دهیم معقول نیست .
2. جایگزینی با میانگین :در این روش هر کدام از feature های f1 تا f5 را بصورت یک آرایه از سطر ها و ستون‌های ویژگی در می‌آوریم .این کار را توسط تابع Separate-feature انجام می‌دهیم. برای مثال آرایه بدست آمده از f1 بصورت زیر است :

```
def Seperate_feature(final_data):
    A=final_data.loc[:,'f1':'f5']
    B=[];
    for j in range(0,5):
        B.append(A.values[0][j])
        for i in range(1,len(final_data)):
            B[-1]=np.concatenate((B[-1],A.values[i][j]))
        B[-1]=B[-1].reshape((len(final_data),len(final_data.iloc[:,j+3][0])))
    return B
```

```
f_pre_normal=Seperate_feature(final_data)
```

```
f_pre_normal[0]
```

```
array([[-0.03723609, -0.02082427, -0.02524613, ..., -0.01388344,
         0.01322598, -0.00587592],
       [-0.00931587, -0.00085005,  0.00080589, ...,  0.02912709,
        -0.07292464,  0.0129727 ],
       [-0.01366237,  0.03466877, -0.06504013, ..., -0.01993099,
        -0.04300289,  0.03011747],
       ...,
       [-0.034456  , -0.00303604, -0.04484462, ..., -0.05977233,
        -0.02448294,  0.13791437],
       [-0.02918069,  0.03243812,  0.01286062, ...,  0.01659756,
        -0.04191961,  0.06275364],
       [-0.04017293, -0.03402152,  0.0642992 , ..., -0.01965886,
         0.0034796 , -0.04388807]])
```

سپس در هر کدام از ستونهای f1 ، Nan ها را شناسایی می‌کنیم و آنرا با میانگین بقیه‌ی عناصر آن ستون جایگزین می‌کنیم . تابع نوشته شده برای این قسمت بصورت زیر است :

Replacing all the missing values with the mean

```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
for i in range(0,5):
    imputer.fit(f_pre_normal[i])
    f_pre_normal[i]= imputer.transform(f_pre_normal[i])
```

3. ترکیب روش 1 و 2 : راه دیگر این است که ابتدا در هر ستون بگردیم و درصد تعداد Nan ها را به اندازه‌ی آن ستون پیدا کنیم . اگر این درصد از یک حدی بیشتر بود ، آن ستون را حذف و در غیر این صورت میانگین بقیه‌ی عناصر آن ستون را جایگزین Nan ها کنیم. این روش ترکیبی از روش 1 و 2 است . برای سه مرز 10 درصد ، 15 درصد و 35 درصد خروجی آورده شده است :

Method2(Arbitrary)(Deleting the columns of feature f1 to f5 which have the most nan and replace the other with mean)

In [529]:
```
f_pre_normal1=5*[0]
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
for j in range(0,5):
    Sel=[]
    for i in range(0,f_pre_normal[j].shape[1]):
        if np.count_nonzero(np.isnan(f_pre_normal[j][:,i]))/f_pre_normal[j].shape[0]*100 >=10 :
            Sel.append(i)
        else:
            continue
    #print(Sel)
    f_pre_normal1[j]=np.delete(f_pre_normal[j],Sel, 1)
    if(f_pre_normal1[j].shape[1]==0):
        print("All the feature was removed for f{}\n".format(j+1))
    else:
        imputer.fit(f_pre_normal1[j])
        f_pre_normal1[j]= imputer.transform(f_pre_normal1[j])
        print("{}\n".format(f_pre_normal1[j].shape))
```

Method2(Arbitrary)(Deleting the columns of feature f1 to f5 which have the most nan and replace the other with mean)

In [530]:
```
f_pre_normal1=5*[0]
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
for j in range(0,5):
    Sel=[]
    for i in range(0,f_pre_normal[j].shape[1]):
        if np.count_nonzero(np.isnan(f_pre_normal[j][:,i]))/f_pre_normal[j].shape[0]*100 >=15 :
            Sel.append(i)
        else:
            continue
    #print(Sel)
    f_pre_normal1[j]=np.delete(f_pre_normal[j],Sel, 1)
    if(f_pre_normal1[j].shape[1]==0):
        print("All the feature was removed for f{}\n".format(j+1))
    else:
        imputer.fit(f_pre_normal1[j])
        f_pre_normal1[j]= imputer.transform(f_pre_normal1[j])
        print("{}\n".format(f_pre_normal1[j].shape))
```

Method2(Arbitrary)(Deleting the columns of feature f1 to f5 which have the most nan and replace the other with mean)

In [532]:
```
f_pre_normal1=5*[0]
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
for j in range(0,5):
    Sel=[]
    for i in range(0,f_pre_normal[j].shape[1]):
        if np.count_nonzero(np.isnan(f_pre_normal[j][:,i]))/f_pre_normal[j].shape[0]*100 >=35 :
            Sel.append(i)
        else:
            continue
    #print(Sel)
    f_pre_normal1[j]=np.delete(f_pre_normal[j],Sel, 1)
    if(f_pre_normal1[j].shape[1]==0):
        print("All the feature was removed for f{}\n".format(j+1))
    else:
        imputer.fit(f_pre_normal1[j])
        f_pre_normal1[j]= imputer.transform(f_pre_normal1[j])
        print("{}\n".format(f_pre_normal1[j].shape))
```

```
(2545, 512)

(2545, 1536)

(2545, 2304)

(2545, 2560)

(2545, 204)
```

```
(2545, 512)

(2545, 1536)

(2545, 2304)

(2545, 2560)

All the feature was removed for f5
```

همانطور که در مرز 10 درصد مشاهده می‌کنید ، تمامی ستون‌های ویژگی برای f1,f2,f3,f5 حذف شده‌اند و تنها تمام ستونهای f4 باقی مانده است .

در مرز 15 درصد تمامی ستونهای f5 حذف شده ولی کلیه ستونهای f1,f2,f3,f4 دست نخورده باقی مانده‌اند. در مرز 35 درصد ، تمامی ستونهای f1 تا f5 دست نخورده‌اند و بنابراین میانگین جایگزین Nan ها شده است. استدلال بالا را می‌توان با نتیجه‌ی کد روبرو نشان داد :

Count the totall number of nans inside each feature column(arbitrary)

```
def count_nan(data):
    for j in range(0,5):
        count=0
        for i in range(0,len(data)):
            count+=np.count_nonzero(np.isnan(data.iloc[i,j+3]))
        print(count)
        percentage=count/(len(data)*len(data.iloc[:,j+3][0]))*100
        print("percentage is :{}%".format(percentage))
```

```
count_nan(data)
```

```
162816
percentage is :12.207293666026871%
417792
percentage is :10.441458733205375%
626688
percentage is :10.441458733205375%
0
percentage is :0.0%
178908
percentage is :33.66602687140115%
```

کد بالا درصد تعداد کل Nan های موجود در ستونهای ویژگی f1 تا f5 می‌دهد. طبق خروجی ها ، میزان درصد Nan در هر یک از f1,f2,f3,f5 بالای 10 درصد می‌باشد .
همچنین در ستون f4 ، Nan وجود ندارد و در ستون f5 بیشترین میزان Nan به مقدار 34 درصد وجود دارد.
*نکته‌ی مهم اینجاست که با دقت در Dataframe داده شده می‌توان فهمید که اگر در یک از ستونها Nan وجود داشته باشد ، در تمامی ستونهای متناظر با سطر آن ، Nan وجود دارد . این بدین معنی است که تصمیم گرفته شده بر مبنای حذف یا جایگزینی با میانگین برای هر ستون با ستون کناری و درنتیجه برای تمامی ستونهای ویژگی f1 تا f5 مشابه می‌باشد . یعنی مثلا اگر در ستون اول f1 تصمیم بر حذف آن ستون شد ، تمامی ستون های f1 باید حذف شوند زیرا تعداد Nan برابری با ستون اول دارند . همین تصمیم برای جایگزینی با میانگین نیز صادق است .
بنابراین در این روش یا همگی ستونها با میانگین جایگزین می‌شوند (روش 2) یا همگی حذف می‌شوند (روش 1) .
برای مثال در بالا در مرز 10 درصد ، چون همگی به جز f4 دارای درصد Nan بیش از 10 درصد هستند ، پس تمامی ستونهای آنها حذف شده و تنها f4 با میانگین جایگزین می‌شود . در مرز 15 درصد تنها f5 هست که دارای Nan بیشتر از 15 درصد هست ، پس f5 حذف شده و f1 تا f4 با میانگین جایگزین می‌شوند. در نهایت برای مرز 35 درصد ، تمامی f1 تا f5 زیر مرز داده شده قرار می‌گیرند پس همگی با میانگین جایگزین می‌شوند.

