تمرین دوم شناسایی الگو

برای حل این تمرین ابتدا دیتاست مربوطه رو از سایت دانلود می کنیم و با استفاده از پانداس آن را لود می کنیم.

```
# Load the dataset
############################
df= pd.read_csv('./data/ObesityDataSet_raw_and_data_sinthetic.csv')
############################
```

پس از لود دیتاست ابتدا باید لیبل های دیتاست که به صورت متن هستند را با استفاده از کتابخانه مربوطه به عدد تبدیل می کنیم تا بتوانیم فرایند یادگیری را شروع می کنیم.

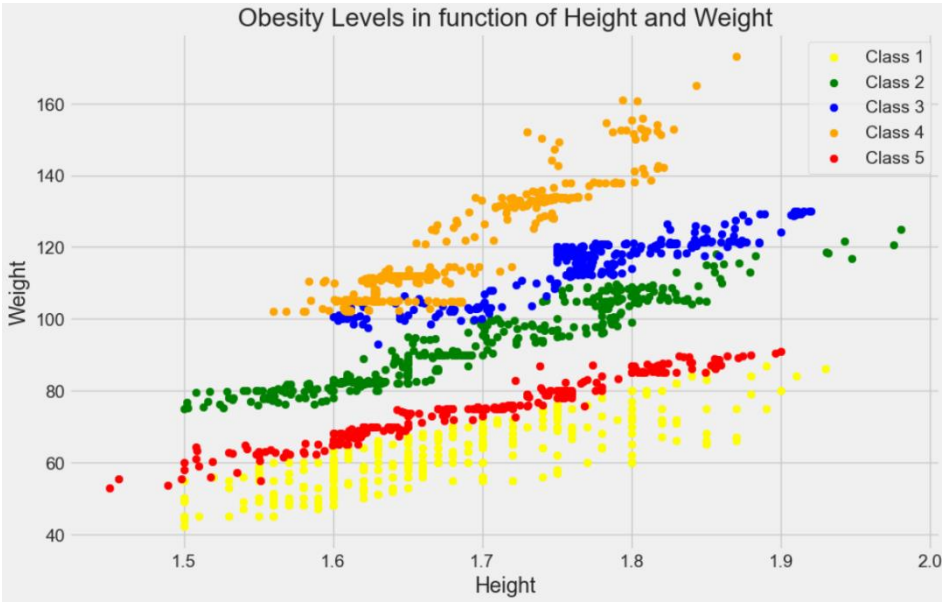
```
# Label Encoding for NObeyesdad
from sklearn.preprocessing import LabelEncoder
############################
lbl_encoder = LabelEncoder()
df['NObeyesdad'] = lbl_encoder.fit_transform(df['NObeyesdad'])
############################
df
```

```
plt.figure(figsize=(11, 7))

plt.scatter(df.Height[df.NObeyesdad==1],df.Weight[df.NObeyesdad==1],c="yellow",label='Class 1')
plt.scatter(df.Height[df.NObeyesdad==2],df.Weight[df.NObeyesdad==2],c="green",label='Class 2')
plt.scatter(df.Height[df.NObeyesdad==3],df.Weight[df.NObeyesdad==3],c="blue",label='Class 3')
plt.scatter(df.Height[df.NObeyesdad==4],df.Weight[df.NObeyesdad==4],c="orange",label='Class 4')
plt.scatter(df.Height[df.NObeyesdad==5],df.Weight[df.NObeyesdad==5],c="red",label='Class 5')

plt.title("Obesity Levels in function of Height and Weight")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.legend();
```

در ادامه یک نمودار برای درک بیشتر داده ها رسم کردیم که میزان تغییرات وزن بر اساس قد را برای هر کلاس نمایش می دهد که به شکل زیر است.



```
categorical_val = []
continuous_val = []
a = 10
for column in df.columns:
    print('==============================')
    print(f"{column} : {df[column].unique()}")
    if df[column].dtype == 'object' or df[column].nunique() < a:
        categorical_val.append(column)
    else:
        continuous_val.append(column)
```

