شرح پروژه : پیش بینی گونه لوبیا با استفاده ماشین لرنینگ

ابتدا باید فایل دیتا را وارد برنامه ژوپیتر نوتبوک کنیم و پس از انجام کار های پیش پردازش دیتا مدلسازی را انجام دهیم .

ایمپورت کتابخانه ها موارد مورد نیاز:

```
:
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5
6  from sklearn.model_selection import train_test_split
7  from sklearn.ensemble import RandomForestClassifier
8  from sklearn.svm import SVC
9  from sklearn.naive_bayes import GaussianNB
10 from sklearn.neighbors import KNeighborsClassifier
11
12 from sklearn import metrics
13
14 from sklearn.metrics import classification_report , confusion_matrix
15
```

چهار کتابخانه اول برای پیش پردازش دیتا و سایر کتابخانه ها برای مدلسازی استفاده میشود.

فراخوانی دیتا:

```
1 data = pd.read_excel("C:/Users/mmmma/OneDrive/Desktop/project/Dry_Bean_Dataset.xlsx")
```

با استفاده از پانداس فایل اکسل دیتا را ایمپورت میکنیم.

تغییر فرمت دیتا به CSV:

```
1 data.to_csv("C:/Users/mmmma/OneDrive/Desktop/project/Dry_Bean_Dataset.xlsx", index = False , encoding = "utf-8")
```

برای انجام کارهای مورد نیاز ابتدا فایل اکسل را به فرمت سی اس وی تبدیل میکنیم.

تبدیل دیتا به دیتافریم:

```
df = pd.DataFrame(data)
df
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.141097	0.763923	0.988856	0.958027
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.272750	0.783968	0.984986	0.887034
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.410904	0.778113	0.989559	0.947849
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.467062	0.782681	0.976696	0.903936
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.896503	0.773098	0.990893	0.984877
...	...	...	...	...	...	...	...	...	...	...	...
13606	42097	759.696	288.721612	185.944705	1.552728	0.765002	42508	231.515799	0.714574	0.990331	0.916603
13607	42101	757.499	281.576392	190.713136	1.476439	0.735702	42494	231.526798	0.799943	0.990752	0.922015
13608	42139	759.321	281.539928	191.187979	1.472582	0.734065	42569	231.631261	0.729932	0.989899	0.918424
13609	42147	763.779	283.382636	190.275731	1.489326	0.741055	42667	231.653248	0.705389	0.987813	0.907906
13610	42159	772.237	295.142741	182.204716	1.619841	0.786693	42600	231.686223	0.788962	0.989648	0.888380

از دیتای خود یک دیتافریم تشکیل میدهیم که بتوانیم روی آن کار کنیم.

```
df.describe(include = "all")
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity
count	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	53048.284549	855.283459	320.141867	202.270714	1.583242	0.750895	53768.200206	253.064220	0.749733	0.987143
std	29324.095717	214.289696	85.694186	44.970091	0.246678	0.092002	29774.915817	59.177120	0.049086	0.004660
min	20420.000000	524.736000	183.601165	122.512653	1.024868	0.218951	20684.000000	161.243764	0.555315	0.919246
25%	36328.000000	703.523500	253.303633	175.848170	1.432307	0.715928	36714.500000	215.068003	0.718634	0.985670
50%	44652.000000	794.941000	296.883367	192.431733	1.551124	0.764441	45178.000000	238.438026	0.759859	0.988283
75%	61332.000000	977.213000	376.495012	217.031741	1.707109	0.810466	62294.000000	279.446467	0.786851	0.990013
max	254616.000000	1985.370000	738.860153	460.198497	2.430306	0.911423	263261.000000	569.374358	0.866195	0.994677

با استفاده از دستور دسکرایب یک شمای کلی از اطلاعات دیتا بدست می آوریم .

اطلاعاتی مانند ماکسیموم و مینیموم و میانگین و سایر اظلاعات هر ستون را به ما نمایش میدهد.

```
1 df.dtypes
```

```
Area                int64
Perimeter         float64
MajorAxisLength   float64
MinorAxisLength   float64
AspectRation      float64
Eccentricity      float64
ConvexArea          int64
EquivDiameter     float64
Extent            float64
Solidity          float64
roundness         float64
Compactness       float64
ShapeFactor1      float64
ShapeFactor2      float64
ShapeFactor3      float64
ShapeFactor4      float64
Class              object
dtype: object
```

با استفاده از دستور بالا تایپ اطلاعات هر ستون را میتوانیم ببینیم که اغلب آن ها به صورت عدد اعشاری هستند و دو ستون فرمت دو ستون عدد صحیح است و ستون تارگت در کلاس آبجکت قرار دارد.