با تفاسیر بالا انتخاب روش 2 (جایگزینی تنها با میانگین) روش مطمئن تری نسبت به بقیه روش ها می‌باشد.

✓ نرمالایز کردن داده‌ها : نرمالایز کردن داده‌ها در دو بخش انجام می‌شود :

1. در همان ابتدای خواندن داده‌ها ، ستون Age را نرمالایز می‌کنیم :

```
data['Age']=data['Age']/data['Age'].abs().max()# Normalize age.
```

2. قبل از اینکه PCA روی داده‌ها اعمال کنیم ، ابتدا آرایه‌های بدست آمده از هر ویژگی را که خود دارای چندین ستون از ویژگی هاست را نرمالایز می‌کنیم :

Seperating and Normalizing each feature to an array

In [509]:
```
def Norm_feature(f_pre_normal):
    min_max_scaler = preprocessing.MinMaxScaler()
    f=[]
    for i in range(0,5):
        x_scaled = min_max_scaler.fit_transform(f_pre_normal[i])
        df = pd.DataFrame(x_scaled)
        f.append(np.array(df))
    return f[0],f[1],f[2],f[3],f[4]
```

✓ روش PCA برای کاهش بعد : از آنجاییکه با curse of dimensionality روبرو می شویم ، به کاهش بعد می پردازیم که ما از روش PCA استفاده کرده ایم.

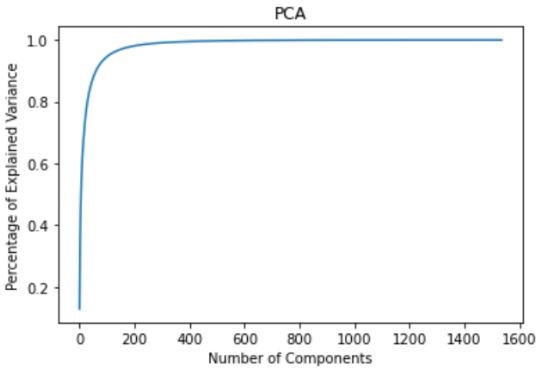
با توجه به نمودار روبرو که برای f1 رسم شده ، مشخص است که حدود 95 تا از component ها ، تقریبا همه داده ها را شامل می شود پس می توان PCA را با استفاده از 95 تا component انجام داد.

PCA

```
In [448]:
feature=[]
from sklearn.decomposition import PCA
pca = PCA(n_components = 0.95)
for i in range(0,5):
    pca.fit(f[i])
    feature.append(pca.transform(f[i]))
    print(feature[-1].shape)
```

```
(2545, 41)
(2545, 107)
(2545, 105)
(2545, 28)
(2545, 5)
```

```
pca.fit(f[1])
eigenvalues = pca.explained_variance_ratio_
plt.figure()
plt.plot(range(len(eigenvalues)), np.cumsum(eigenvalues))
plt.xlabel("Number of Components")
plt.ylabel("Percentage of Explained Variance")
plt.title("PCA")
plt.show()
```



✓ Upsampling : در این روش طبق الگوریتم Upsampeling با اضافه کردن داده‌های تکراری تعداد سطر های با لیبل مرد و زن را برابر می‌کنیم. با توجه به اینکه تعداد زن و مرد در داده ها ، imbalanced است ، نیاز به upsampling داریم (چون تعداد داده های بیشتر نتایج بهتری برایمان دارند و تعداد داده ها هم خیلی زیاد نیست ، downsampling(که تعداد داده را کم می کند) انجام نمی دهیم.)

➢ Feature set :

در این قسمت ایده ای که استفاده شده و مورد تایید دستیار آموزشی هم قرار گرفته است استفاده از correlation است ، بدین صورت که بعد از پیاده سازی PCA در هر مجموعه ویژگی correlation هر ستون از ویژگی ها را با یکدیگر و با label مورد نظر بررسی کرده ایم و در هرکدام از f1 تا f5 تعداد ویژگی هایی که دارای correlation مثبت هستند را بدست می آوریم و سپس بر اساس بیشترین تعداد این مورد

بهترین ویژگی را انتخاب میکنیم دلیل انتخاب correlation های مثبت این است که ستون هایی با این ویژگی دارای جهت تغییرات یکسان با label هستند که این خودش یک فاکتور برتری برای آن ستون است

برای پیاده سازی این روش در مورد label های numeric مثل age که مشکلی نداریم و همچنین به دلیل binary بودن label مربوط به Gender ، برای این label هم بدون مشکل correlation ها را بدست می آوریم که نتایج Ranking feature ها طبق روش بالا بصورت روبرو است :

Ranking f1 to f5 by finding the correlation between age & Gender lebels and feature [f1:f5]

```
labels = ['Gender' , 'Age']
column=['f1','f2','f3','f4','f5']
for label in labels:
    sum_more_than = {'f1':0 , 'f2':0 , 'f3':0 , 'f4':0 , 'f5':0}
    for i in range(5):
        df = pd.DataFrame(f_final[i])
        df=pd.concat([df,df_final[label]],axis=1,join='inner')
        corrmat = df.corr()
        label_row = corrmat[label]
        grater = label_row > 0
        sum_more_than[column[i]] = grater.sum()
    print(label , "ranking features ->")
    print(sorted(sum_more_than.items(), key=lambda x: x[1], reverse=True))
```

```
Gender ranking features ->
[('f2', 51), ('f3', 46), ('f1', 21), ('f4', 14), ('f5', 4)]
Age ranking features ->
[('f2', 63), ('f3', 53), ('f1', 20), ('f4', 19), ('f5', 4)]
```

Convert the Gender column to binary format and Status column to one hot

In [29]:
```
df_final['Gender'] =df_final['Gender'].astype('category').cat.codes
status_column = df_final.iloc[:, 2:3].values
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
status_column = np.array(ct.fit_transform(status_column))
status_df = pd.DataFrame(status_column)
status_df.columns = ['H' , 'N' , 'S']
#df_final
status_df
```

Out[29]:

	H	N	S
0	1.0	0.0	0.0
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	1.0	0.0	0.0
4	0.0	1.0	0.0
...	...	...	...
2540	0.0	1.0	0.0
2541	1.0	0.0	0.0
2542	1.0	0.0	0.0
2543	1.0	0.0	0.0
2544	1.0	0.0	0.0

ولی چون correlation برای داده های non_numeric قابل تعریف نیست ، ستون مربوط به status را به کمک روش one hot به سه ستون گسترش می‌دهیم که در این صورت باز هم label ها مانند Gender حالت binary پیدا کرده و می توان برای آن ها correlation بدست آورد . خروجی one hot برای ستون Status بصورت روبرو می‌باشد

Ranking f1 to f5 by finding the correlation between Status lebel and feature [f1:f5]

```
sum_more_than = {'f1':0 , 'f2':0 , 'f3':0 , 'f4':0 , 'f5':0}
for status in status_df.columns:
    for i in range(5):
        df = pd.DataFrame(f_final[i])
        df=pd.concat([df,status_df[status]],axis=1,join='inner')
        corrmat = df.corr()
        label_row = corrmat[status]
        grater = label_row > 0
        sum_more_than[column[i]] += grater.sum()
for key in sum_more_than:
    sum_more_than[key] *=  1/3
print("status ranking features -> ")
print(sorted(sum_more_than.items(), key=lambda x: x[1], reverse=True))
```

```
status ranking features ->
[('f2', 57.666666666666664), ('f3', 54.666666666666664), ('f1', 19.333333333333332), ('f4', 13.333333333333332), ('f5', 3.33
3333333333333)]
```

و در آخربرای label مربوط به status میانگین مقادیر بدست آمده برای f1 تا f5 را بدست آورده و sort میکنیم که نتایج آن در روبرو آمده است :

➢ Clustering :

در این قسمت با توجه به اینکه نویز و عوامل مختلف بر داده ها اثر گذاشته اند، خوشه های مختلف به هم نزدیک شده و احتمالا ادغام می شوند. به طوریکه حتی در استفاده از کل داده ها ، هیچ یک از مدل های Kmeans, Hierarchical که در این پروژه استفاده کردیم ، نمی توانند دقت مناسبی داشته باشند.

