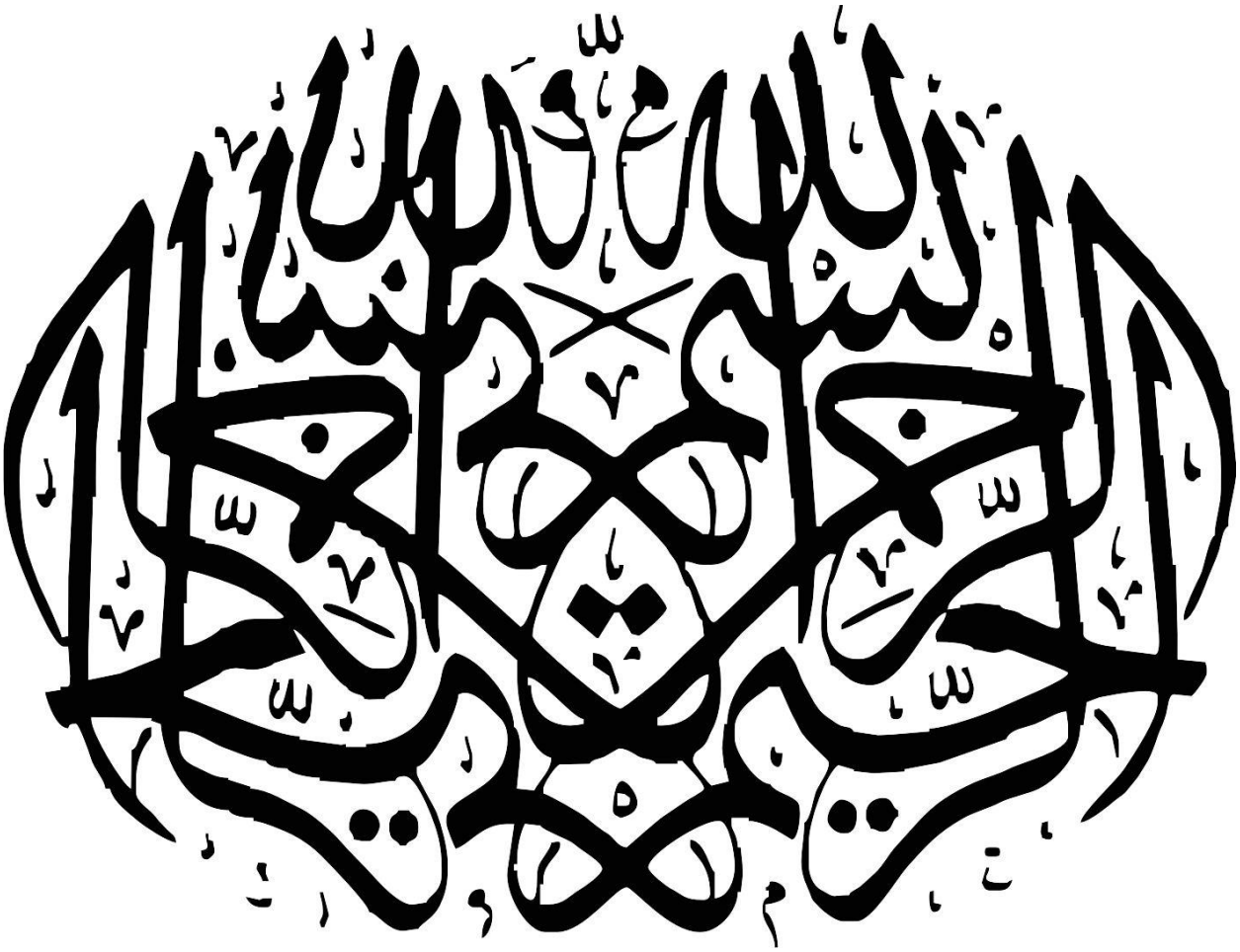
دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش تکلیف چهارم درس یادگیری ماشین کاربردی

استاد درس: دکتر ناظرفرد

تهیه کننده: سید نیما محمودیان

شماره دانشجویی: ۴۰۲۱۲۵۰۰۵



سید نیما محمودیان ۴۰۲۱۲۵۰۰۵

فهرست مطالب

فهرست شکل‌ها

۲-پاسخ سوال دوم

۲-۱-آماده‌سازی داده‌ها

کد شکل (۱)، کتابخانه‌های مورد نیاز برای پردازش و تحلیل داده‌ها، از جمله pandas، numpy و ابزارهای مختلف از scikit-learn را وارد می‌کند.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

```

شکل ۱: وارد کردن کتابخانه‌های مورد نیاز

داده‌ها از فایل crash.csv بارگذاری شده و در DataFrame به نام df ذخیره می‌شوند. (شکل ۲)

```python
df= pd.read_csv("crash.csv")
df
```

شکل ۲: بارگذاری داده‌ها

سپس ساختار داده‌ها و اطلاعات اولیه آن با استفاده از df.info() نمایش داده می‌شود تا نوع و تعداد مقادیر گمشده بررسی شود. (شکل ۳)

```python
df.info()
```

شکل ۳: بررسی ساختار اولیه داده‌ها

شکل (۴) خروجی این متد را نمایش می‌دهد.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    891 non-null    int64
 1   PassengerId   891 non-null    int64
 2   Survived      891 non-null    object
 3   Class         891 non-null    object
 4   Name          891 non-null    object
 5   Sex           891 non-null    object
 6   Age           714 non-null    float64
 7   Ticket Price  891 non-null    float64
 8   Safety        889 non-null    float64
dtypes: float64(3), int64(2), object(4)
memory usage: 62.8+ KB
```

شکل ۴: خروجی متد ()info.

سپس تعداد مقادیر گمشده در هر ستون داده‌ها با استفاده از ()df.isna().sum محاسبه می‌شود. شکل (۵) خروجی این متد را نمایش می‌دهد.

```
Unnamed: 0        0
PassengerId       0
Survived          0
Class             0
Name              0
Sex               0
Age             177
Ticket Price      0
Safety            2
dtype: int64
```

شکل ۵: تعداد مقادیر گمشده هر ستون

ستون‌های غیر ضروری مانند " Unnamed: ۰"، "PassengerId" و "Name" از دیتافریم حذف می‌شوند تا فقط ستون‌های مهم باقی بمانند. این کار با متد ()drop. انجام می‌شود. شکل (۶) نحوه انجام این کار را نمایش می‌دهد.