تبدیل همه ستون ها به نوع فلوت به غیر از ستون تارگت:

```
1 df.iloc[:, :-1] = df.iloc[:, :-1].astype(float)
2 df.dtypes
```

```
Area              float64
Perimeter         float64
MajorAxisLength   float64
MinorAxisLength   float64
AspectRation      float64
Eccentricity      float64
ConvexArea        float64
EquivDiameter     float64
Extent            float64
Solidity          float64
roundness         float64
Compactness       float64
ShapeFactor1      float64
ShapeFactor2      float64
ShapeFactor3      float64
ShapeFactor4      float64
Class              object
dtype: object
```

همه ستون هارا بجز ستون تارگت با استفاده از دستور آیلاک به فرمت فلوت تغییر میدهیم.

ستون تارگت باید جداگانه اطلاعاتش به عدد تبدیل شود.

```
1 print("missing values of dataframe : \n" , df.isnull().sum())
```

```
missing values of dataframe :
 Area               0
Perimeter           0
MajorAxisLength     0
MinorAxisLength     0
AspectRation        0
Eccentricity        0
ConvexArea          0
EquivDiameter       0
Extent              0
Solidity            0
roundness           0
Compactness         0
ShapeFactor1        0
ShapeFactor2        0
ShapeFactor3        0
ShapeFactor4        0
Class               0
dtype: int64
```

```
1 df.dropna(inplace = True)
```

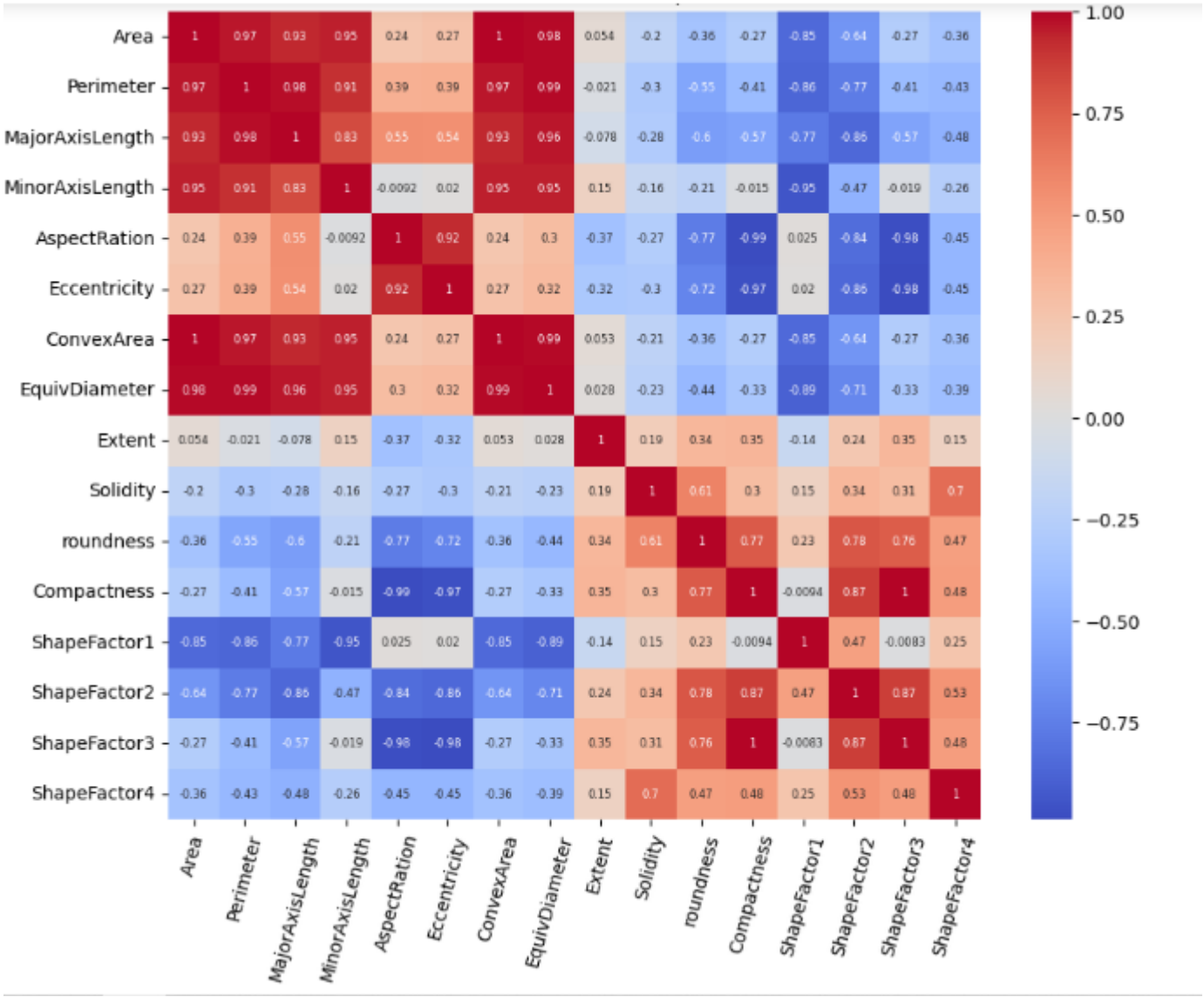
با استفاده از دستورات بالا ابتدا متوجه میشویم که که هیچ ستونی دارای میسینگ ولیو نیست.