ابتدا با تعداد بعد 2 شروع کردیم. از آنجائیکه Gender هم به دو دسته تقسیم شده ، پس در این حالت، دو دسته جنسیت را نشان می دهد.در روش Kmeans ، random state را مقدار ثابتی قرار دادیم ضمنا باید به این موضوع توجه داشت که انتخاب centroid اولیه برای هر دسته در همگرایی به پاسخ نهایی و مطلوب نقش مهمی دارد به این منظور مقدار init را روی k-means++ قرار می‌دهیم تا با مشکل initial trap مواجه نشویم .

از آنجائیکه در ستون Gender دو دسته "زن" یا "مرد" وجود دارد پس می توان آنرا به صورت باینری در نظر گرفت.(با جایگشت 2) لیبل های 0 یا 1 با کد زیر توسط مدل آموزش دیده به داده ها اختصاص می یابند :

```
df_cluster['Gender'] =df_cluster['Gender'].astype('category').cat.codes
```

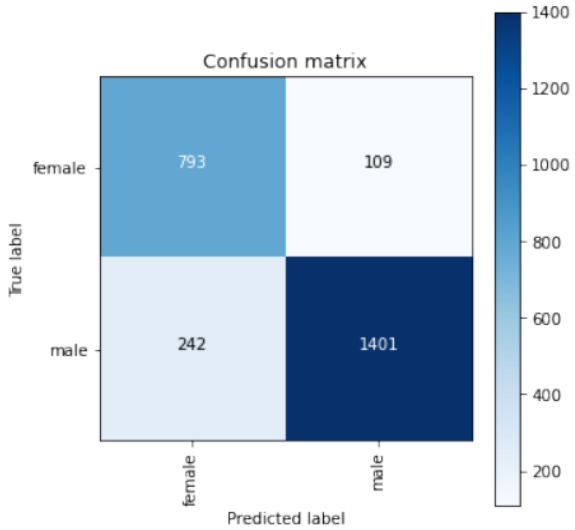
*نکته‌ی مهم اینجاست که ما از توزیع باینری در Label آموزش داده ها اطلاع داریم (این کا را با تطبیق ستون Gender باینری شده با ستون حاوی male,female متوجه می‌شویم که در اینجا 0 به female و 1 به male اختصاص یافته است) . ولی نمی‌دانیم خروجی حاصل از clustering (y_pred) به هر کدام از female,male ها چه مقداری را از بین 0 و 1 نسبت داده است .

برای این منظور نیاز داریم که 2 جایگشت مکمل هم از y_pred را در نظر بگیریم : فرض کنید که خروجی حاصل از y_pred بصورت [0,1,1,…,0,0,1] باشد . بنابراین جایگشت دیگر این خروجی بصورت مکمل آن می‌باشد . یعنی هر المان داخل لیست فوق را not کنیم به جایگشت دوم می‌رسیم .به عبارت دیگر جمع دو جایگشت اول و دوم یک لیست با تمام المان های 1 می‌باشد . همین الگوریتم را در کد برای بدست آوردن جایگشت دوم استفاده می‌کنیم :

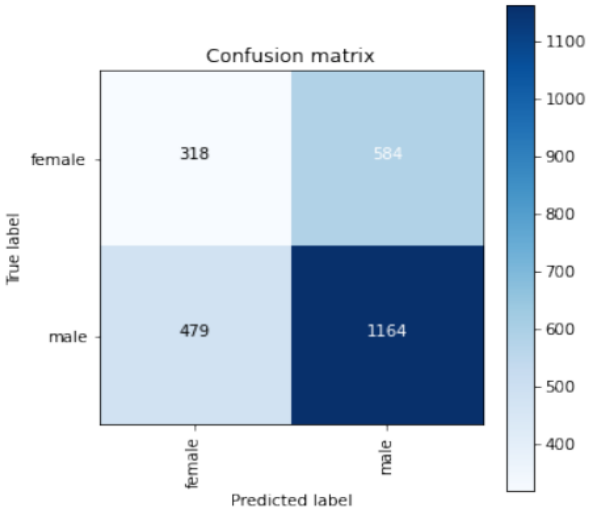
```
y_k_tilda=np.ones((feature[0].shape[0]))-y_pred
```

در نهایت confusion matrix و f1_score بیشینه بین هر دو جایگشت را برای f1 تا f5 رسم می‌کنیم :

F1_score(f1) = 0.8886774500475737

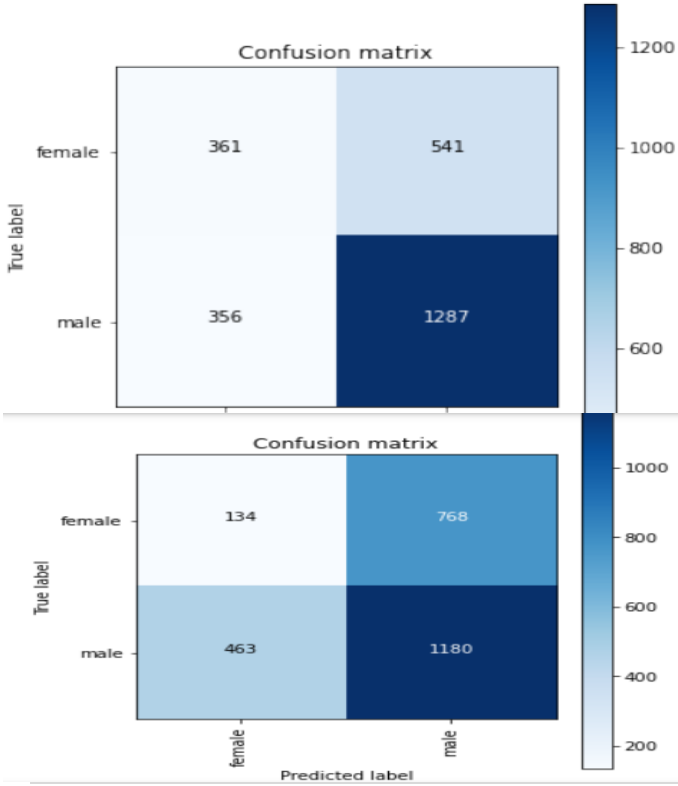


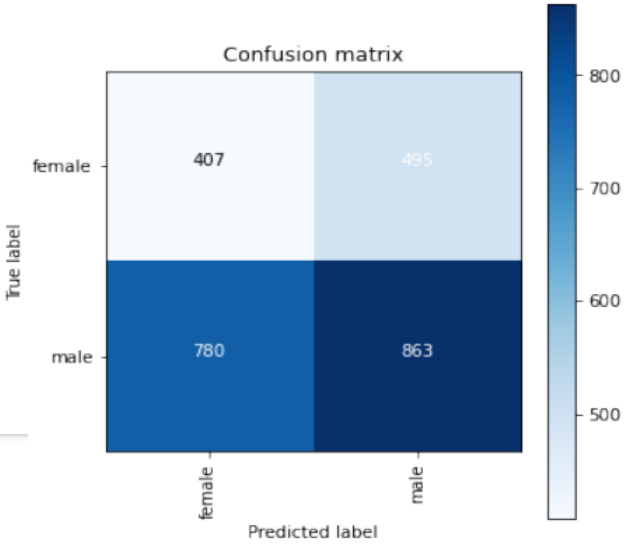
F1_score(f2) = 0.6865231495134179



F1_score(f3) = 0.7415730337078652

F1_score(f4) = 0.5751416194601799





F1_score(f5) = 0.6571985519353941

سپس با تعداد بعد 3 مدل سازی کردیم. با توجه به اینکه Status سه وضعیت دارد ، دسته ها در این حالت، سه حالت چهره را نشان می دهند . همانند دسته بندی برای جنسیت ،در اینجا 3 لیبل 0 ، 1 ،2 به داده ها تخصیص می یابد.(با جایگشت 6). برای ایجاد جایگشت های مختلف برای y_pred فرض می‌کنیم که در هر زمان یکی از سه مقدار 0,1,2 ثابت باشد و دو مقدار دیگر جابجا شوند :

$$\begin{cases} [0,1,2] \rightarrow [0,2,1] \\ [1,0,2] \rightarrow [1,2,0] \\ [2,0,1] \rightarrow [2,1,0] \end{cases}$$

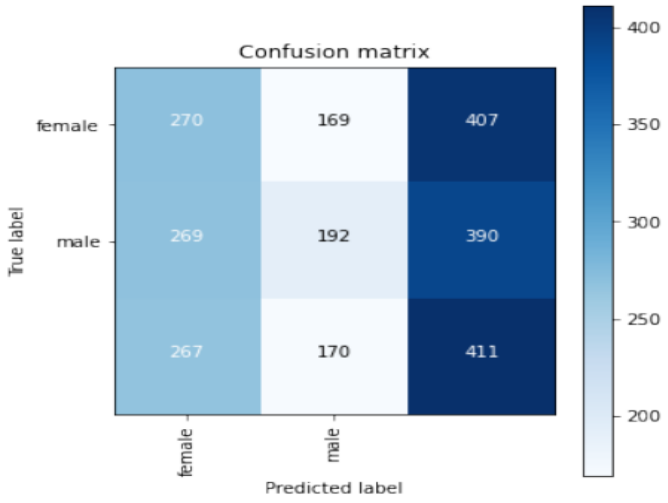
در ادامه تابعی بنام Get_sequnece می‌نویسیم که با گرفتن y_pred ، 6 جایگشت مختلف را درون یک لیست ذخیره می‌کند.

