

## امیر بهنام-9831133- تمرین دوم داده کاوی -قسمت پیاده سازی

### آماده سازی داده ها:

در ابتدا کتاب خانه های مورد نیاز را import می کنیم و سپس فایل Csv را به صورت زیر بار گذاری می کنیم:

```
df = pd.read_csv(r'C:\Users\dr. Behnam\Videos\HousePrice\housePrice.csv' , names=['Area', 'Room', 'Parking', 'Warehouse', 'Elevator' , 'Address']
```

در ادامه با استفاده از `isna().sum()` تعداد ردیف های که دارای NaN هستند را برای هر ستون پیدا می کنیم:

```
1 import pandas as pd
2 from sklearn.preprocessing import LabelEncoder, StandardScaler
3 from sklearn.model_selection import train_test_split
4
5
6
7 df = pd.read_csv(r'C:\Users\dr. Behnam\Videos\HousePrice\housePrice.csv' , names=['Area', 'Room', 'Parking', 'Warehouse', 'Elevator' , 'Address'])
8
9 print(df.isna().sum())
10
11
12
13
```



Feature	Count
Area	0
Room	0
Parking	0
Warehouse	0
Elevator	0
Address	23
Price	0

dtype: int64

سپس با استفاده از دستور `df.shape[0]` تعداد سطر ها را قبل از حذف مشخص می کنیم. سپس با استفاده از دستور `df.dropna(how='any')` سطر های که حاوی مقدار NaN هستند را حذف می کنیم و در نهایت با استفاده از دستور `df.shape[0]` تعداد سطر ها را بعد از حذف را مشخص می کنیم:

```

11 number_of_rows_before_deletion = df.shape[0]
12 df = df.dropna(how='any')
13 number_of_rows_after_deletion = df.shape[0]
14 print(df.isna().sum())
15
16

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER Python Debug Console

```

:.\Users\dr. Behnam\.vscode\extensions\ms-python.python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher '50714
r. Behnam\Pictures\HousePrices.py'
Area      0
Room      0
Parking   0
Warehouse 0
Elevator  0
Address   0
Price     0
dtype: int64
PS C:\Users\dr. Behnam\Pictures>

```

## Quartile های pricelevel:

```

16 df['Price'] = pd.to_numeric(df['Price'], errors='coerce')
17 quartiles = df['Price'].quantile([0, 0.25, 0.5, 0.75, 1])
18 labels = ['cheap', 'underMean', 'upperMean', 'expensive']
19 df['priceLevel'] = pd.qcut(df['Price'], q=4, labels=labels)
20
21 print("Quartiles of Price column:\n", df['Price'].describe())
22
23

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```

Quartiles of Price column:
count    3.450000e+03
mean     5.375563e+09
std      8.125918e+09
min      3.600000e+06
25%      1.419250e+09
50%      2.900000e+09
75%      6.000000e+09
max      9.240000e+10
Name: Price, dtype: float64

```

در کد بالا، ستون Price را به چهار quartile تقسیم کردیم و در ادامه به هر کدام label مورد نظر را می‌هیم و یک ستون به نام pricelevel می‌سازیم و در نهایت با استفاده از تابع describe نتایج مورد نظر را چاپ می‌کنیم.

و در ادامه داده‌ها به صورت زیر می‌باشند:

```

24 print("\nUpdated dataframe:\n", df)
25
26
27
28
...
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Updated dataframe:

	Area	Room	Parking	Warehouse	Elevator	Address	Price	priceLevel
0	63	1	True	True	True	Shahran	1850000000	underMean
1	60	1	True	True	True	Shahran	1850000000	underMean
2	79	2	True	True	True	Pardis	550000000	cheap
3	95	2	True	True	True	Shahrake Qods	902500000	cheap
4	123	2	True	True	True	Shahrake Gharb	7000000000	expensive
...	...	...	...	...	...	...	...	...
3468	86	2	True	True	True	Southern Janatabad	3500000000	upperMean
3469	83	2	True	True	True	Niavaran	6800000000	expensive
3470	75	2	False	False	False	Parand	365000000	cheap
3471	105	2	True	True	True	Dorous	5600000000	upperMean
3472	82	2	False	True	True	Parand	360000000	cheap