سپس برای اطمینان کامل با استفاده از دستور بعدی میسینگ ولیو های احتمالی را از دیتافریم حذف میکنیم.

نمایش اسکتر پلات همه ستون ها:

```
column_options = df.columns.tolist()

fig = px.scatter(df, x='Area', y='Class')

fig.update_layout(
    updatemenus=[
        dict(
            buttons=[
                dict(
                    label=column,
                    method="update",
                    args=[
                        {"x": [df[column]], "y": [df["Class"]] },
                        {"xaxis": {"title": column }, "yaxis": {"title": "Class"}},
                    ],
                )
                for column in column_options
            ],
            direction="down",
        )
    ]
)

fig.update_traces(
    hovertemplate="%{yaxis.title.text}: %{y}<br>%{xaxis.title.text}: %{x}"
)


fig.show()

```

با استفاده از این کد میتوانیم اسکتر پلات های جداگانه از هر ستون با ستون تارگت را مشاهده کنیم و یک دید و شمای کلی از نحوه ارتباط هر ستون با ستون تارگت به ما ارایه مدهد.



این پلات به ما همبستگی بین فیچر هارا نمایش میدهد که هرچه به عدد 1 نزدیکتر باشد رنگ آن قرمز تر است و نشان دهنده این است که آن ستون ها با هم همبستگی زیادی دارند و هرچه همبستگی کم باشد به سمت صفر میرود و رنگ آن آبی تر میشود و نشان میدهد همبستگی آن ستون ها با یکدیگر کمتر است. عموما در الگوریتم های رگرشن و پیوسته استفاده میشود و در این پروژه که تارگت ما کلسفیکیشن میباشد کمک خاصی نمیکند و فقط یک دید کلی از میزان همبستگی فیچر ها میتوانیم داشته باشیم.

```
df2 = df.copy()
```

```
df2["Class"].unique()
```

```
array(['SEKER', 'BARBUNYA', 'BOMBAY', 'CALI', 'HOROZ', 'SIRA', 'DERMASON'],
      dtype=object)
```

یک دیتافریم دوم با استفاده از کپی گرفتن از دیتافریم اولیه ایجاد میکنیم که تغییرات ما روی دیتافریم اصلی ایجاد نشود و اگر جایی لازم شد بتوانیم برگردیم و اصلاح کنیم.

در خط بعدی با استفاده از دستور یونیک در ستون کلاس مقدار های یونیک ستون تارگت را مشاهده میکنیم که هفت مورد خاص میباشد.

تبدیل کلاس های تارگت به اعداد قابل درک برای سیستم:

```
mapping={'SEKER' : 1 , 'BARBUNYA' : 2 , 'BOMBAY' : 3 , 'CALI' : 4 , 'HOROZ' : 5 , 'SIRA' : 6 , 'DERMASON' : 7 }
```

```
df2["Class"] = df2["Class"].map(mapping)
```

یک دیکشنری ایجاد میکنیم و به هر گونه خاص در ستون تارگت یک عدد اختصاص میدهیم.