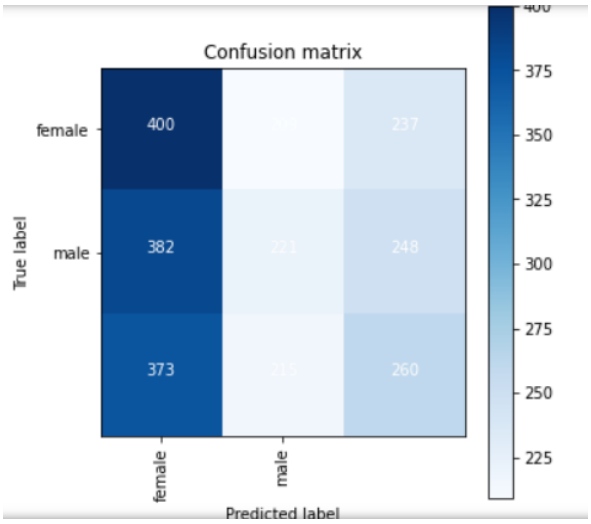
```python
def Get_sequnece(y_pred):
    y_pred_2  = y_pred
    y_pred_3  = y_pred

    for i in range(len(y_pred)):
        y_pred_2 = np.where(y_pred == 0 , 3 , y_pred_2)
        y_pred_2 = np.where(y_pred == 1 , 0 , y_pred_2)
        y_pred_2 = np.where(y_pred_2 == 3 , 1 , y_pred_2)

        y_pred_3 = np.where(y_pred == 1 , 3 , y_pred_3)
        y_pred_3 = np.where(y_pred == 2 , 1 , y_pred_3)
        y_pred_3 = np.where(y_pred_3 == 3 , 2 , y_pred_3)

    y_pred_4 = y_pred_3
    y_pred_5 = y_pred_3
    for i in range(len(y_pred)):
        y_pred_4 = np.where(y_pred_3 == 0 , 3 , y_pred_4)
        y_pred_4 = np.where(y_pred_3 == 2 , 0 , y_pred_4)
        y_pred_4 = np.where(y_pred_4 == 3 , 2 , y_pred_4)

        y_pred_5 = np.where(y_pred_3 == 0 , 3 , y_pred_5)
        y_pred_5 = np.where(y_pred_3 == 1 , 0 , y_pred_5)
        y_pred_5 = np.where(y_pred_5 == 3 , 1 , y_pred_5)

    y_pred_6 = y_pred_5
    for i in range(len(y_pred)):

        y_pred_6 = np.where(y_pred_5 == 1 , 3 , y_pred_6)
        y_pred_6 = np.where(y_pred_5 == 2 , 1 , y_pred_6)
        y_pred_6 = np.where(y_pred_6 == 3 , 2 , y_pred_6)
    return y_pred,y_pred_2,y_pred_3,y_pred_4,y_pred_5,y_pred_6
```

همانند روش مطرح شده برای n=2 که برای Gender به کار گرفته شد، در اینجا نیز بیشترین مقدار f1_score برای f1 تا f5 بین 6 جایگشت را پیدا می‌کنیم و سپس confusion matrix را رسم می‌کنیم.

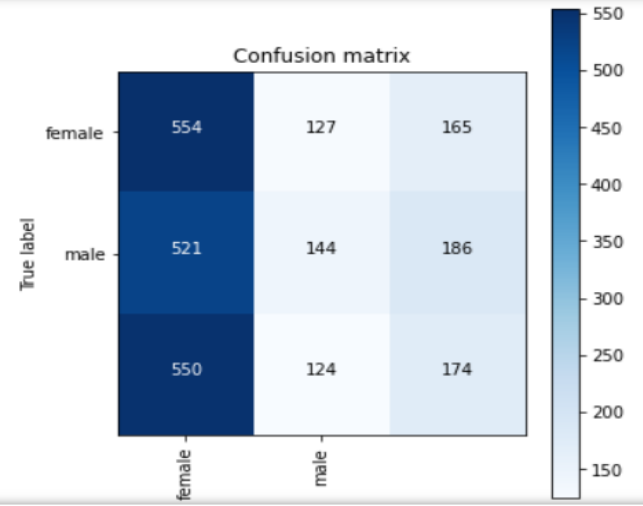
F1_score(f1) = 0.3430255402750491

F1_score(f2) = 0.3461689587426326

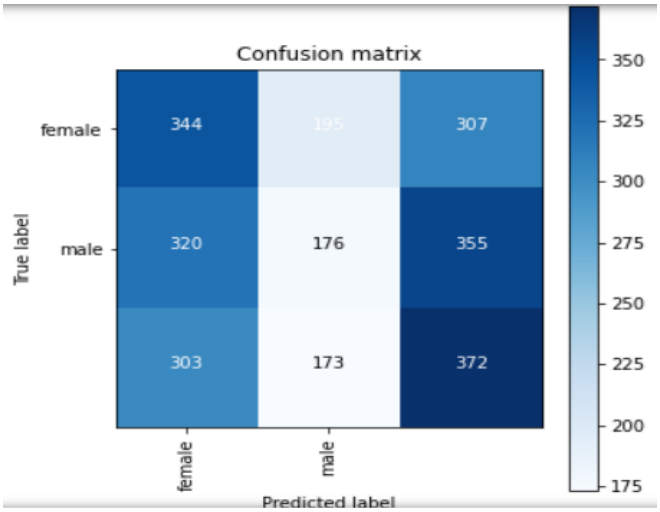




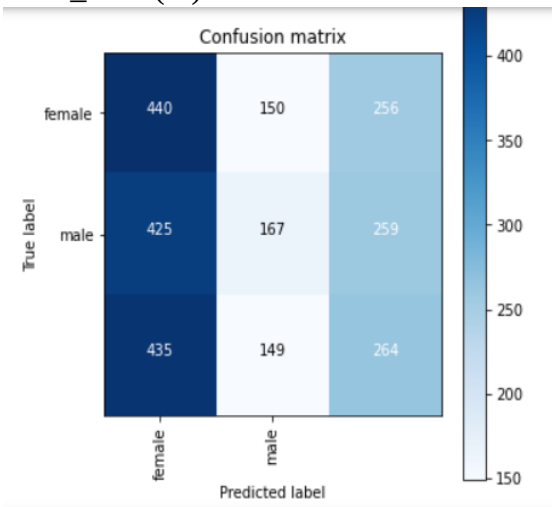
F1_score(f3) = 0.3426326129666012



F1_score(f4) = 0.35049115913555995



F1_score(f5) = 0.3422396856581532



برای تعداد خوشه های 5 و 6 هم که دارای مفهوم می باشد ، خوشه بندی را انجام می دهیم. دسته ها زمانی که n = 5 است، نشان دهنده زاویه دوربین خواهد بود.

هم چنین دسته ها زمانی که n = 6 است ، نشان دهنده حالت و جنسیت (هردو باهم) است. لیبل های اختصاص یافته از 0 تا 5 هستند که تقسیم بندی زیر به صورت فرضی در نظر گرفته شده است .انتظار می رود با افزایش تعداد خوشه ها، خوشه ها به خود سمپل ها تبدیل شوند.

*طبیعی است که اینجا به دلیل اینکه لیبل n=5,6 در داده ها موجود نمی‌باشد ، صرفا خروجی را بدست می‌آوریم و قابلیت تحلیل آن مانند یافتن مقدار f1-score را نداریم .