برای جداسازی ویژگی های پیوسته و گسسته دیتاست درون حلقه فور برای هر ستون (ویژگی) باید بررسی کنیم که آیا مقادیر این ویژگی از چه نوعی است. حال برای اینکه بفهمیم از نوع داده های آن ستون و تعداد آن ها میتوانیم کنترل کنیم که از چه نوعی است. برای مثال نوع داده object میتواند گسسته بودن را تشخیص دهد و همچنین اگر تعداد مقادیر یکتای هر ستون از یک آستانه ای کمتر باشد میتواند به گسسته بودن آن اشاره کند. برای مثال ما اینجا 10 در نظر گرفتیم و بر این معنی است که اگر تعداد مقادیر یکتای یک ستون از 10 کمتر باشد آن ستون یا ویژگی گسسته است.

```
# One-hot encode categorical variables

##############################import pandas as pd
X = pd.get_dummies(X, columns=categorical_val,drop_first=True)
##############################
```

در ادامه باید ویژگی هایی که در مرحله قبل به عنوان گسسته پیدا کردیم را one-hot کنیم. برای اینکار از get_dummies استفاده میکنیم تا این ویژگی ها به یک قالب عددی تبدیل شود که برای یادگیری قابل فهم است. برای مثال یک ویژگی که درون دیتاست ما وجود دارد Gender است که مقادیر مرد و زن را دارد. پس از انجام این کار این ستون به Gender_Male تبدیل می شود که مقادیر true و false را در بر می گیرد و بر این معنا است که آیا این فرد مرد است.

نکته: اگر drop_first=True را ست نکنیم ستون Gender_Female هم ایجاد می شود که عملا نیازی به آن نیست. این کار اولین مقدار را نادیده می گیرد.

```
# Scaling features
from sklearn.preprocessing import StandardScaler

X_st = StandardScaler().fit_transform(X)
X_st
```

سپس برای یادگیری بهتر مدل باید داده ها را استاندارد کنیم یعنی میانگین آن ها برابر با صفر و انحراف از معیار آن ها برابر با یک باشد. برای این کار باید مقادیر X را به تابع مربوطه بدهیم.

```
# Separate features and target variable
X = X_st
y = df['NObeyesdad']
```

در ادامه برای شروع فرایند یادگیری ابتدا مقادیر X که استانداردسازی کردیم و y را جدا می کنیم.

```
# Stratified K-Fold Cross Validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
model = XGBClassifier(**params)
```

در ادامه یک کلاس از نوع StratifiedKFold می سازیم که برای تقسیم بندی داده ها استفاده میشود. این کلاس ابتدا داده های آموزشی را به تعداد دسته هایی با توزیع برابر برای هر کلاس تقسیم میکند.

ما اینجا تعداد دسته ها را 5 انتخاب کردیم و قبل از تقسیم داده ها آن را شافل می کند. در ادامه مقدار random_state=42 قرار می دهیم تا نتایج قابل تکرار باشد.

```
auc_scores = []
for train_index, val_index in skf.split(X, y):
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    model.fit(X_train, y_train)
    y_pred = model.predict_proba(X_val)
    auc = roc_auc_score(y_val, y_pred, multi_class="ovr")
    auc_scores.append(auc)

return sum(auc_scores) / len(auc_scores)
```

در ادامه برای محاسبه auc score باید مطابق کد بالا عمل کنیم. این کد برای محاسبه عملکرد مدل با استفاده از Cross-Validation است که ابتدا یک حلقه فور است که هر با داده هارا تقسیم می کند. سپس مقادیر X_train و X_val و سپس y_train و y_val را برای آموزش و ولیدیشن داده ها مطابق تقسیم بندی ایجاد می کنیم تا بتوانیم با استفاده از این دسته مخصوص مدل را فیت می کنیم.