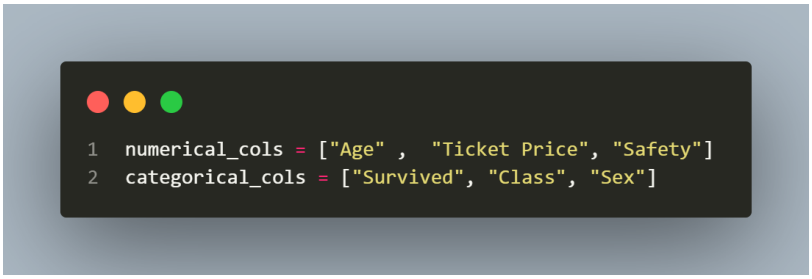
```
1  df.drop(["Unnamed: 0", "PassengerId", "Name"] , axis=1, inplace=True)
```

شکل ۶: حذف ستون‌های غیر ضروری

سپس توزیع مقادیر مختلف در ستون‌های "Class" "Survived" و "Sex" با استفاده از value_counts بررسی می‌شود. در ادامه تعداد مقادیر صفر و "؟" در هر ستون محاسبه و نمایش داده می‌شود. مشاهده می‌شود که ۱۵ مقدار ستون ticke price برابر صفر است. ممکن است این

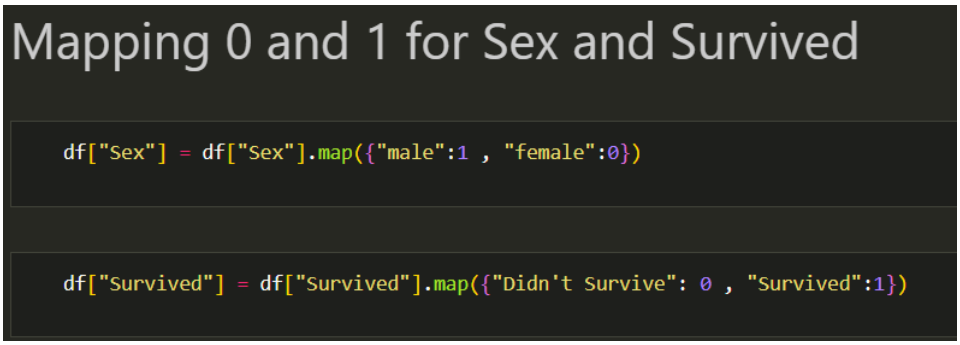
رکوردها یا گم شده باشند (بلیط رایگان کمی غیر منطقی است) یا مربوط به خدمه‌ی هواپیما باشند. در هر صورت، برای اطمینان از یکدست بودن نتایج، این رکوردها را حذف می‌کنیم. همچنین مشاهده می‌شود که هیچ علامت سوالی در دیتاست موجود نمی‌باشد.

در ادامه، ستون‌های عددی و دسته‌ای به صورت جداگانه شناسایی و دسته‌بندی می‌شوند. شکل (۷) این ستون‌ها را نمایش می‌دهد.



شکل ۷: دسته بندی ستون‌های عددی و دسته‌ای

در ادامه مقادیر دسته‌ای در ستون‌های "Sex" و "Survived" به مقادیر عددی تبدیل می‌شوند تا برای مدل‌سازی مناسب باشند. شکل (۸) نحوه انجام این کار را با متد map() نمایش می‌دهد.



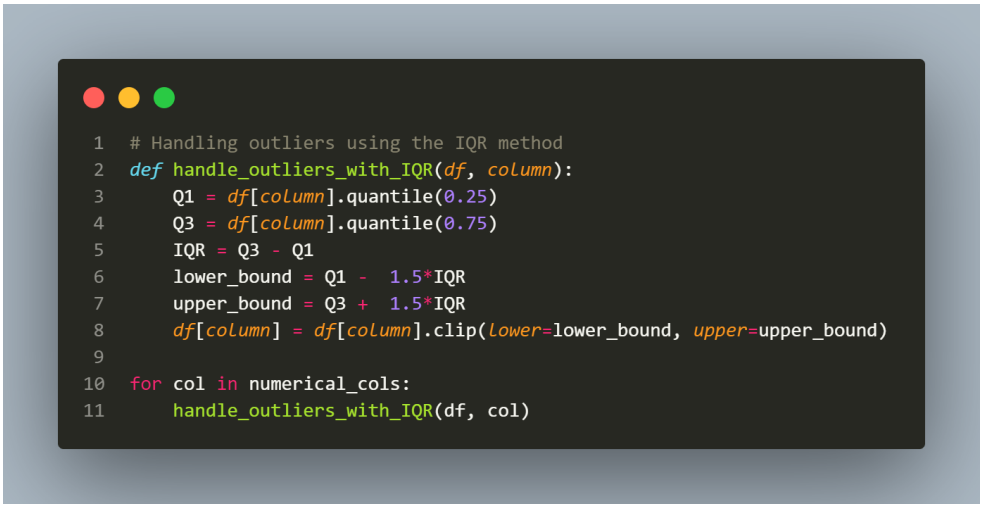
شکل ۸: نگاشت مقادیر دسته‌ای در ستون‌های Sex و Survived

سپس مقادیر دسته‌ای در ستون "Class" با استفاده از OneHotEncoder کدگذاری شده و به دیتافریم اصلی اضافه می‌شوند. البته می‌توانستیم از کدگذاری سلسله‌مراتبی هم برای کد گذاری کلاس‌های پرواز استفاده کنیم. اما از آنجایی که کاردینالیتی این ستون برابر ۳ است، و تعداد ستون‌ها در مجموع کم است، اضافه کردن دو ستون دیگر به ستون‌های دیتافریم مشکلی ایجاد نمی‌کند. شکل (۹) نحوه انجام این کار را نمایش می‌دهد.

```python
# Create an instance of the OneHotEncoder
encoder = OneHotEncoder()

# Fit the encoder on the "Class" column and transform it
class_encoded = encoder.fit_transform(df[['Class']])

# Convert the encoded result to an array and create a DataFrame
class_encoded_df = pd.DataFrame(class_encoded.toarray(), columns=encoder.get_feature_names_out(['Class']))

# Concatenate the encoded DataFrame with the original DataFrame
df = pd.concat([df.drop(columns=['Class']), class_encoded_df], axis=1)
```

شکل ۹: OneHotEncoding

در ادامه یک تابع برای مدیریت مقادیر پرت با استفاده از روش IQR (Interquartile Range) تعریف و برای ستون‌های عددی اعمال می‌شود. شکل (۱۰) این تابع را نمایش می‌دهد.



```python
# Handling outliers using the IQR method
def handle_outliers_with_IQR(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 -  1.5*IQR
    upper_bound = Q3 +  1.5*IQR
    df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)

for col in numerical_cols:
    handle_outliers_with_IQR(df, col)
```

شکل ۱۰: حذف داده‌های پرت با استفاده از روش دامنه میان چارکی

با بررسی اتریبیوت shape. متوجه می‌شویم که هیچ مقداری حذف نشده است. درنتیجه داده پرتی در دیتاست وجود ندارد.