$0 \rightarrow Female - Happy$

$1 \rightarrow Female - Sad$

$2 \rightarrow Female - Neutral$

$3 \rightarrow Male - Happy$

$4 \rightarrow Male - Sad$

$5 \rightarrow Male - Neutral$

خروجی ها برای n=5,6 بصورت زیر می‌باشد :

Clustering for n=5(5-angle of taken pictures)

```
kmeans= KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
for i in range(0,5):
    y_pred=kmeans.fit_predict(feature[i])
    print("prediction for y by f{} is:{}\n".format(i+1,y_pred))
```

```
prediction for y by f1 is:[2 4 4 ... 2 4 3]

prediction for y by f2 is:[0 1 2 ... 4 2 4]

prediction for y by f3 is:[3 0 0 ... 0 0 1]

prediction for y by f4 is:[2 2 0 ... 1 0 1]

prediction for y by f5 is:[2 1 0 ... 2 0 0]
```

Clustering for n=6(Status & Gender columns)

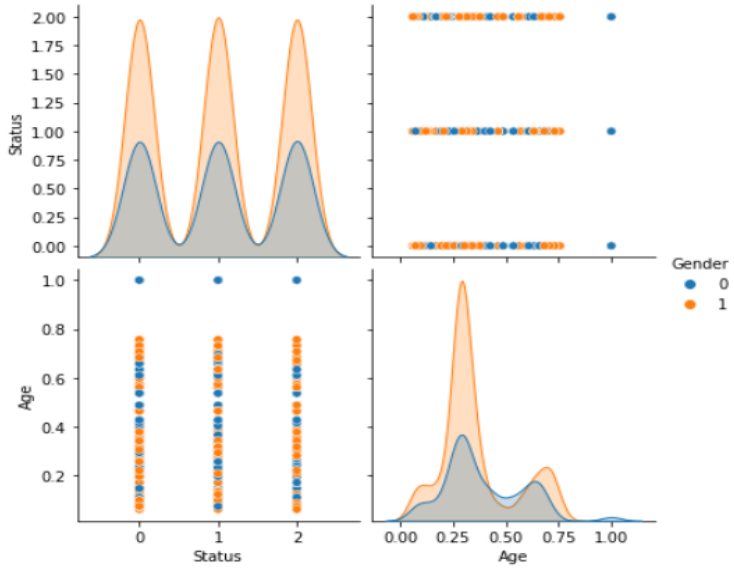
```
kmeans= KMeans(n_clusters = 6, init = 'k-means++', random_state = 42)
for i in range(0,5):
    y_pred=kmeans.fit_predict(feature[i])
    print("prediction for y by f{} is:{}\n".format(i+1,y_pred))
```

```
prediction for y by f1 is:[0 1 5 ... 0 3 2]

prediction for y by f2 is:[2 4 1 ... 2 1 3]

prediction for y by f3 is:[0 2 2 ... 2 2 5]

prediction for y by f4 is:[1 3 3 ... 4 0 4]

prediction for y by f5 is:[1 5 0 ... 1 0 0]
```

در نهایت نمودار توزیع داده‌ها را برای لیبل های Age,Status,Gender توسط Sns رسم می‌کنیم :

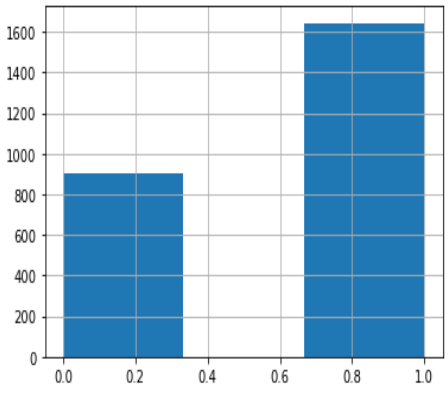
```
frames=[df_cluster["Status"],df_cluster["Gender"],df_cluster["Age"]]
F=pd.concat(frames,axis=1)
sns.pairplot(F,hue='Gender',height=3)
```

```
<seaborn.axisgrid.PairGrid at 0x1ab81ce7dc0>
```

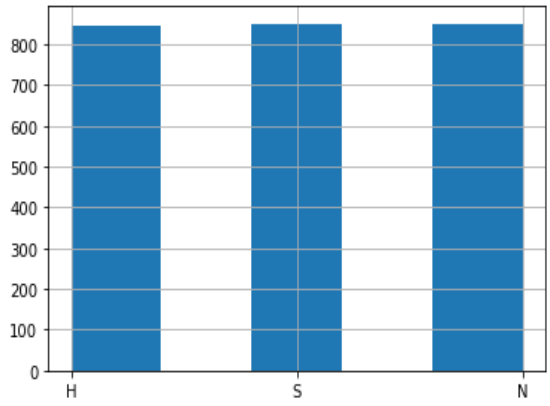


همانطور که مشخص است چون سن یک مقدار پیوسته دارد ، توزیع آن بر حسب جنسیت به خوبی ترسیم شده است ولی چون لیبل های Status و Gender دارای مقادیر گسسته هستند ، نمودار رسم شده برای آنها درهم بوده و اطلاعات خاصی نمی‌دهد ، به همین دلیل برای این دو، Histogram را بر حسب لیبل های female,male برای Gender و لیبل‌های H,N,S برای Status رسم می‌کنیم :

```
In [469]: df_final.loc[:,"Gender"].hist(bins=3)
Out[469]: <AxesSubplot:>
```




```
df_final.loc[:,"Status"].hist(bins=5)
<AxesSubplot:>
```



➢ Classification :

این بخش دارای مساله‌ی طبقه بندی برای دو لیبل زیر می‌باشد :

- ❖ Gender label
- ❖ Status Label

A. Gender Label : برای اینکه طبقه‌بندی را برای لیبل جنسیت انجام دهیم ، ابتدا باید مقادیر X و Y را مشخص کنیم. طبیعتا Y ، ستون مربوط به لیبل جنسیت در Dataframe و X هر کدام از feature های f1 تا f5 می‌تواند باشد . ما مساله‌ی طبقه بندی برای شش classifier مختلف برسی می‌کنیم :

- KNN
- SVM
- Kernel_SVM
- Naive Bayes
- Decision Tree
- Random forest

★ KNN : در این روش با استفاده از طبقه بند نزدیک ترین همسایه و با قرار دادن k=5 و قرار دادن metric بر روی minkowski اقدام به طبقه بندی می‌کنیم :

KNN(Estimating Gender column)

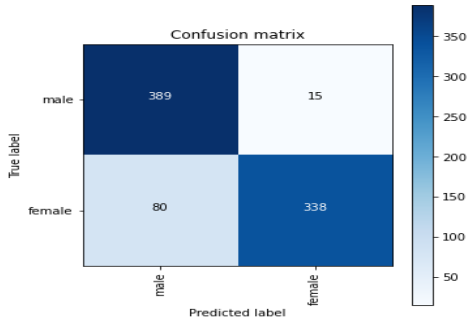
```
a1=[]
target_names = ['male', 'female']
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
for i in range(0,5):
    X_train, X_test, y_train, y_test = train_test_split(f_final[i], final_pd.loc[:,"Gender"].values, test_size = 0.25, random
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    classes = list(dict.fromkeys(final_pd['Gender']))
    print("f{} => \n{}\n".format(i+1,classification_report(y_test, y_pred, target_names=target_names)))
    plot_confusion_matrix(confusion_matrix(y_test,y_pred),classes)
    a1.append(accuracy_score(y_test, y_pred))
```

در ابتدا داده‌ها را به test & tarin تقسیم بندی می‌کنیم و سپس Classifier مور نظر را به داده‌های آموزش fit می‌کنیم .