سپس با استفاده از دستور مپ ستون تارگت را که مقدار های آبجکتیو داشتند به مقادیر عددی قابل درک برای سیستم تبدیل میکنیم.

```
df2 = df2.astype(float)
```

```
df2.dtypes
```

```
Area              float64
Perimeter         float64
MajorAxisLength   float64
MinorAxisLength   float64
AspectRation      float64
Eccentricity      float64
ConvexArea        float64
EquivDiameter     float64
Extent            float64
Solidity          float64
roundness         float64
Compactness       float64
ShapeFactor1      float64
ShapeFactor2      float64
ShapeFactor3      float64
ShapeFactor4      float64
Class             float64
dtype: object
```

در این مرحله با استفاده از دستور فوق همه ستون هارا به فرمت فلوت تغییر میدهیم که ستون تارگت هم مشاهده میشود که به فرمت فلوت تغییر کرده است و حال میتوانیم مدلسازی را انجام دهیم.

مقیاس بندی و اسکیل کردن دیتافریم (بجز ستون تارگت) برای دقت بالاتر:

```
scaler = preprocessing.MinMaxScaler(feature_range = (0 , 1))
```

```
columns_norm = df2.columns[:-1]
```

```
target = df2.columns[-1]
```

```
norm1 = scaler.fit_transform(df2[columns_norm])
```

```
df2_norm = pd.DataFrame(norm1 , columns = columns_norm)
df2_norm[target] = df2[target]
df2_norm
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compac
0	0.034053	0.058574	0.044262	0.152142	0.122612	0.477797	0.033107	0.070804	0.671024	0.922824	0.934823	0.7
1	0.035500	0.077557	0.030479	0.178337	0.051577	0.278472	0.034991	0.073577	0.735504	0.871514	0.793138	0.9
2	0.038259	0.068035	0.052633	0.158190	0.131521	0.496448	0.037126	0.078816	0.716671	0.932141	0.914511	0.7
3	0.040940	0.082942	0.048548	0.177691	0.091623	0.403864	0.041389	0.083854	0.731365	0.761614	0.826871	0.8
4	0.041504	0.065313	0.032862	0.200679	0.025565	0.165680	0.040123	0.084906	0.700538	0.949832	0.988408	0.9
...	...	...	...	...	...	...	...	...	...	...	...	...
13606	0.092559	0.160862	0.189318	0.187843	0.375584	0.788553	0.089967	0.172180	0.512286	0.942381	0.852151	0.4
13607	0.092576	0.159358	0.176450	0.201964	0.321303	0.746241	0.089910	0.172207	0.786890	0.947954	0.862952	0.5
13608	0.092739	0.160605	0.176384	0.203370	0.318558	0.743877	0.090219	0.172463	0.561689	0.936648	0.855785	0.5

در این مرحله ابتدا لازم است برای بالا بردن دقت مدل ایجاد شده ابتدا مقادیر دیتافریم را مقیاس بندی کنیم که فواصل بین فیچر ها کمتر شود و کار برای سیستم راحت تر شود.

با استفاده از دستورات بالا همه ستون هارا بجز ستون تارگت که مقادیر مشخص آن را خودمان تایین کردیم به شکل مقیاس بندی شده بین عدد صفر و یک در میاوریم.

ایجاد مدل اول با (random forest):

```
### random forest model ###
```

جدا سازی دیتافریم فیچر ها از تارگت:

```
x = pd.DataFrame(df2_norm.drop(columns = [df2_norm.columns[-1]]))
```

```
y = df2_norm['Class'].values.reshape(-1 , 1)
```

ایجاد دیتای تست و دیتای ترین:

```
x_train , x_test , y_train , y_test = train_test_split ( x , y , test_size = 0.3 , random_state = 0)
```

مدلسازی رندوم فارست با دیتای ترین:

```
clf = RandomForestClassifier(max_depth = 10 , n_estimators = 200)
clf.fit(x_train , y_train)
y_pred = clf.predict(x_test)
```

حال همه چیز برای مدلسازی آماده است . ابتدا با مدل رندوم فارست شروع میکنیم.

در مرحله اول فیچر های همه ستون هارا از فیچر های ستون تارگت جدا میکنیم و در ایکس و ایگرگ جدا قرار میدهیم.

سپس آن هارا به دو بخش ترین و تست تقسیم میکنیم که مدل ما با استفاده از دیتای ترین آموزش میبیند و با استفاده از دیتای تست آن را ارزیابی میکنیم.

مدل خود را ایجاد میکنیم و سپس با دیتای ایکس تست پردیکت یا پیش بینی را انجام میدهیم.

اکیوریسی اسکور بر روی تارگت تست و تارگت پیش بینی شده:

```
1 print("accuracy :" , metrics.accuracy_score(y_test , y_pred))
```

```
accuracy : 0.9240940254652301
```

ریپورت گرفتن اسکور های مختلف بر روی کل دیتای تارگت تست و تارگت پیش بینی شده:

```
1 print(classification_report(y , clf.predict(x)))
```

	precision	recall	f1-score	support
1.0	0.98	0.97	0.97	2027
2.0	0.98	0.93	0.95	1322
3.0	1.00	1.00	1.00	522
4.0	0.96	0.97	0.96	1630
5.0	0.98	0.97	0.97	1928
6.0	0.91	0.91	0.91	2636
7.0	0.94	0.96	0.95	3546
accuracy			0.95	13611
macro avg	0.96	0.96	0.96	13611
weighted avg	0.95	0.95	0.95	13611

بر روی ایگرگ تست و ایگرگ پردیکت شده اکیوریسی میگیریم که یک معیار ارزیابی کلی به ما نشان میدهد . اسکور بالای 90 دریافت کرده که مقدار خوبی است.

سپس بر روی کل دیتای ایگرگ و ایگرگ پردیکت بر روی کل ایکس یک کلسفیکیشن ریپورت میگیریم که چندین معیار ارزیابی مهم را به ما نشان میدهد.

همه اسکور ها بالای 90 هستند که نشان میدهد مدل به صورت تقریبا کاملی پیش بینی کرده است

در ستون سوم مقادیر ارزیابی عدد یک را نشان میدهند که به ما میگوید مدل در ستون سوم دچار اورفیت شدگی شده است.

ایجاد کانفیوزن ماتریکس برای نمایش مقدار های پیش بینی شده ی درست و نادرست:

```
confusion_matrix(y , clf.predict(x))
```

```
array([[1965,    1,    0,    0,    0,   46,   15],
       [   6, 1231,    0,   52,    3,   30,    0],
       [   0,    1,  521,    0,    0,    0,    0],
       [   2,   23,    0, 1582,   15,    8,    0],
       [   0,    2,    0,   18, 1862,   33,   13],
       [  12,    3,    0,    3,   14, 2411,  193],
       [  19,    0,    0,    0,    1,  122, 3404]], dtype=int64)
```

```
cm = confusion_matrix(y , clf.predict(x))
fig , ax = plt.subplots(figsize = ( 9 , 9))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks = ( 1 , 2 , 3 , 4 , 5 , 6 , 7), ticklabels = ( "predicted 1s" , "predicted 2s" , "predicted 3s" ,
ax.xaxis.set_tick_params(labelsize = 6)
ax.yaxis.set(ticks = ( 1 , 2 , 3 , 4 , 5 , 6 , 7), ticklabels = ( "actual 1s" , "actual 2s" , "actual 3s" , "actual 4
ax.yaxis.set_tick_params(labelsize = 9)

for i in range(7):
    for j in range(7) :
        ax.text(j , i , cm[i , j], ha = "center" , va = "center" , color = "red")
plt.show()
```

با استفاده از دستور کانفیوژن ماتریکس میتوانیم ببینیم که مقدار های پیش بینی شده چه میزان با مقدار اصلی مطابقت دارند و چه اندازه از مقادیر پیش بینی شده غلط پیش بینی شده اند.

با استفاده از دستور پایینتر هم میتوانیم به صورت گرافیکی مشاهده کنیم که چگونه دیتاها پردیکت شده اند.

¶ ایجاد مدل دوم با (SVM):

```
### SVM Model ###
```

```
x = pd.DataFrame(df2_norm.drop(columns = [df2_norm.columns[-1]]))
```

```
y = df2_norm['Class'].values.reshape(-1 , 1)
```

```
x_train , x_test , y_train , y_test = train_test_split ( x , y , test_size = 0.3 , random_state = 0)
```

```
clf2 = SVC(kernel = "poly" , degree = 4 , gamma = "scale" , coef0 = 2)
clf2.fit(x_train , y_train.ravel())
y_pred = clf2.predict(x_test)
```

مدل دوم را با همان روال قبلی این بار با استفاده از الگوریتم اس وی ام ایجاد میکنیم.