سپس مقادیر y را با استفاده از مدلی که ساختیم برای داده های ولیدیشن پیش بینی می کنیم (بر اساس احتمال) و سپس مقدار auc را محاسبه می کنیم و به لیستش اضافه می کنیم.

این کار برای هر دسته بندی که اینجا 5 تا است تکرار می شود. در نهایت میانگین امتیاز همه fold ها را بر می گردانیم.

```
optimizer = BayesianOptimization(f=xgb_evaluate, pbounds=params_bounds, random_state=42, verbose=2)
optimizer.maximize(init_points=5, n_iter=15)
```

در ادامه باید از کتابخانه ی BayesianOptimization برای بهینه سازی پارامتر ها استفاده کنیم. هدف این کار پیدا کردن بهترین پارامتر هایی است که عملکرد مدل را بهتر کند.

برای این کار یک کلاس از این نوع می سازیم و مقادیر ورودی این کلاس را ست می کنیم. ابتدا تابعی که باید بهینه شود را ست می کنیم. در ادامه پارامتر هایی که باید بهینه شود را مشخص می کنیم. در ادامه مقدار random_state برای یکی شدن نتایج و verbose را برای مشاهده روند آموزش برابر 2 قرار می دهیم.

در خط بعدی عملیات بهینه سازی شروع میشود که 2 پارامتر می گیرد. اولی مقدار init_points است که تعداد نقاط تصادفی اولیه برای شروع فرآیند بهینه‌سازی را مشخص می‌کند که اینجا برابر 5 است. این کار برای این است که اطلاعات اولیه ای از مدل بدست بیارد.

دومی n_iter است که برابر تعداد تکرار های اصلی برای بهینه سازی است تا بهترین پارامتر ها را پیدا کند.

```
# Output the best parameters and corresponding AUC score
best_params = optimizer.max['params']
best_auc = optimizer.max['target']
print("Best Parameters found with Bayesian optimization:", best_params)
print("Best AUC Score:", best_auc)
```

پس از پایان حلقه فور نوبت به چاپ بهترین پارامتر هایی است که توسط بهینه ساز بیز تعیین شده است می رسد. در ادامه بهترین امتیاز AUC که مدل با پارامترهای بهینه به آن دست یافته را نمایش می دهیم. همه این اطلاعات در optimizer.max ذخیره شده است.

نتیجه بهترین پارامتر ها:

iter	target	colsam...	gamma	learni...	max_depth	min_ch...	n_esti...	reg_alpha	reg_la...	subsample
1	0.9699	0.6873	4.754	0.3687	7.191	0.7801	831.2	5.808	86.62	0.8006
2	0.9868	0.854	0.1029	0.4853	8.827	1.062	836.4	18.34	30.42	0.7624
3	0.9075	0.716	1.456	0.3098	3.976	1.461	873.3	45.61	78.52	0.5998
4	0.8657	0.7571	2.962	0.03276	7.253	0.8526	813.0	94.89	96.56	0.9042
5	0.9918	0.6523	0.4884	0.3453	6.081	0.6102	899.0	3.439	90.93	0.6294
6	0.971	0.8891	4.134	0.4941	8.187	2.214	830.5	4.787	88.63	0.7233
7	0.9884	0.5613	3.278	0.3766	7.707	0.03822	906.1	1.293	6.456	0.5363
8	0.9788	0.601	2.805	0.2942	6.434	1.228	984.0	0.4279	97.48	0.597
9	0.997	1.0	0.0	0.01	10.0	0.0	1e+03	0.0	0.0	1.0
10	0.9024	0.8205	1.153	0.2013	9.129	2.277	997.4	89.49	1.527	0.7598
11	0.9929	0.5378	1.491	0.2433	3.061	4.173	802.0	3.132	0.8604	0.7549
12	0.9946	0.944	1.224	0.2354	7.679	0.8235	960.3	0.1804	40.82	0.9765
13	0.9905	0.5417	4.346	0.1774	9.048	4.494	848.2	1.676	1.463	0.7335
14	0.9002	0.988	3.68	0.452	8.852	2.022	801.2	70.17	0.1649	0.747
15	0.9981	0.5348	0.02742	0.07461	5.168	2.95	800.8	0.364	42.73	0.9941
16	0.9841	0.961	4.312	0.2899	4.259	4.719	870.8	0.6453	47.54	0.9209
17	0.9957	0.9157	1.85	0.1823	5.377	0.7485	956.8	0.6497	0.1336	0.9992
18	0.9958	0.9788	0.4192	0.2522	9.091	2.851	935.4	0.08462	96.78	0.7557
19	0.9901	0.8351	2.096	0.1545	6.438	4.822	828.1	0.03636	29.24	0.5077
20	0.7425	0.6672	1.506	0.4458	8.689	2.847	997.3	96.87	98.17	0.573