سپس طبق خواسته‌ی سوال ماتریس confusion و مقادیر precision,recall,f1-score,accuracy را برای هر کدام از f1 تا f5 بدست می‌آوریم :

نمونه‌ی خروجی برای f1 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

```
f1 =>
              precision    recall  f1-score   support

        male       0.83      0.96      0.89       404
      female       0.96      0.81      0.88       418

    accuracy                           0.88       822
   macro avg       0.89      0.89      0.88       822
weighted avg       0.89      0.88      0.88       822
```

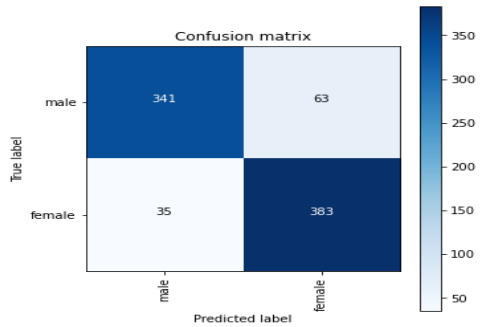



```
acc1 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a1 })
acc1.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.923358
2	f3	0.923358
0	f1	0.884428
3	f4	0.692214
4	f5	0.625304

★ SVM : مشابه قسمت قبل تنها نوع classifier را به svm تغییر داده و kernel آنرا بر روی حالت linear قرار می‌دهیم . نمونه‌ی خروجی برای f1 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

```
f1 =>
              precision    recall  f1-score   support

        male       0.91      0.84      0.87       404
      female       0.86      0.92      0.89       418

    accuracy                           0.88       822
   macro avg       0.88      0.88      0.88       822
weighted avg       0.88      0.88      0.88       822
```

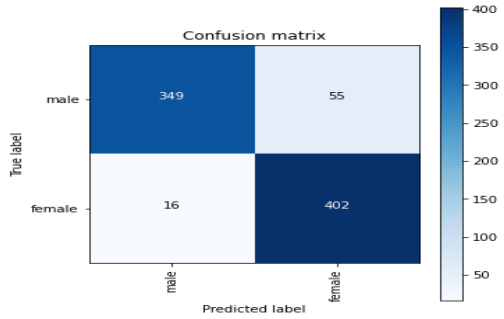



```
acc2 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a2 })
acc2.sort_values("column2",ascending=False)
```

	column1	column2
2	f3	0.944039
1	f2	0.931873
0	f1	0.880779
3	f4	0.613139
4	f5	0.608273

★ Kernel_SVM : فرق این حالت با SVM بخش قبل در این است که تنها kernel را روی rbf تنظیم می‌کنیم . نمونه‌ی خروجی برای f1 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

```
f1 =>
              precision    recall  f1-score   support

        male       0.96      0.86      0.91       404
      female       0.88      0.96      0.92       418

    accuracy                           0.91       822
   macro avg       0.92      0.91      0.91       822
weighted avg       0.92      0.91      0.91       822
```

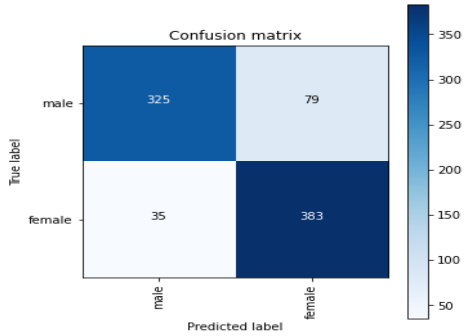



```
acc3 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a3 })
acc3.sort_values("column2",ascending=False)
```

	column1	column2
2	f3	0.945255
1	f2	0.940389
0	f1	0.913625
3	f4	0.714112
4	f5	0.642336

★ Naïve bayes : GaussianNB را به عنوان Classifier انتخاب می‌کنیم . نمونه‌ی خروجی برای f1 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

```
f1 =>
              precision    recall  f1-score   support

        male       0.90      0.80      0.85       404
      female       0.83      0.92      0.87       418

    accuracy                           0.86       822
   macro avg       0.87      0.86      0.86       822
weighted avg       0.87      0.86      0.86       822
```

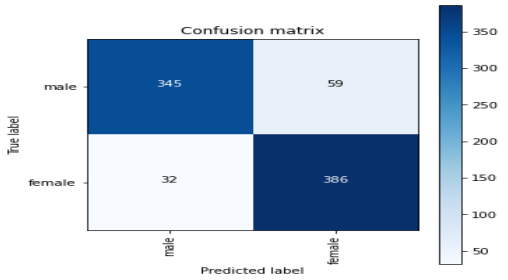



```
acc4 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a4 })
acc4.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.902676
2	f3	0.877129
0	f1	0.861314
4	f5	0.610706
3	f4	0.591241

★ Decision Tree : DecisionTreeClassifier را به عنوان طبقه بند و criterion را روی entropy قرار می‌دهیم. . نمونه‌ی خروجی برای f1 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

```
f1 =>
              precision    recall  f1-score   support

        male       0.92      0.85      0.88       404
      female       0.87      0.92      0.89       418

    accuracy                           0.89       822
   macro avg       0.89      0.89      0.89       822
weighted avg       0.89      0.89      0.89       822
```

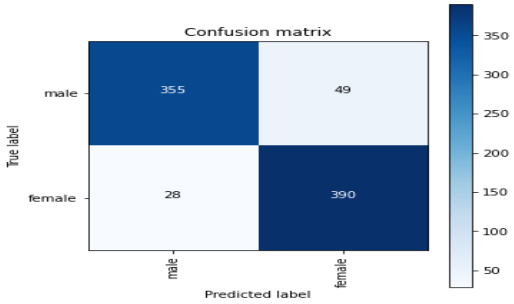



```
acc6 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a6})
acc6.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.909976
2	f3	0.908759
0	f1	0.906326
3	f4	0.784672
4	f5	0.716545

★ Random forest : تنها n_estimators = 10 را به Classifier اضافه می‌کنیم . نمونه‌ی خروجی برای f1 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

```
f1 =>
              precision    recall  f1-score   support

        male       0.93      0.88      0.90       404
      female       0.89      0.93      0.91       418

    accuracy                           0.91       822
   macro avg       0.91      0.91      0.91       822
weighted avg       0.91      0.91      0.91       822
```




```
acc6 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a6})
acc6.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.909976
2	f3	0.908759
0	f1	0.906326
3	f4	0.784672
4	f5	0.716545

خلاصه‌ی 6 طبقه بند بالا بصورت sort شده بر حسب ستون Svm در جدول زیر آمده است :

	feature	KNN	SVM	K_SVM	NB	DT	RF
0	f3	0.923358	0.944039	0.945255	0.877129	0.916058	0.908759
1	f2	0.923358	0.931873	0.940389	0.902676	0.913625	0.909976
2	f1	0.884428	0.880779	0.913625	0.861314	0.889294	0.906326
3	f4	0.692214	0.613139	0.714112	0.591241	0.773723	0.784672
4	f5	0.625304	0.608273	0.642336	0.610706	0.700730	0.716545

طبق جدول برای تمام طبقه بند ها، feature های f1,f4,f5 از نظر accuracy در آخر جدول قرار گرفته اند و در سه مورد f2 و سه مورد دیگر f3 دارای accuracy بیشتری می‌باشد و اگر با نتایج حاصل از بخش feature set برای لیبل Gender مقایسه کنیم ، می‌بینیم که در آنجا هم ترتیب feature های بدست آمده از چپ به راست و از نظر کارایی بصورت f2,f3,f1,f4, f5 می‌باشد .

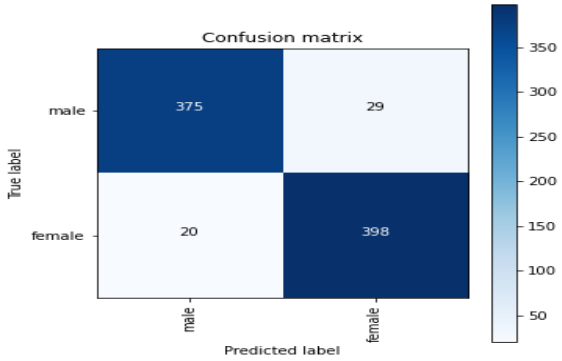
* باید دقت شود که روش اصولی در قسمت feature set برسی شد و اینجا تنها نتایج با هم مقایسه شد.

همچنین با مقایسه‌ی طبقه بند های فوق K_SVM برای سه feature اول بسیار خوب جواب داده و accuracy بالای 90 درصد می‌دهد ولی برای دو feature آخر افت کرده و برای feature آخر مقدار 64 درصد را ایجاد کرده است . در طرف دیگر RF برای تمام feature ها خوب عمل کرده و accuracy بالای 70 برمی‌گرداند ولی برای دو feature اصلی f2,f3 به اندازه‌ی K_SVM کارا نیست.