۲-۲-مدیریت مقادیر گمشده

حال به مدیریت مقادیر گمشده می‌پردازیم. مقادیر گمشده در ستون‌های مشخص با استفاده از تکنیک‌های مختلف جایگزین شدند: برای این منظور از دو روش استفاده شد:

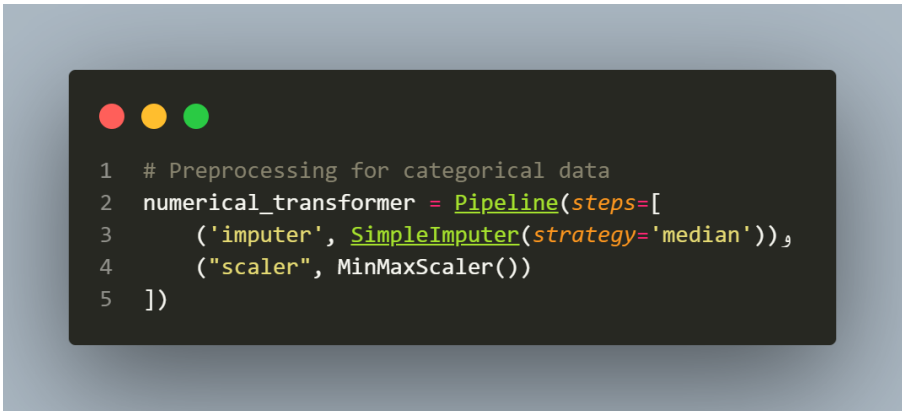
- استفاده از SimpleImputer برای جایگزینی مقادیر گمشده با میانه.
- استفاده از KNNImputer برای جایگزینی مقادیر گمشده با استفاده از مقادیر مشابه در نزدیک‌ترین همسایگان.

حال برای راحتی اجرای پیش پردازش، از کلاس Pipeline از کتابخانه scikitlearn استفاده می‌کنیم. این کلاس، مجموعه‌ای از اعمال را به شکل پیاپی بر روی دیتاست اجرا می‌کند.

پایپ‌لاین مورد استفاده به دو قسمت تقسیم می‌شود: پیش‌پردازش داده‌های عددی، آموزش مدل و تست و ارزیابی مدل.

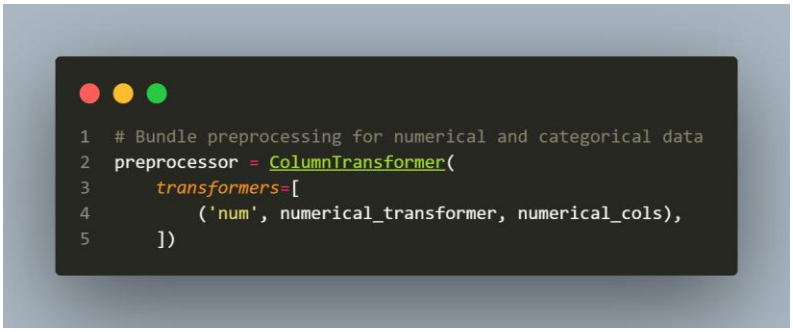
پیش‌پردازش داده‌های دسته‌ای خود شامل یک پایپ‌لاین کوچک‌تر است که شامل اجرای دو column transformer است. ترتیب وارد کردن ترنسفورمرها در شیِ پایپ‌لاین، نماینده ترتیب اجرا نیز می‌باشد. اولین ترنسفورمر SimpleImputer است که کار جایگزینی np.nan را با میانه به عهده دارد. دومین ترنسفورمر روی هر ستون MinMaxScaler را اجرا می‌کند.

نحوه کارکرد این ترنسفورمرها به این صورت است که یک به یک ستون‌ها را دریافت می‌کنند، مقادیر خالی را با میانه جایگذاری می‌کند و سپس ستون را با روش مین-مکس مقیاس می‌کند. شکل (۱۱) پایپ‌لاین مربوط به پیش‌پردازش داده‌های عددی را نمایش می‌دهد.



شکل ۱۱: پایپ‌لاین پیش پردازش داده‌های عددی

حال با استفاده از کلاس ColumnTransformer() یک شی به نام preprocessor ایجاد می‌کنیم. در این کلاس، لیستی از تاپل‌ها را به عنوان آرگومان پاس می‌دهیم به طوری که عضو اول تاپل، نام ترنسفورمر، عضو دوم عمل یا اعمالی است که باید روی ستون‌ها انجام شود و عضو سوم لیستی شامل نام ستون‌های هدف است که کارهای تعریف شده بر روی آنها اعمال می‌شود. شکل (۱۲) نحوه تعریف کلاس ColumnTransformer() را نمایش می‌دهد.



شکل ۱۲: نحوه تعریف کلاس ColumnTransformer()

در قدم بعدی، مدل را تعریف می‌کنیم و پایپ‌لاین اصلی را ایجاد می‌کنیم که شامل دو مرحله‌ی پیش‌پردازش و آموزش است. شکل (۱۳) این بخش را نمایش می‌دهد.

```python
model = RandomForestClassifier(n_estimators=100, random_state=0)


my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('model', model)
                             ])
```

شکل ۱۳: تعریف مدل و پایپ‌لاین اصلی

از آنجایی که این مدل صرفاً نقش یک معیار برای ارزیابی عملکرد روش‌ها را دارد، احتیاجی به fine tune کردن آن نیست چون که این مدل قرار نیست در ادامه کار پیش‌بینی داده‌های جدید را انجام دهد.

در قدم بعدی با استفاده از متد fit() و با پاس دادن داده‌های آموزشی به عنوان آرگومان، ابتدا داده‌ها را طبق توضیحات گفته شده پیش‌پردازش می‌کنیم و سپس مدل را آموزش می‌دهیم. در ادامه با استفاده از متد predict و پاس دادن داده‌های معیارسنجی، ابتدا این داده‌ها پیش‌پردازش می‌شوند و سپس مدل آموزش داده شده روی داده‌های آموزشی بر روی این داده‌ها معیارسنجی می‌شود. مشاهده می‌شود استفاده از میانه مقدار AUC را برابر با ۰.۶۳۶۳ نتیجه می‌دهد.