Best Parameters found with Bayesian optimization: {'colsample_bytree': 0.5347694405193053, 'gamma': 0.027418875203317206, 'learning_rate': 0.0746092508274788, 'max_depth': 5.168155135164 61, 'min_child_weight': 2.950301965234802, 'n_estimators': 800.7933567382288, 'reg_alpha': 0.3639733881765084, 'reg_lambda': 42.73271151915936, 'subsample': 0.9940896111007402}
Best AUC Score: 0.9981353164942359

```
# Train the final model with the best parameters on the entire dataset
best_params['max_depth'] = int(best_params['max_depth'])
best_params['n_estimators'] = int(best_params['n_estimators'])
best_model = XGBClassifier(**best_params)
```

در ادامه برای ساخت مدل با بهتری پارامتر ها ابتدا 2 تا از پارامتر هایی بدست آمده اند را به int تبدیل می کنیم و سپس مدل را با بهترین پارامتر هایی که بدست اوردیم ایجاد می کنیم.

```
# Split the data into training and test sets (20% for test set is recommended)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

در ادامه داده های دیتاست رو با random_state 42 به دو بخش train و test تقسیم می کنیم.

```
best_model.fit(X_train, y_train)

y_pred_train = best_model.predict(X_train)
y_pred_test = best_model.predict(X_test)
```

سپس نوبت به فیت کردن مدل می رسد. پس از آن y های آموزشی و تست را با استفاده از مدل پیش بینی میکنیم تا بتوانیم مدل رو ارزیابی کنیم.

```
# Calculate Metrics for Train Data
accuracy_train = accuracy_score(y_train, y_pred_train)
recall_train = recall_score(y_train, y_pred_train, average='macro')
precision_train = precision_score(y_train, y_pred_train, average='macro')
```

برای محاسبه معیار های ارزیابی برای داده های آموزشی میزان accuracy را با کتابخانه مربوطه محاسبه میکنیم. در ادامه برای دو معیار precision و recall با میانگین ماکرو استفاده می کنیم تا میانگین این دو معیار برای تمام کلاس ها محاسبه شود.

```
# Calculate Metrics for Test Data
accuracy_test = accuracy_score(y_test, y_pred_test)
recall_test = recall_score(y_test, y_pred_test, average='macro')
precision_test = precision_score(y_test, y_pred_test, average='macro')
```

حال همین معیار ها را برای داده های تست محاسبه می کنیم.

```
# Multi-Class Specificity (per-class true negatives rate approximation)
conf_matrix_test = confusion_matrix(y_test, y_pred_test)