به همین دلیل K_SVM را به عنوان طبقه بند مطلوب لیبل Gender انتخاب می‌کنیم و Classification report را برای آن در زیر می‌آوریم : (برای f1 قبلا آورده شده است)

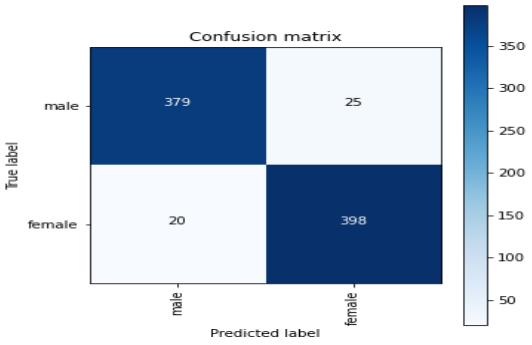
f2 =>

	precision	recall	f1-score	support
male	0.95	0.93	0.94	404
female	0.93	0.95	0.94	418
accuracy			0.94	822
macro avg	0.94	0.94	0.94	822
weighted avg	0.94	0.94	0.94	822



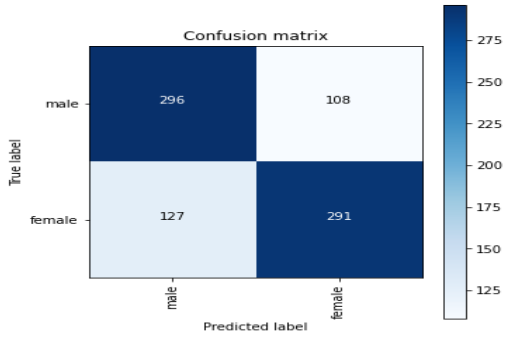
f3 =>

	precision	recall	f1-score	support
male	0.95	0.94	0.94	404
female	0.94	0.95	0.95	418
accuracy			0.95	822
macro avg	0.95	0.95	0.95	822
weighted avg	0.95	0.95	0.95	822



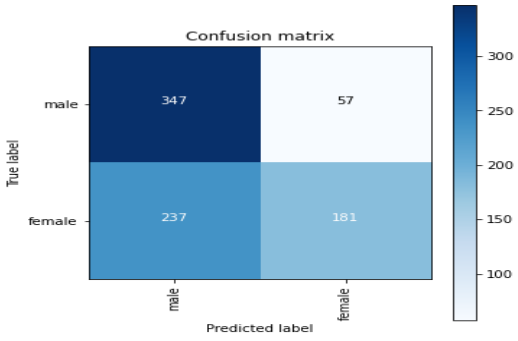
f4 =>

	precision	recall	f1-score	support
male	0.70	0.73	0.72	404
female	0.73	0.70	0.71	418
accuracy			0.71	822
macro avg	0.71	0.71	0.71	822
weighted avg	0.71	0.71	0.71	822



f5 =>

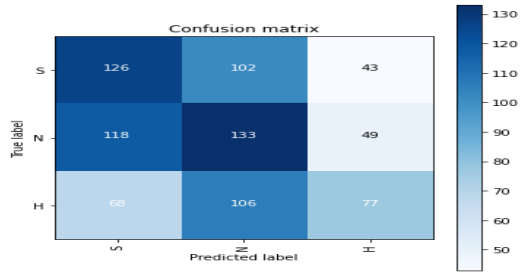
	precision	recall	f1-score	support
male	0.59	0.86	0.70	404
female	0.76	0.43	0.55	418
accuracy			0.64	822
macro avg	0.68	0.65	0.63	822
weighted avg	0.68	0.64	0.63	822



B. Status label : طبیعتا Y ، ستون مربوط به لیبل حالت هر فرد در Dataframe و X هر کدام از feature های f1 تا f5 می‌تواند باشد . ما مساله‌ی طبقه بندی برای 6 classifier قسمت A تکرار می‌کنیم :

★ KNN : نمونه‌ی خروجی برای f2 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

```
f2 =>
              precision    recall  f1-score   support

           H       0.40      0.46      0.43       271
           S       0.39      0.44      0.41       300
           N       0.46      0.31      0.37       251

    accuracy                           0.41       822
   macro avg       0.42      0.41      0.40       822
weighted avg       0.41      0.41      0.41       822
```

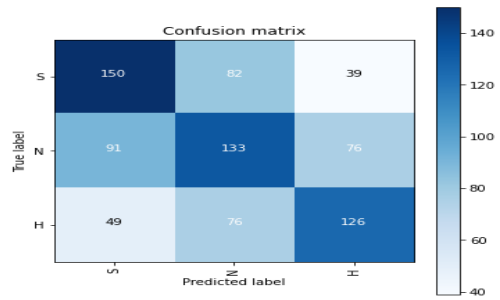



```
acc1 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a1})
acc1.sort_values("column2",ascending=False)
```

	column1	column2
3	f4	0.427007
0	f1	0.412409
1	f2	0.408759
2	f3	0.396594
4	f5	0.350365

★ SVM : نمونه‌ی خروجی برای f2 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

```
f2 =>
              precision    recall  f1-score   support

           H       0.52      0.55      0.53       271
           S       0.46      0.44      0.45       300
           N       0.52      0.50      0.51       251

    accuracy                           0.50       822
   macro avg       0.50      0.50      0.50       822
weighted avg       0.50      0.50      0.50       822
```



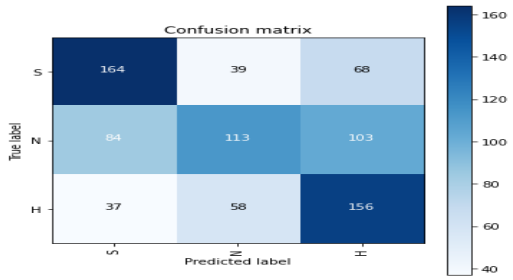

```
acc2 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a2})
acc2.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.497567
2	f3	0.463504
0	f1	0.409976
3	f4	0.380779
4	f5	0.324818

★ K_SVM : نمونه‌ی خروجی برای f2 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

f2 =>

	precision	recall	f1-score	support
H	0.58	0.61	0.59	271
S	0.54	0.38	0.44	300
N	0.48	0.62	0.54	251
accuracy			0.53	822
macro avg	0.53	0.53	0.52	822
weighted avg	0.53	0.53	0.52	822



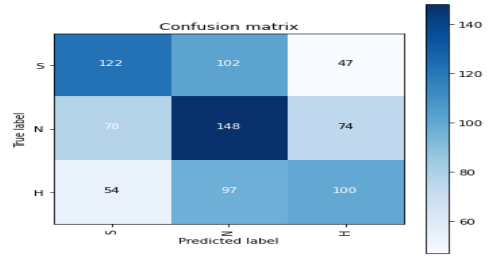

```
acc3 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a3})
acc3.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.526764
2	f3	0.492701
0	f1	0.490268
3	f4	0.391727
4	f5	0.310219

★ Naïve bayes : نمونه‌ی خروجی برای f2 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

f2 =>

	precision	recall	f1-score	support
H	0.48	0.45	0.46	271
S	0.43	0.49	0.46	300
N	0.45	0.40	0.42	251
accuracy			0.45	822
macro avg	0.45	0.45	0.45	822
weighted avg	0.45	0.45	0.45	822



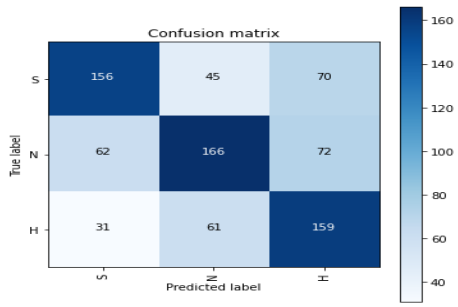

```
acc4 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a4})
acc4.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.450122
2	f3	0.442822
0	f1	0.403893
3	f4	0.361314
4	f5	0.343066

★ Decision Tree : نمونه‌ی خروجی برای f2 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

f2 =>

	precision	recall	f1-score	support
H	0.63	0.58	0.60	271
S	0.61	0.55	0.58	300
N	0.53	0.63	0.58	251
accuracy			0.59	822
macro avg	0.59	0.59	0.59	822
weighted avg	0.59	0.59	0.59	822




```
acc5 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a5})
acc5.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.585158
0	f1	0.582725
3	f4	0.568127
2	f3	0.565693
4	f5	0.434307