مشابه همین کار را مجدداً تکرار می‌کنیم، با این تفاوت که به جای SimpleImputer از KNNImputer استفاده می‌کنیم. در این حالت AUC برابر ۰.۶۳۱۵ به دست می‌آید.

مشاهده می‌شود میانه عملکرد بهتری دارد. پس مقادیر گمشده را با میانه جایگذاری می‌کنیم. شکل (۱۳) نحوه انجام این کار را نمایش می‌دهد.

```python
# Create a SimpleImputer instance
imputer = SimpleImputer(strategy='median')

# Fit and transform your data with the imputer
imputed_array = imputer.fit_transform(df)
df = pd.DataFrame(imputed_array, columns=df.columns)
```

شکل ۱۴: جایگذاری مقادیر خالی با میانه

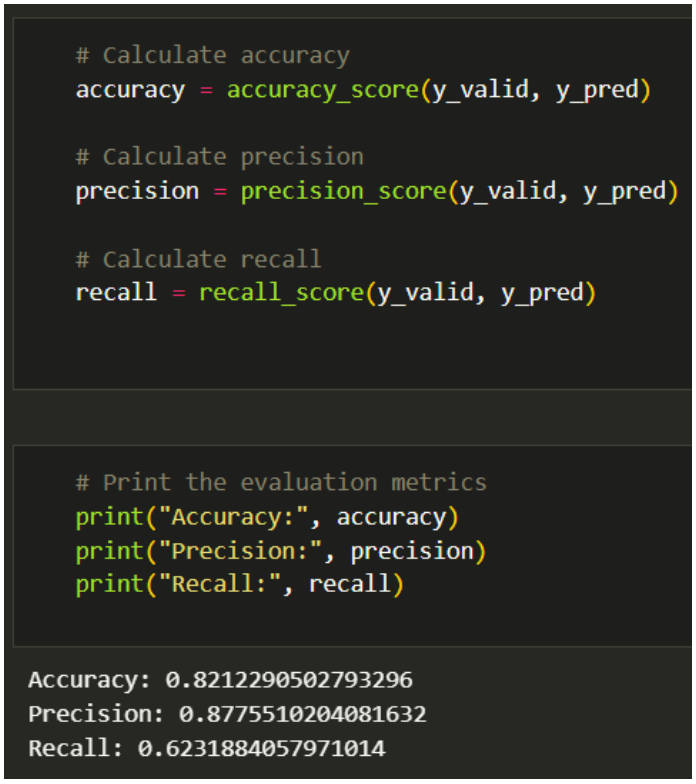
۲-۳-آموزش مدل ماشین بردار پشتیبان

ابتدا داده‌ها به مجموعه‌های آموزشی و ارزیابی تقسیم می‌شوند تا مدل بتواند با داده‌های آموزشی آموزش ببیند و با داده‌های ارزیابی تست شود. سپس مشابه قبل، یک پایپ‌لاین برای آموزش مدل ایجاد می‌شود. شکل(۱۵) پایپ‌لاین ایجاد شده را نمایش می‌دهد.

```
my_pipeline = Pipeline(steps=[('preprocessor', MinMaxScaler()),
                              ('model', SVC())
                             ])
```

شکل ۱۵: پایپ‌لاین آموزش SVM

سپس مشابه بخش قبلی، مدل را آموزش می‌دهیم و تست می‌کنیم و معیارهای ارزیابی را محاسبه می‌کنیم. شکل (۱۶) محاسبه معیارهای ارزیابی و نتیجه آنها را نمایش می‌دهد.



```
# Calculate accuracy
accuracy = accuracy_score(y_valid, y_pred)

# Calculate precision
precision = precision_score(y_valid, y_pred)

# Calculate recall
recall = recall_score(y_valid, y_pred)
```

```
# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.8212290502793296
Precision: 0.8775510204081632
Recall: 0.6231884057971014
```

شکل ۱۶: محاسبه و نمایش مقادیر معیارهای ارزیابی

۲-۴-بهینه‌سازی هایپرپارامترها

شبکه‌ای از پارامترها برای جستجوی شبکه‌ای ایجاد می‌شود و یک پایپ‌لاین جدید، با استفاده از GridSearchCV بهینه‌سازی می‌شود. شکل (۱۷) پارامترها و شکل (۱۸) جست و جوی شبکه‌ای را نمایش می‌دهد.

```python
param_grid = {
    'model__C': [0.1, 1, 10, 100],
    'model__kernel': ['linear', 'rbf', 'poly'],
    'model__degree': [2, 3,4]
}
```

شکل ۱۷: پارامترهای مدل

```python
# Initialize GridSearchCV with the SVM classifier and parameter grid
grid_search = GridSearchCV(my_pipeline, param_grid, cv=5, scoring='accuracy')
# Perform grid search with cross-validation
grid_search.fit(X_train, y_train)
# Get the best hyperparameters
best_params = grid_search.best_params_
```

شکل ۱۸: انجام جست و جوی شبکه‌ای

سپس مقادیر بهینه را نمایش می‌دهیم. شکل (۱۹) هایپرپارامترهای بهینه را نمایش می‌دهد.

```
best_params
```

```
{'model__C': 10, 'model__degree': 3, 'model__kernel': 'poly'}
```

شکل ۱۹: هایپرپارامترهای بهینه

سپس مشابه بخش‌های قبلی با استفاده از یک پایپ‌لاین مدل با پارامترهای بهینه را آموزش می‌دهیم. پایپ‌لاین به روز شده در شکل (۲۰) نمایش داده شده است و نتایج مربوطه در شکل (۲۱) قرار گرفته.

```python
my_pipeline = Pipeline(steps=[('preprocessor', MinMaxScaler()),
                              ('model', SVC(C = 1, degree=3 , kernel="poly"))
                             ])
```