specificity_per_class = []
for i in range(conf_matrix_test.shape[0]):
    tn = conf_matrix_test.sum() - (conf_matrix_test[i, :].sum() + conf_matrix_test[:, i].sum() - conf_matrix_test[i, i])
    fp = conf_matrix_test[:, i].sum() - conf_matrix_test[i, i]
    specificity = tn / (tn + fp)
    specificity_per_class.append(specificity)
specificity_test = np.mean(specificity_per_class)
```

در ادامه ماتریس آشفتگی (Confusion Matrix) برای داده های تست را حساب کردیم. پس از آن specificity_per_class را برای ذخیره‌سازی مقدار Specificity برای هر کلاس ایجاد کردیم.

برای محاسبه Specificity باید tn و fp را برای هر کلاس به صورت جداگانه محاسبه کنیم و طبق فرمول میزان Specificity را برای هر کلاس بدست بیاوریم. سپس مقادیر Specificity هر کلاس را به متغیر مربوطه اضافه می کنیم. در نهایت میانگین Specificity را به عنوان جواب نهایی بر می گردانیم.

در ادامه نوبت به چاپ معیار های ارزیابی می رسد که خروجی به صورت زیر است:

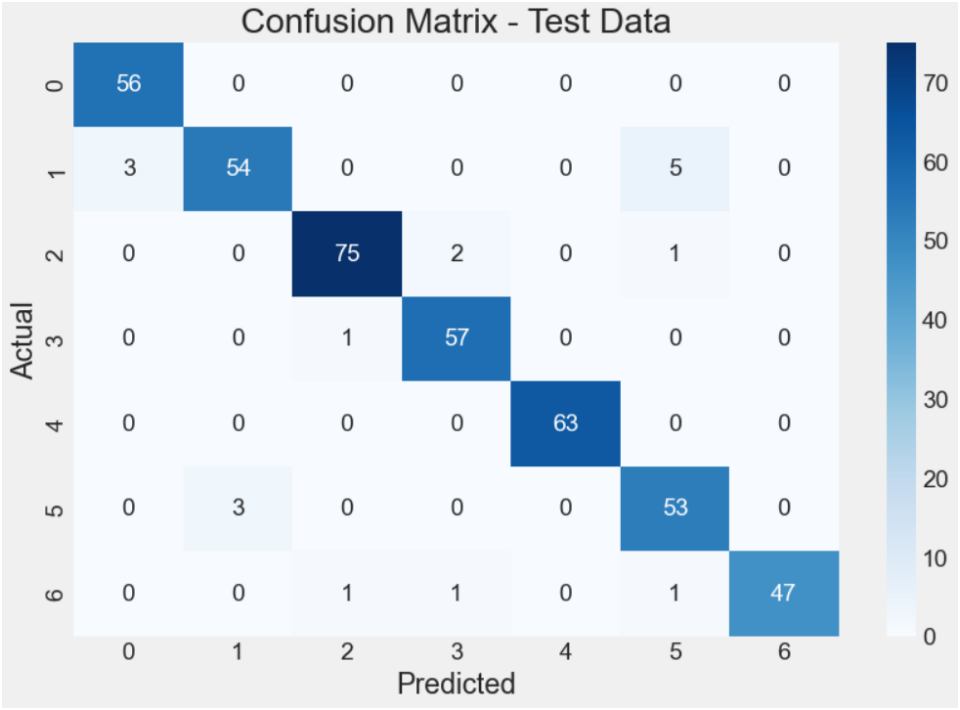
```python
# Print the results
############################
print("Train Metrics:")
print("Accuracy:", accuracy_train)
print("Recall:", recall_train)
print("Precision:", precision_train)
print()
print("Test Metrics:")
print("Accuracy:", accuracy_test)
print("Recall:", recall_test)
print("Precision:", precision_test)
print("Specificity:", specificity_test)
############################
```

```
Train Metrics:
Accuracy: 0.9970379146919431
Recall: 0.9969241645965783
Precision: 0.9969145620461409

Test Metrics:
Accuracy: 0.9574468085106383
Recall: 0.9573847707988816
Precision: 0.9576971815406886
Specificity: 0.9929179306796835
```

```python
# Confusion Matrix Heatmap for Test Data
############################
plt.figure(figsize=(9, 6))
sns.heatmap(conf_matrix_test, annot=True, cmap='Blues', cbar=True, annot_kws={'size': 14})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Test Data')
plt.show()
############################
```

در ادامه برای نمایش ماتریس آشفتگی بر روی هیت مپ از کد بالا استفاده می کنیم که به صورت زیر است:



خب به سراغ الگوریتم Decision Tree میرویم.