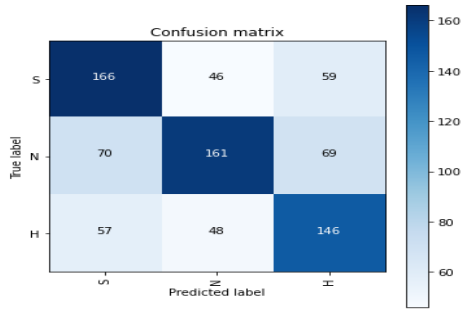
★ Random forest : نمونه‌ی خروجی برای f2 و همچنین خلاصه‌ی نتایج sort شده برای f1 تا f5 آورده شده است :

f2 =>

	precision	recall	f1-score	support
H	0.57	0.61	0.59	271
S	0.63	0.54	0.58	300
N	0.53	0.58	0.56	251
accuracy			0.58	822
macro avg	0.58	0.58	0.58	822
weighted avg	0.58	0.58	0.58	822

```
acc6 = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":a6})
acc6.sort_values("column2",ascending=False)
```

	column1	column2
1	f2	0.575426
2	f3	0.563260
3	f4	0.546229
0	f1	0.540146
4	f5	0.452555



خلاصه‌ی 6 طبقه بند بالا بصورت sort شده بر حسب ستون Svm در جدول زیر آمده است :

	feature	KNN	SVM	K_SVM	NB	DT	RF
0	f2	0.408759	0.497567	0.526764	0.450122	0.585158	0.575426
1	f3	0.396594	0.463504	0.492701	0.442822	0.565693	0.563260
2	f1	0.412409	0.409976	0.490268	0.403893	0.582725	0.540146
3	f4	0.427007	0.380779	0.391727	0.361314	0.568127	0.546229
4	f5	0.350365	0.324818	0.310219	0.343066	0.434307	0.452555

طبق جدول برای تمام طبقه بند ها به جز KNN که در آن f4 از بقیه کارایی بهتری داشته ، feature های f1,f4,f5 از نظر accuracy در آخر جدول قرار گرفته اند و در تمامی موارد f2 از f3 بالاتر قرار گرفته است و دارای accuracy بیشتری می‌باشد و اگر با نتایج حاصل از بخش feature set برای لیبل Status مقایسه کنیم ، می‌بینیم که در آنجا هم ترتیب feature های بدست آمده از چپ به راست و از نظر کارایی بصورت f2,f3,f1,f4, f5 می‌باشد .

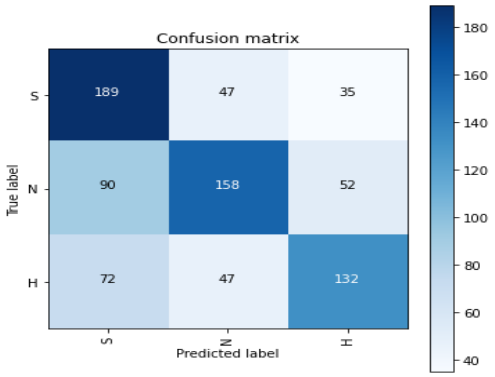
* باید دقت شود که روش اصولی در قسمت feature set برسی شد و اینجا تنها نتایج با هم مقایسه شد.

همچنین با مقایسه‌ی طبقه بند های فوق DT برای چهار feature اول بسیار خوب جواب داده و accuracy بالای 55 درصد می‌دهد و فقط برای feature آخر افت کرده است.

به همین دلیل DT را به عنوان طبقه بند مطلوب لیبل Status انتخاب می‌کنیم و Classification report را برای آن در زیر می‌آوریم : (برای f2 قبلا آورده شده است)

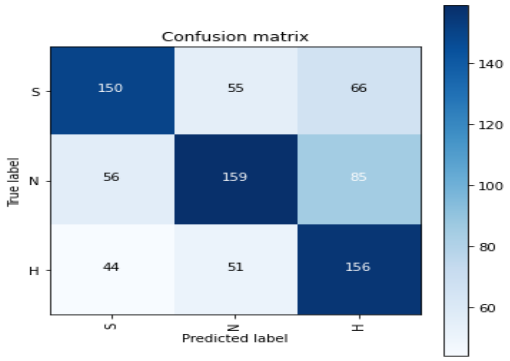
f1 =>

	precision	recall	f1-score	support
H	0.54	0.70	0.61	271
S	0.63	0.53	0.57	300
N	0.60	0.53	0.56	251
accuracy			0.58	822
macro avg	0.59	0.58	0.58	822
weighted avg	0.59	0.58	0.58	822



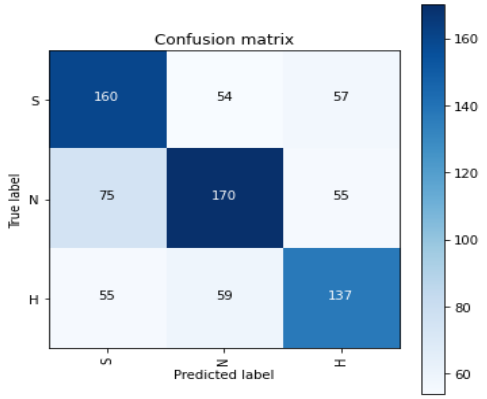
f3 =>

	precision	recall	f1-score	support
H	0.60	0.55	0.58	271
S	0.60	0.53	0.56	300
N	0.51	0.62	0.56	251
accuracy			0.57	822
macro avg	0.57	0.57	0.57	822
weighted avg	0.57	0.57	0.57	822



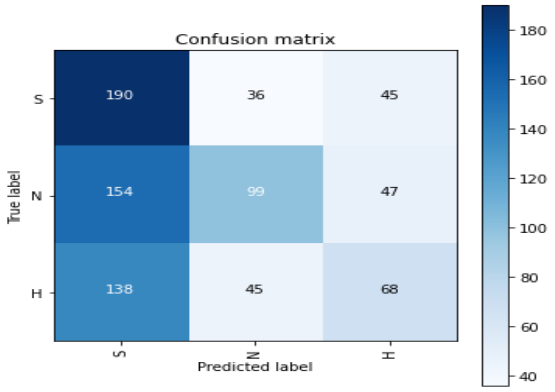
f4 =>

	precision	recall	f1-score	support
H	0.55	0.59	0.57	271
S	0.60	0.57	0.58	300
N	0.55	0.55	0.55	251
accuracy			0.57	822
macro avg	0.57	0.57	0.57	822
weighted avg	0.57	0.57	0.57	822



f5 =>

	precision	recall	f1-score	support
H	0.39	0.70	0.50	271
S	0.55	0.33	0.41	300
N	0.42	0.27	0.33	251
accuracy			0.43	822
macro avg	0.46	0.43	0.42	822
weighted avg	0.46	0.43	0.42	822



➢ Linear Regression : در این قسمت تنها کافی است که لیبل Age را توسط feature های f1 تا f5 بصورت خطی تخمین بزنیم و مقادیر MSE را اعلام کنیم . مانند قسمت های قبل ابتدا Classifier را با LinearRegression() ایجاد می‌کنیم و بعد از تقسیم داده‌ها به test و train ، آنها را fit می‌کنیم :

```
mse=[]
for i in range(0,5):
    X_train, X_test, y_train, y_test = train_test_split(f_final[i], final_pd.loc[:,"Age"].values, test_size = 0.25, random_st
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)
    classes = list(dict.fromkeys(final_pd['Age']))
    mse.append(mean_squared_error(y_test, y_pred))
    print("f{} MSE is : {}\n".format(i+1,mean_squared_error(y_test, y_pred)))
```

وخروجی بصورت زیر خواهد بود :

```
f1 MSE is : 0.017240455617756712

f2 MSE is : 0.010456170413864077

f3 MSE is : 0.009982742513663883

f4 MSE is : 0.030166718994366997

f5 MSE is : 0.03179490158004619
```

```
MSE = pd.DataFrame({"column1": ["f1", "f2", "f3", "f4","f5"], "column2":mse})
MSE.sort_values("column2",ascending=False).reset_index(drop=True)
```

	column1	column2
0	f5	0.031795
1	f4	0.030167
2	f1	0.017240
3	f2	0.010456
4	f3	0.009983

همانطور که در شکل بالا دیده می‌شود ، feature های f1 تا f5 را بر حسب مقادیر MSE از بزرگ به کوچک rank می‌کنیم . طبق نتایج، f3 feature در این میان از بقیه کاراتر است(MSE کوچکتر) و به ترتیب feature های f2,f1,f4,f5 قرار می‌گیرند. اگر ranking مربوط به feature ها در بخش feature set را برای ستون Age را برسی کنیم ، با نتایج بدست آمده هماهنگی دارد.

* باید دقت شود که روش اصولی در قسمت feature set برسی شد و اینجا تنها نتایج با هم مقایسه شد.

پایان ...