شکل ۲۰: پایپ‌لاین با هایپرپارامترهای بهینه

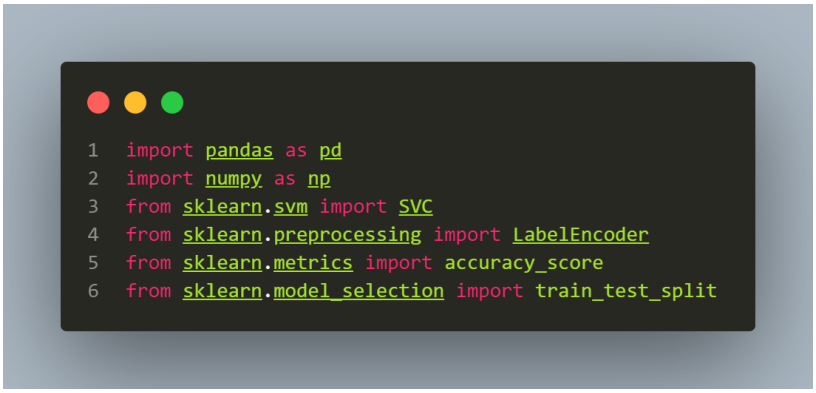
```
# Calculate accuracy
accuracy = accuracy_score(y_valid, y_pred)
# Calculate precision
precision = precision_score(y_valid, y_pred)
# Calculate recall
recall = recall_score(y_valid, y_pred)


# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.7988826815642458
Precision: 0.7796610169491526
Recall: 0.6666666666666666
```

شکل ۲۱: نتایج بهینه‌سازی پارامترها

۳-پاسخ سوال سوم

کتابخانه‌های مورد نیاز برای تحلیل داده‌ها و مدل‌سازی وارد می‌شوند. این شامل pandas، numpy و ابزارهای مختلف از scikit-learn برای پیش‌پردازش داده‌ها، تقسیم داده‌ها، ساخت مدل‌ها و ارزیابی مدل‌ها است. شکل (۲۲) این کتابخانه‌ها را نمایش می‌دهد.



شکل ۲۲: ایمپورت کردن کتابخانه‌های مورد نیاز

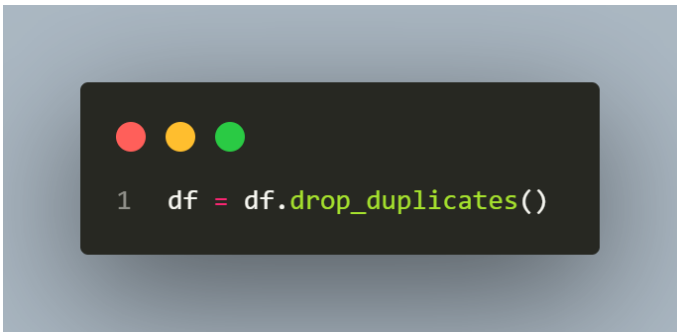
پس از فراخوانی دیتاست، با استفاده از متد values_counts() تعداد هر یک از کلاس‌ها را بررسی می‌کنیم. شکل (۲۳) خروجی این متد را نمایش می‌دهد.

```
df["Class"].value_counts()
```

```
Class
'0'    284315
'1'       492
Name: count, dtype: int64
```

شکل ۲۳: بررسی تعداد هر یک از کلاس‌ها

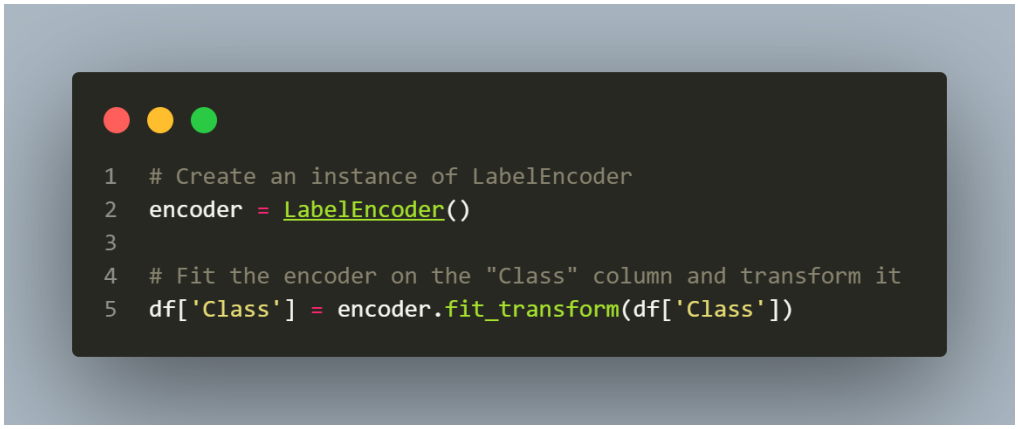
سپس داده‌های تکراری شناسایی و حذف می‌شوند تا کیفیت داده‌ها بهبود یابد. شکل (۲۴) نحوه انجام این کار را نمایش می‌دهد.



```
df = df.drop_duplicates()
```

شکل ۲۴: حذف ردیف‌های تکراری

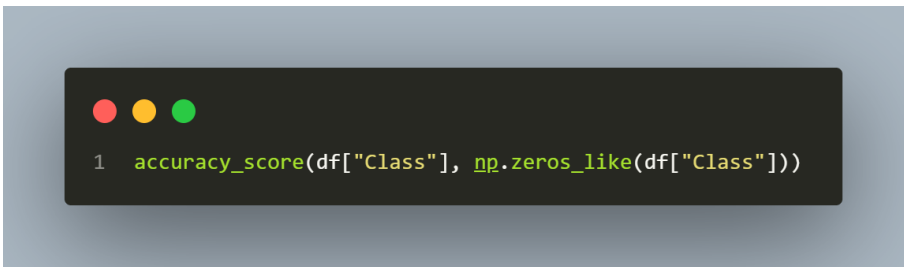
سپس ستون "Time" را با متد drop() حذف می‌کنیم. در ادامه با استفاده از labelEncoder() ستون هدف را به صفر و یک انکود می‌کنیم. شکل (۲۵) نحوه انجام این کار را نمایش می‌دهد.



```
# Create an instance of LabelEncoder
encoder = LabelEncoder()

# Fit the encoder on the "Class" column and transform it
df['Class'] = encoder.fit_transform(df['Class'])
```

شکل ۲۵: انکود کردن ستون هدف

در مرحله بعد، مدل اولیه‌ای با استفاده از آرایه‌ای شامل صفر برای ارزیابی دقت ایجاد می‌شود. این مرحله به عنوان پایه‌ای برای مقایسه با مدل‌های پیشرفته‌تر استفاده می‌شود.



```
accuracy_score(df["Class"], np.zeros_like(df["Class"]))
```

شکل ۲۶: ارزیابی دقت با آرایه‌ای متشکل از صفر

دقت پایه برابر ۰.۹۹۸۳ است. همچنین یک گزارش عملکرد از این حال پایه می‌گیریم. شکل (۲۷) این گزارش را نمایش می‌أهد.

```
report = classification_report(df["Class"], np.zeros_like(df["Class"]))
print("Classification Report:\n", report)
✓ 0.2s

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    283253
           1       0.00      0.00      0.00       473

    accuracy                           1.00    283726
   macro avg       0.50      0.50      0.50    283726
weighted avg       1.00      1.00      1.00    283726
```