```
# Stratified K-Fold Cross Validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

همانند الگوریتم بوستینگ که بالا توضیح دادیم از cross-validation برای این الگوریتم نیز استفاده می کنیم.

```
auc_scores = []
for train_index, val_index in skf.split(X, y):
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    model = DecisionTreeClassifier(**params)
    model.fit(X_train, y_train)
    y_pred = model.predict_proba(X_val)

    if y_pred.shape[1] > 1:
        auc = roc_auc_score(y_val, y_pred, multi_class='ovr', average='macro')
    else:
        auc = roc_auc_score(y_val, y_pred[:, 1])

    auc_scores.append(auc)

return sum(auc_scores) / len(auc_scores)
```

مقدار auc را برای هر کلاس محاسبه می کنیم و میانگین می گیریم.

```
optimizer = BayesianOptimization(f=dt_evaluate, pbounds=params_bounds, random_state=42, verbose=2)
optimizer.maximize(init_points=5, n_iter=15)
```

بهینه ساز بیز را برای تابع dt_evaluate ایجاد می کنیم تا بهترین پارامتر ها برای این الگوریتم نیز پیدا کنیم.

```
best_params = optimizer.max['params']
best_auc = optimizer.max['target']
print("Best Parameters found with Bayesian optimization:", best_params)
print("Best AUC Score:", best_auc)
```

پس از پایان کار بهینه ساز بهترین پارامتر ها را چاپ می کنیم:

```
Best Parameters found with Bayesian optimization: {'criterion': 0.6534516421196439, 'max_depth': 13.779477645012141, 'max_features': 0.977115380106919,
'min_samples_leaf': 16.107233381154646, 'min_samples_split': 7.500246156570386}
Best AUC Score: 0.9843151114170364
```

```
criterion_map = {0: 'gini', 1: 'entropy', 2: 'log_loss'}
best_params['max_depth'] = int(best_params['max_depth'])
best_params['min_samples_leaf'] = int(best_params['min_samples_leaf'])
best_params['min_samples_split'] = int(best_params['min_samples_split'])
best_params['criterion'] = criterion_map[int(best_params['criterion'])]
best_model = DecisionTreeClassifier(**best_params)
```

قبل از ایجاد بهترین مدل برای این الگوریتم ابتدا مقادیر بالا را به عدد صحیح تبدیل می کنیم تا به مشکل نخوریم.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

best_model.fit(X_train, y_train)

y_pred_train = best_model.predict(X_train)
y_pred_test = best_model.predict(X_test)
# Calculate Metrics for Train Data
accuracy_train = accuracy_score(y_train, y_pred_train)
recall_train = recall_score(y_train, y_pred_train, average='macro')
precision_train = precision_score(y_train, y_pred_train, average='macro')