```
1 print("accuracy :" , metrics.accuracy_score(y_test , y_pred))
```

```
accuracy : 0.9297257590597453
```

```
1 print(classification_report(y , clf2.predict(x)))
```

```
              precision    recall  f1-score   support

         1.0       0.96      0.95      0.96      2027
         2.0       0.95      0.93      0.94      1322
         3.0       1.00      1.00      1.00       522
         4.0       0.94      0.95      0.95      1630
         5.0       0.97      0.96      0.96      1928
         6.0       0.88      0.88      0.88      2636
         7.0       0.92      0.93      0.92      3546

    accuracy                           0.93     13611
   macro avg       0.95      0.94      0.94     13611
weighted avg       0.93      0.93      0.93     13611
```

بر روی ایگرگ تست و ایگرگ پردیکت شده اکیوریسی میگیریم که یک معیار ارزیابی کلی به ما نشان میدهد . اسکور بالای 90 دریافت کرده که مقدار خوبی است.

سپس بر روی کل دیتای ایگرگ و ایگرگ پردیکت بر روی کل ایکس یک کلسفیکیشن ریپورت میگیریم که چندین معیار ارزیابی مهم را به ما نشان میدهد.

همه اسکور ها بالای 90 هستند که نشان میدهد مدل به صورت تقریبا کاملی پیش بینی کرده است

در ستون سوم مقادیر ارزیابی عدد یک را نشان میدهند که به ما میگوید مدل در ستون سوم دچار اورفیت شدگی شده است.

```
: 1 confusion_matrix(y , clf2.predict(x))
```

```
: array([[1932,   13,    0,    0,    0,   50,   32],
         [   8, 1224,    0,   64,    5,   21,    0],
         [   0,    0,  521,    1,    0,    0,    0],
         [   5,   40,    0, 1553,   22,   10,    0],
         [   0,    5,    0,   24, 1843,   37,   19],
         [  26,    7,    0,    4,   30, 2317,  252],
         [  46,    1,    0,    0,    3,  186, 3310]], dtype=int64)
```

با استفاده از دستور کانفیوژن ماتریکس میتوانیم ببینیم که مقدار های پیش بینی شده چه میزان با مقدار اصلی مطابقت دارند و چه اندازه از مقادیر پیش بینی شده غلط پیش بینی شده اند.

ایجاد مدل سوم با (Naive Bayes):

```
### Naive Bayes Gaussian model ###
```

```
x = pd.DataFrame(df2_norm.drop(columns = [df2_norm.columns[-1]]))
y = df2_norm['Class'].values.reshape(-1 , 1)
```

```
x_train , x_test , y_train , y_test = train_test_split ( x , y , test_size = 0.3 , random_state = 0)
```

```
clf3 = GaussianNB()
clf3.fit(x_train , y_train.ravel())
y_pred = clf.predict(x_test)
```

مدل سوم را با همان روال قبلی این بار با استفاده از الگوریتم نایو بیز ایجاد میکنیم.

نایو بیز خودش چندین الگوریتم دارد که ما از نایو بیز گاوسی استفاده کرده ایم.

```
print("accuracy :" , metrics.accuracy_score(y_test , y_pred))
```

```
accuracy : 0.9240940254652301
```

```
print(classification_report(y , clf3.predict(x)))
```

```
              precision    recall  f1-score   support

         1.0       0.93      0.94      0.94      2027
         2.0       0.87      0.81      0.84      1322
         3.0       0.99      1.00      1.00       522
         4.0       0.87      0.90      0.89      1630
         5.0       0.95      0.95      0.95      1928
         6.0       0.82      0.87      0.84      2636
         7.0       0.93      0.88      0.90      3546

    accuracy                           0.90     13611
   macro avg       0.91      0.91      0.91     13611
weighted avg       0.90      0.90      0.90     13611
```

بر روی ایگرگ تست و ایگرگ پردیکت شده اکیوریسی میگیریم که یک معیار ارزیابی کلی به ما نشان میدهد . اسکور بالای 90 دریافت کرده که مقدار خوبی است.

سپس بر روی کل دیتای ایگرگ و ایگرگ پردیکت بر روی کل ایکس یک کلسفیکیشن ریپورت میگیریم که چندین معیار ارزیابی مهم را به ما نشان میدهد.

همه اسکور ها بالای 90 هستند که نشان میدهد مدل به صورت تقریبا کاملی پیش بینی کرده است

در ستون سوم مقادیر ارزیابی عدد یک را نشان میدهند که به ما میگوید مدل در ستون سوم دچار اورفیت شدگی شده است.

ایجاد مدل چهارم با (KNN):