[3450 rows x 8 columns]

PS C:\Users\dr. Behnam> []

با استفاده از Label Encoding نتایج به صوت زیر می باشد:

```

25 selected_str_cols = ['Parking', 'Warehouse', 'Elevator', 'Address']
26
27 le_Parking = LabelEncoder()
28 le_Warehouse = LabelEncoder()
29 le_Elevator = LabelEncoder()
30 le_Address = LabelEncoder()
31
32 df['Parking'] = le_Parking.fit_transform(df['Parking'])
33 df['Warehouse'] = le_Warehouse.fit_transform(df['Warehouse'])
34 df['Elevator'] = le_Elevator.fit_transform(df['Elevator'])
35 df['Address'] = le_Address.fit_transform(df['Address'])
36
37 print(df[selected_str_cols])
38 print("\nUpdated dataframe For Label Encoding:\n", df)
39
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

[3450 rows x 8 columns]

PS C:\Users\dr. Behnam> c;; cd 'c:\Users\dr. Behnam'; & 'C:\Users\dr. Behnam\AppData\Local\Pro  
\vscode\extensions\ms-python.python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\...\deb  
.py'

	Parking	Warehouse	Elevator	Address
0	1	1	1	156
1	1	1	1	156
2	1	1	1	117
3	1	1	1	152
4	1	1	1	150
...	...	...	...	...
3468	1	1	1	163
3469	1	1	1	105
3470	0	0	0	115
3471	1	1	1	39
3472	0	1	1	115

و همچنین داده ها به صورت کلی به شکل زیر است:

```
25 selected_str_cols = ['Parking', 'Warehouse', 'Elevator', 'Address']
26
27 le_Parking = LabelEncoder()
28 le_Warehouse = LabelEncoder()
29 le_Elevator = LabelEncoder()
30 le_Address = LabelEncoder()
31
32 df['Parking'] = le_Parking.fit_transform(df['Parking'])
33 df['Warehouse'] = le_Warehouse.fit_transform(df['Warehouse'])
34 df['Elevator'] = le_Elevator.fit_transform(df['Elevator'])
35 df['Address'] = le_Address.fit_transform(df['Address'])
36
37 print(df[selected_str_cols])
38 print("\nUpdated dataframe For Label Encoding:\n", df)
39
```

	Area	Room	Parking	Warehouse	Elevator	Address	Price	priceLevel
0	63	1	1	1	1	156	1850000000	underMean
1	60	1	1	1	1	156	1850000000	underMean
2	79	2	1	1	1	117	550000000	cheap
3	95	2	1	1	1	152	902500000	cheap
4	123	2	1	1	1	150	7000000000	expensive
...	...	...	...	...	...	...	...	...
3468	86	2	1	1	1	163	3500000000	upperMean
3469	83	2	1	1	1	105	6800000000	expensive
3470	75	2	0	0	0	115	365000000	cheap
3471	105	2	1	1	1	39	5600000000	upperMean
3472	82	2	0	1	1	115	360000000	cheap

[3450 rows x 8 columns]

نرمال سازی داده ها به صورت زیر است:

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	JUPYTER
	Area	Room	Parking	Warehouse Elevator Address Price priceLevel
0	-0.631533	-1.422380	0.424135	0.304647 0.519406 1.013652 -0.433929 underMean
1	-0.674673	-1.422380	0.424135	0.304647 0.519406 1.013652 -0.433929 underMean
2	-0.401453	-0.106774	0.424135	0.304647 0.519406 0.238524 -0.593934 cheap
3	-0.171372	-0.106774	0.424135	0.304647 0.519406 0.934151 -0.550548 cheap
4	0.231268	-0.106774	0.424135	0.304647 0.519406 0.894401 0.199937 expensive
...	...	...	...	...
3468	-0.300793	-0.106774	0.424135	0.304647 0.519406 1.152777 -0.230846 upperMean
3469	-0.343933	-0.106774	0.424135	0.304647 0.519406 0.000023 0.175321 expensive
3470	-0.458973	-0.106774	-2.357739	-3.282491 -1.925275 0.198774 -0.616704 cheap
3471	-0.027572	-0.106774	0.424135	0.304647 0.519406 -1.311732 0.027624 upperMean
3472	-0.358313	-0.106774	-2.357739	0.304647 0.519406 0.198774 -0.617320 cheap

[3450 rows x 8 columns]  
PS C:\Users\dr. Behnam>