شکل ۲۷:گزارش عملکرد در حالت پایه

پس از تقسیم کردن دیتاست به دیتاست آموزشی و آزمایشی، برای آموزش بهتر مدل، باید تعداد کلاس‌ها را در دیتاست بالانس کنیم. برای این منظور، از under sampling بهره می‌گیریم. روش مورد نظر برای انجام این کار Tomek links است. Tomek Links جفت‌هایی از نمونه‌ها را از کلاس‌های مختلف شناسایی می‌کند که نزدیک‌ترین همسایه‌های یکدیگر هستند. حذف نمونه های کلاس اکثریت در این جفت‌ها به تمیز کردن مرزهای کلاس کمک می کند. سپس مدل را روی این دیتاست بالانس شده آموزش می‌دهیم. شکل (۲۸) نحوه تقسیم‌بندی، بالانس کردن دیتا با کلاس TomekLinks() و آموزش مدل را نمایش می‌دهد.

Train test split

```
# Divide data into training and validation subsets
X_train, X_valid, y_train, y_valid = train_test_split(df.drop(["Class"], axis=1), df["Class"], train_size=0.8, test_size=0.2,random_state=0)
✓ 0.1s
```

Under-sampling

```
rus = TomekLinks()

X_train, y_train = rus.fit_resample(X_train, y_train)
✓ 1m 31.4s
```

```
model = SVC()
✓ 0.0s
```

```
model.fit(X_train , y_train)
✓ 17.3s
```

SVC

SVC()

شکل ۲۸: آموزش مدل با داده‌های بالانس شده

سپس مدل را تست می‌کنیم و دقت را ارزیابی می‌کنیم. شکل (۲۹) تست مدل و خروجی دقت را نمایش می‌دهد.

```
y_preds = model.predict(X_valid)
✓ 4.6s
```

```
accuracy_score(y_valid , y_preds)
✓ 0.0s
0.998748810488845
```

شکل ۲۹: تست مدل و ارزیابی دقت

همچنین شکل (۳۰) گزارش عملکرد مدل را نمایش می‌دهد.

```
report = classification_report(y_valid, y_preds)
print("Classification Report:\n", report)
✓ 0.0s
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56659
           1       0.72      0.32      0.44        87

    accuracy                           1.00     56746
   macro avg       0.86      0.66      0.72     56746
weighted avg       1.00      1.00      1.00     56746
```

شکل ۳۰: گزارش عملکرد مدل آموزش داده شده

مشاهده می‌شود که مدل آموزش داده شده، عملکرد بهتری نسبت به حالت پایه دارد.
در یک مجموعه داده نامتعادل، افزایش دقت برای کلاس اقلیت (نامعتبر) اغلب شامل چندین استراتژی است که می تواند به بهبود عملکرد مدل کمک کند. این روش‌ها می‌توانند بر دقت کلی و هر کلاس تأثیر بگذارند و در تقاضاهای محاسباتی متفاوت هستند. می‌توان از روش‌های under sampling کمک گرفت. این روش‌ها نیاز به قدرت محاسباتی کمتری دارند، زیرا تعداد داده‌های آموزشی را بسیار کم می‌کنند. روش‌های oversampling نیز موجود هستند که تلاش می‌کنند از کلاس اقلیت، نمونه‌های مصنوعی درست کنند. این روش‌ها احتیاج به پردازش بیشتری دارند زیرا حجم داده‌های آموزشی را بسیار زیاد می‌کنند.

۴-پاسخ سوال چهارم

شکل (۳۱) کتابخانه‌های مورد نیاز را نمایش می‌دهد.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.compose import ColumnTransformer
```

شکل ۳۱: کتابخانه‌های مورد نیاز

همانند قبل دیتاست را بارگذاری می‌کنیم و تعداد مقادیر گمشده و تعداد صفر در ستون‌ها را بررسی می‌کنیم. مشاهده می‌شود که دیتاست دارای مقادی Nan نیست، ولی برخی از ستون‌ها که مجاز به داشتن مقدار صفر نیستند، این مقدار را در برخی از ردیف‌های خود دارند. برای مثال یازده مقدار صفر در ستون BMI داریم که یعنی این اشخاص قد برابر با صفر دارند که منطقی نیست. از این موارد می‌توان استنتاج کرد که مقادیر گمشده به جای Nan با صفر مقداردهی شده است.

ابتدا با روش IQR مانند شکل (۱۰) بخش ۲-۱ به بررسی و حذف داده‌های پرت می‌پردازیم. مشاهده می‌شود که داده پرتی در دیتاست وجود ندارد. حال می‌توانیم به پر کردن مقادیر گمشده بپردازیم. برای این منظور دقیقاً از روش بخش ۲-۲ استفاده می‌کنیم. مشاهده می‌شود که پر کردن مقادیر گمشده با میانه، نتیجه بهتری نسبت به پر کردن مقادیر گمشده با KNN می‌دهد. در نتیجه مقادیر گمشده را با میانه پر می‌کنیم.

۴-۱-آموزش جنگل تصادفی

ابتدا دیتاست را به دو قسمت آموزشی و آزمایشی تقسیم می‌کنیم. سپس با استفاده از روش SMOTE کلاس‌های دیتاست آموزشی را بالانس می‌کنیم. در ادامه یک پایپ‌لاین تعریف می‌کنیم که داده‌های پاس داده شده را به ترتیب با استفاده از روش مین مکس در یک مقیاس قرار می‌دهد و سپس از مدل جنگل تصادفی استفاده می‌کند. شکل (۳۲) نحوه تعریف این پایپ‌لاین را نمایش می‌دهد.