```
### KNN Model ###
```

```
x_train , x_test , y_train , y_test = train_test_split ( x , y , test_size = 0.3 , random_state = 0)
```

```
k = 5
clf4 = KNeighborsClassifier(k)
clf4.fit(x_train , y_train.ravel())
```

```
KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
y_pred = clf4.predict(np.array(x_test))
```

مدل چهارم را با همان روال قبلی این بار با استفاده از الگوریتم کی ان ان ایجاد میکنیم.

به صورت پیش فرض مقدار کا را 5 قرار میدهیم.

```
print("accuracy :" , metrics.accuracy_score(y_test , y_pred))
```

```
accuracy : 0.9231145935357493
```

بر روی ایگرگ تست و ایگرگ پردیکت شده اکیوریسی میگیریم که یک معیار ارزیابی کلی به ما نشان میدهد . اسکور بالای 90 دریافت کرده که مقدار خوبی است.

اکنون باید مقدار مناسب کا را بیابیم که بهترین دقت را به ما بدهد.

ایجاد حلقه برای یافتن مقدار مناسبk:

```
k = 20
Acc = np.zeros((k))
for i in range (1 , k+1):
    clf4 = KNeighborsClassifier(n_neighbors = i)
    clf4.fit(x_train , y_train.ravel())
    y_pred = clf4.predict(np.array(x_test))
    Acc[ i - 1 ] = metrics.accuracy_score(y_test , y_pred)

Acc
```

با استفاده از کد بالا و ایجاد حلقه مدل خود را با مقادیر کا از 1 تا 20 ایجاد میکنیم و هر بار اکیوریسی میگیریم تا در نهایت ببینیم کدام مقدار برای کا بیشترین دقت را به ما میدهد.

```
array([0.90817826, 0.89642507, 0.91895201, 0.91821743, 0.92311459,
       0.92213516, 0.92580803, 0.92311459, 0.92531832, 0.92507346,
       0.92556317, 0.92654261, 0.92678746, 0.92629775, 0.92580803,
       0.92580803, 0.92605289, 0.92556317, 0.92507346, 0.92556317])
```

```
Acc.max()
```

```
0.9267874632713027
```

```
Acc.min()
```

```
0.8964250734573947
```

به این صورت به ما نمایش داده میشود که مقدار ماکس و مین آن را هم میتوانیم ببینیم.

استفاده از گرید سرچ برای یافتن عدد مناسب:

```
parameters = {"n_neighbors" : range(1 , 20)}
grid_kn = GridSearchCV( estimator = knn,
                        param_grid = parameters ,
                        scoring = "accuracy",
                        cv = 5 ,
                        verbose = 1,
                        n_jobs = -1)

grid_kn.fit( x_train , y_train.ravel())
```

```
Fitting 5 folds for each of 19 candidates, totalling 95 fits
```

```
d:\Anacnda\EXE\Lib\site-packages\sklearn\model_selection\_search.py:976: UserWarning:

One or more of the test scores are non-finite: [nan nan nan nan nan nan nan nan nan nan nan
 nan]
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(n_neighbors=29), n_jobs=-1,
             param_grid={'n_neighbors': range(1, 20)}, scoring='accuracy',
             verbose=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
grid_kn.best_params_
```

```
{'n_neighbors': 1}
```

گرید سرچ نیز به ما امکان یافتن پارامتر های بهتر را میدهد که میتوانیم از آن برای یافتن مقدار کا مناسب استفاده کنیم.

مشاهده میشود که پیشنهاد گرید سرچ برای مقدار کا عدد یک میباشد.

```
x_train , x_test , y_train , y_test = train_test_split ( x , y , test_size = 0.3 , random_state = 0)
```

```
k = 1
clf5 = KNeighborsClassifier(k)
clf5.fit(x_train , y_train.ravel())
y_pred = clf5.predict(np.array(x_test))
```

مدل خود را مجدد با کا برابر یک ایجاد میکنیم.