# Calculate Metrics for Test Data
accuracy_test = accuracy_score(y_test, y_pred_test)
recall_test = recall_score(y_test, y_pred_test, average='macro')
precision_test = precision_score(y_test, y_pred_test, average='macro')
```

مقادیر train و test را جدا کرده و مدل را فیت می کنیم.

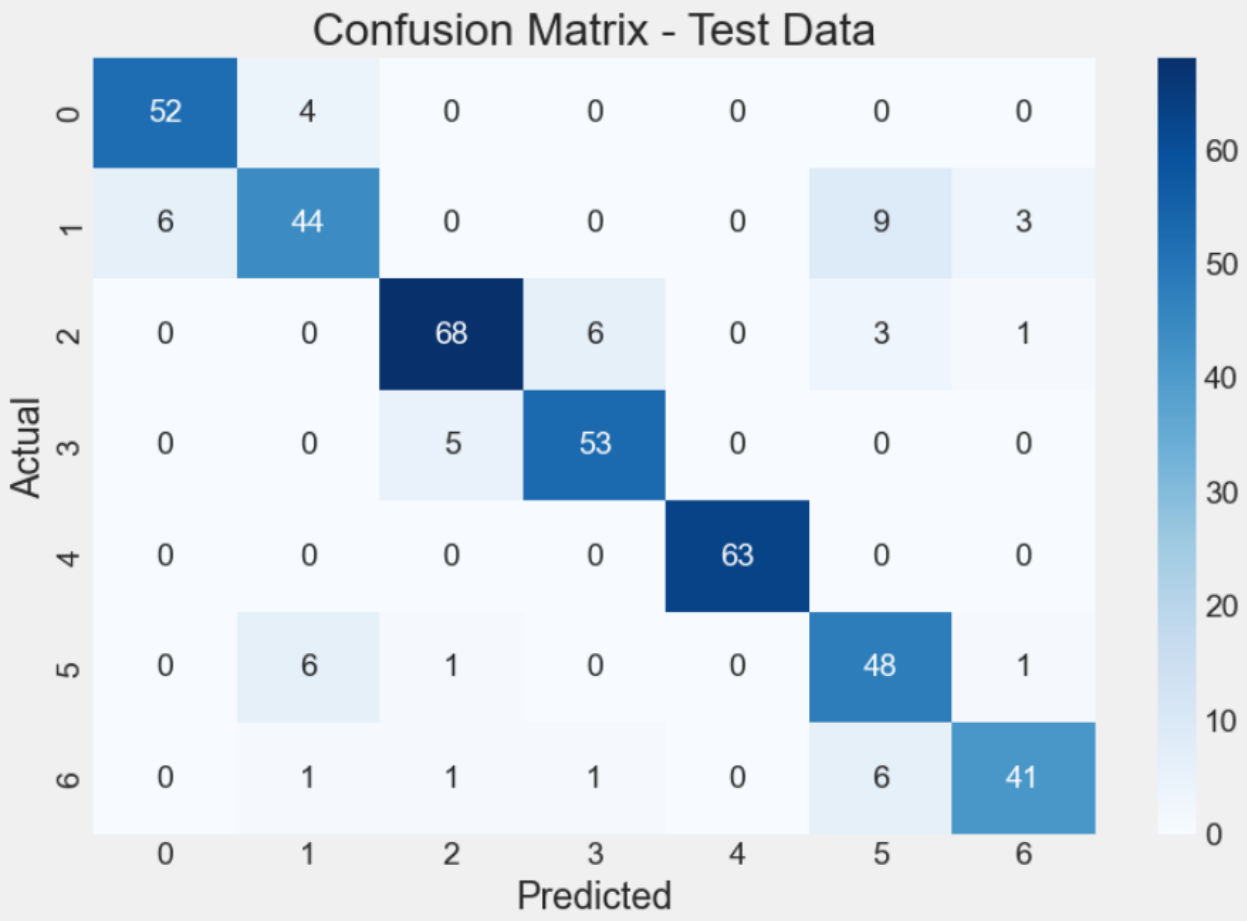
مقادیر y_pred_train و y_pred_test را با استفاده از مدل پیش بینی می کنیم.

سپس معیار های ارزیابی را محاسبه می کنیم و در نهایت آن ها را چاپ می کنیم:

```
Train Metrics:
Accuracy: 0.8969194312796208
Recall: 0.8957182598551087
Precision: 0.8965037196324976

Test Metrics:
Accuracy: 0.8723404255319149
Recall: 0.871568525758896
Precision: 0.8721612570338207
Specificity: 0.9787516039290313
```

در اینجا مشاهده شد که در الگوریتم Decision Tree هم دقت مدل برای داده های آموزشی و هم برای داده های تست از الگوریتم XGBoost بسیار کمتر است.



محمد حقیقت – 403722042

برای رفع برخی ایرادات و ابهامات از Chatgpt استفاده شده است.