```python
# Preprocessing for numerical data
numerical_transformer = Pipeline(steps=[
    ("scaler", MinMaxScaler()),
])


# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, df.drop("class" , axis= 1).columns),
    ])


my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('model', RandomForestClassifier())
                             ])
```

شکل ۳۲: تعریف پایپ‌لاین برای آموزش یا تست جنگل تصادفی

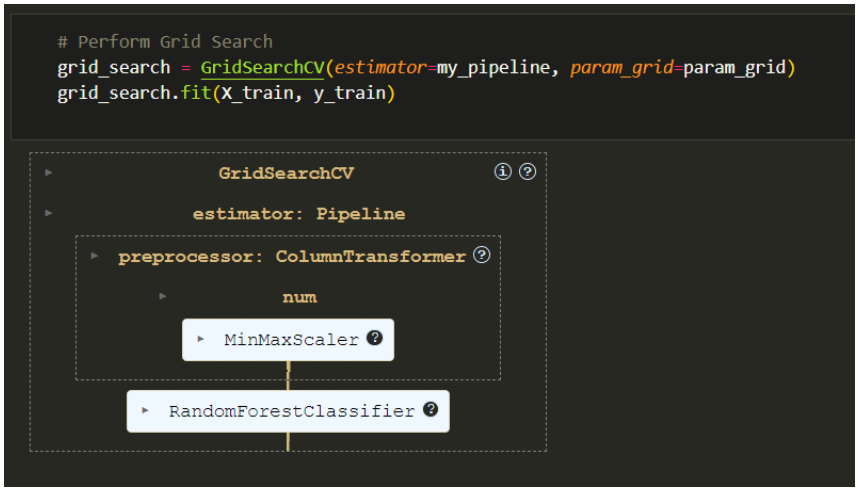
سپس در یک دیکشنری، پارامترهایی که می‌خواهیم به گرید سرچ بدهیم را تعریف می‌کنیم. شکل (۳۳) این دیکشنری را نمایش می‌دهد.

Grid search

```python
# Define the parameter grid
param_grid = {
    'model__n_estimators': [50, 100, 200],
    'model__max_features': [8, 'sqrt', 'log2'],
    'model__max_depth': [None, 10, 20]
}
```

شکل ۳۳: تعریف پارامترهای گرید سرچ

سپس این دیکشنری و پایپ‌لاین تعریف شده را به تابع گرید سرچ پاس می‌دهیم. این تابع ابتدا با استفاده از پایپ‌لاین داده‌ها را مقیاس می‌کند، سپس با پارامترهای داده شده، مدل را تعریف می‌کند، مدل را آموزش می‌دهد، داده‌های تست را مقیاس می‌کند (با استفاده از transform) و مدل را تست می‌کند و نتایج را ذخیره می‌کند. شکل (۳۴) این تابع را نمایش می‌دهد.



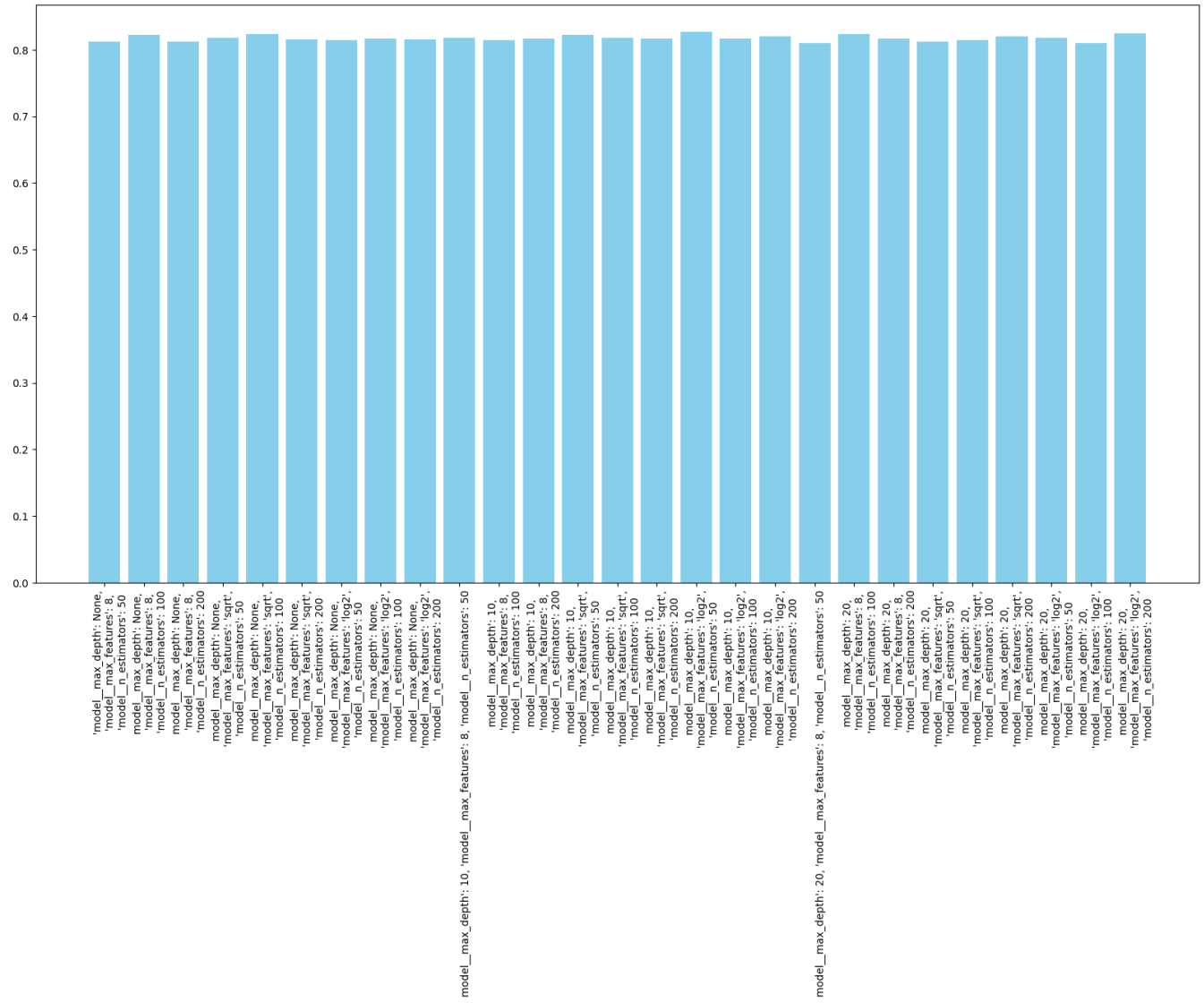
شکل ۳۴: استفاده از گرید سرچ برای بهینه‌سازی هایپرپارامترها

حال با استفاده از یک نمودار، نتایج به دست آمده از گرید سرچ را نمایش می‌دهیم. ابتدا یک لیست از استرینگ‌ها تعریف می‌کنیم که حاوی تمامی حالات استفاده شده از گرید سرچ است. شکل (۳۵) بخشی از این لیست را نمایش می‌دهد.