```
1 print(classification_report(y , clf5.predict(np.array(x))))
```

```
d:\Anacnda\EXE\Lib\site-packages\sklearn\base.py:464: UserWarning:

X does not have valid feature names, but KNeighborsClassifier was fitted
```

	precision	recall	f1-score	support
1.0	0.98	0.98	0.98	2027
2.0	0.98	0.96	0.97	1322
3.0	1.00	1.00	1.00	522
4.0	0.97	0.98	0.97	1630
5.0	0.98	0.98	0.98	1928
6.0	0.95	0.95	0.95	2636
7.0	0.97	0.97	0.97	3546
accuracy			0.97	13611
macro avg	0.98	0.98	0.98	13611
weighted avg	0.97	0.97	0.97	13611

بر روی کل دیتای ایگرگ و ایگرگ پردیکت بر روی کل ایکس یک کلسفیکیشن ریپورت میگیریم که چندین معیار ارزیابی مهم را به ما نشان میدهد.

همه اسکور ها بالای 90 هستند که نشان میدهد مدل به صورت تقریبا کاملی پیش بینی کرده است

در ستون سوم مقادیر ارزیابی عدد یک را نشان میدهند که به ما میگوید مدل در ستون سوم دچار اورفیت شدگی شده است.

در نهایت تمامی اسکور های مدل های ساخته شده را مشاهده میکنیم که باید بهترین مدل را بین آن ها انتخاب کنیم.

random forest model scores : ¶

```
print(classification_report(y , clf.predict(x)))
```

	precision	recall	f1-score	support
1.0	0.98	0.97	0.97	2027
2.0	0.98	0.93	0.95	1322
3.0	1.00	1.00	1.00	522
4.0	0.95	0.97	0.96	1630
5.0	0.98	0.96	0.97	1928
6.0	0.91	0.92	0.91	2636
7.0	0.94	0.96	0.95	3546
accuracy			0.95	13611
macro avg	0.96	0.96	0.96	13611
weighted avg	0.95	0.95	0.95	13611

SVM model scores :

```
print(classification_report(y , clf2.predict(x)))
```

	precision	recall	f1-score	support
1.0	0.96	0.95	0.96	2027
2.0	0.95	0.93	0.94	1322
3.0	1.00	1.00	1.00	522
4.0	0.94	0.95	0.95	1630
5.0	0.97	0.96	0.96	1928
6.0	0.88	0.88	0.88	2636
7.0	0.92	0.93	0.92	3546
accuracy			0.93	13611
macro avg	0.95	0.94	0.94	13611
weighted avg	0.93	0.93	0.93	13611

naive bayes model scores :

```
1 print(classification_report(y , clf3.predict(x)))
```

	precision	recall	f1-score	support
1.0	0.93	0.94	0.94	2027
2.0	0.87	0.81	0.84	1322
3.0	0.99	1.00	1.00	522
4.0	0.87	0.90	0.89	1630
5.0	0.95	0.95	0.95	1928
6.0	0.82	0.87	0.84	2636
7.0	0.93	0.88	0.90	3546
accuracy			0.90	13611
macro avg	0.91	0.91	0.91	13611
weighted avg	0.90	0.90	0.90	13611

KNN model scores (K = 5 , K = 14) :

```
1 print(classification_report(y , clf4.predict(np.array(x))))
```

```
d:\Anacnda\EXE\Lib\site-packages\sklearn\base.py:464: UserWarning:

X does not have valid feature names, but KNeighborsClassifier was fitted with
```

	precision	recall	f1-score	support
1.0	0.94	0.95	0.94	2027
2.0	0.96	0.89	0.92	1322
3.0	1.00	1.00	1.00	522
4.0	0.92	0.95	0.94	1630
5.0	0.97	0.95	0.96	1928
6.0	0.87	0.89	0.88	2636
7.0	0.92	0.92	0.92	3546
accuracy			0.93	13611
macro avg	0.94	0.94	0.94	13611
weighted avg	0.93	0.93	0.93	13611

```
1 print(classification_report(y , clf5.predict(np.array(x))))
```

```
d:\Anacnda\EXE\Lib\site-packages\sklearn\base.py:464: UserWarning:

X does not have valid feature names, but KNeighborsClassifier was fitted with
```

	precision	recall	f1-score	support
1.0	0.98	0.98	0.98	2027
2.0	0.98	0.96	0.97	1322
3.0	1.00	1.00	1.00	522
4.0	0.97	0.98	0.97	1630
5.0	0.98	0.98	0.98	1928
6.0	0.95	0.95	0.95	2636
7.0	0.97	0.97	0.97	3546

مدل رندوم فارست در بین سایر مدل ها اسکور بیشتر و عملکرد بهتری را داشته است.