```
param_values = [
  """
  'model__max_depth': None,
  'model__max_features': 8,
  'model__n_estimators': 50"""
 ,
 """
 model__max_depth': None,
  'model__max_features': 8,
  'model__n_estimators': 100""",
 """
 model__max_depth': None,
  'model__max_features': 8,
  'model__n_estimators': 200""",
 """
 model__max_depth': None,
  'model__max_features': 'sqrt',
  'model__n_estimators': 50""",
 """
 model__max_depth': None,
  'model__max_features': 'sqrt',
  'model__n_estimators': 100""",
 """
 model__max_depth': None,
  'model__max_features': 'sqrt',
  'model__n_estimators': 200""",
 """
 model__max_depth': None,
  'model__max_features': 'log2',
  'model__n_estimators': 50""",
 """
 model__max_depth': None,
  'model__max_features': 'log2',
  'model__n_estimators': 100""",
 """
```

شکل ۳۵: لیست پارامترهای گرید سرچ

حال در یک متغیر، نتایج به دست آمده متناظر با این پارامترها را ذخیره می‌کنیم و نمودار میله‌ای را رسم می‌کنیم. شکل (۳۶) این نمودار را نمایش می‌دهد.



شکل ۳۶: نمودار میله‌ای به دست آمده از گرید سرچ

همچنین شکل (۳۷) بهترین نتیجه به دست آمده و پارامترهای متناظر با آن را نمایش می‌دهد.

```
# Print the best parameters and the best score
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best cross-validation accuracy: {grid_search.best_score_:.4f}")
```

```
Best parameters: {'model__max_depth': 10, 'model__max_features': 'log2', 'model__n_estimators': 50}
Best cross-validation accuracy: 0.8265
```

شکل ۳۷: هایپرپارامترهای بهینه و دقت به دست آمده از آن

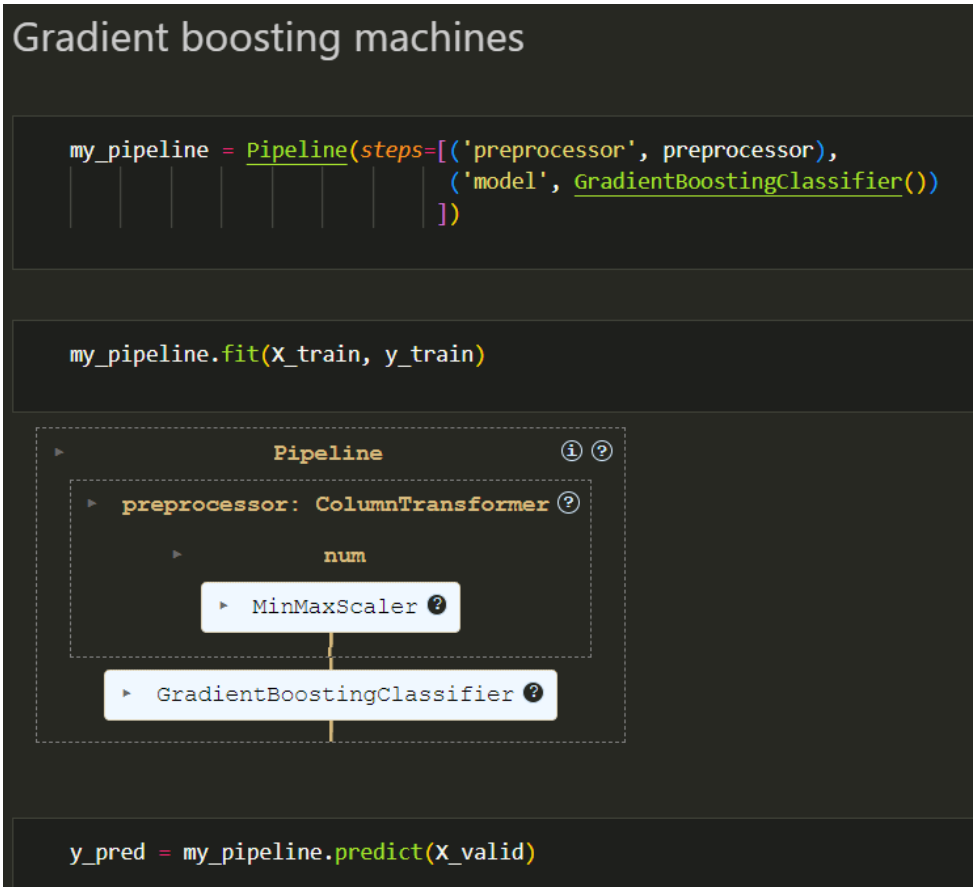
مشاهده می‌شود که دقت به دست آمده از این پارامترها، برابر ۸۲ درصد است.

۴-۲-آموزش سه الگوریتم یادگیری جمعی دیگر

برای این منظور، همانند بخش قبلی، دیتاست را به دو قسمت آموزشی و آزمایشی تقسیم می‌کنیم، لیبل‌ها را بالانس می‌کنیم و همان پایپ‌لاین قبلی را تعریف می‌کنیم، با این تفاوت که به جای مدل جنگل تصادفی، از یک مدل دیگه استفاده می‌کنیم.

۴-۲-۱-استفاده از تقویت گرادیان

شکل (۳۸) پایپ‌لاین به روز شده، آموزش و آزمایش آن را نمایش می‌دهد.



شکل ۳۸: پایپ‌لاین به روز شده برای تقویت گرادیان

سپس دقت حاصل از این روش را محاسبه می‌کنیم.

```
accuracy_score(y_valid, y_pred)
```

```
0.7705627705627706
```

شکل ۳۹: دقت محاسبه شده از تقویت گرادیان

۴-۲-۲-استفاده از آدابوست

مشابه قبل عمل می‌کنیم و در پایپ‌لاین، از مدل آدابوست استفاده می‌کنیم. سپس دقت را محاسبه می‌کنیم. شکل (۴۰) این فرایند را نمایش می‌دهد.

شکل ۴۰: فرایند آموزش و تست آدابوست

مشاهده می‌شود که دقت به دست آمده از این روش ۷۴ درصد است.

۴-۲-۳-استفاده از XGBoost

مشابه بخش قبلی عمل می‌کنیم. شکل (۴۱) این فرایند را نمایش می‌دهد. مشاهده می‌شود در این روش دقت برابر ۷۷ درصد محاسبه می‌شود. جنگل تصادفی با پارامترهای بهینه شده، از هر سه روش عملکرد بهتری دارد.

شکل ۴۱: فرایند آموزش و تست XGBoost