```
54
55 X = df.drop(['priceLevel'], axis=1)
56 y = df['Price']
57
58 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
59 print("Training set shape:", X_train.shape)
60 print("Testing set shape:", X_test.shape)
61
62
63
64
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    JUPYTER

PS C:\Users\dr. Behnam> c:: cd 'c:\Users\dr. Behnam'; & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\P
.vscode\extensions\ms-python.python-2022.16.1\pythonFiles\lib\python\debugpy\adapter/../..debugpy\launcher'
.py'
PS C:\Users\dr. Behnam> c:: cd 'c:\Users\dr. Behnam'; & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\P
.vscode\extensions\ms-python.python-2022.16.1\pythonFiles\lib\python\debugpy\adapter/../..debugpy\launcher'
.py'
Training set shape: (2760, 7)
Testing set shape: (690, 7)
PS C:\Users\dr. Behnam>
```

در اینجا ستون PriceLevel را حذف کردیم و ستون Price را به عنوان برچسپ در نظر گرفتیم (می توانستیم بر عکس نیز در نظر بگیریم و اینجا برای نمونه به این شکل انجام شده) و در ادامه 80٪ را برای آموزش و 20٪ را برای تست انتخاب کردیم و سپس آن را چاپ کردیم.

## رگرسیون:

رگرسیون خطی به صورت زیر است:

```
65 #Linear Regression
66
67 model = LinearRegression()
68 model.fit(X_train, y_train)
69 y_train_pred = model.predict(X_train)
70 y_test_pred = model.predict(X_test)
71 train_r2, test_r2 = r2_score(y_train, y_train_pred), r2_score(y_test, y_test_pred)
72 train_mse, test_mse = mean_squared_error(y_train, y_train_pred), mean_squared_error(y_test, y_test_pred)
73
74 print(f'Training set:\nR-squared: {train_r2:.3f}\nMSE: {train_mse:.3f}')
75 print(f'Testing set:\nR-squared: {test_r2:.3f}\nMSE: {test_mse:.3f}')
76
77
78
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Python Debug

Windows PowerShell

Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\dr. Behnam> & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\python.exe' 'c:\Users\dr. Behnam\.vscc  
python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '55936' '--' 'c:\Users\dr. Behnam\Official.py'

Training set:

R-squared: 0.540

MSE: 0.443

Testing set:

R-squared: 0.532

MSE: 0.536

PS C:\Users\dr. Behnam>

این قطعه کد تحلیل رگرسیون خطی را روی یک مجموعه داده ما با متغیرهای ورودی X\_train و متغیر خروجی y\_train انجام می دهد. تابع LinearRegression() نمونه ای از مدل رگرسیون خطی را ایجاد می کند. متد fit() با استفاده از X\_train و y\_train به عنوان آرگومان برای آموزش مدل فراخوانی می شود. سپس از مدل آموزش دیده برای پیش بینی پاسخ های مجموعه آموزشی و آزمایشی با استفاده از روش predict() استفاده می کنیم که مقادیر پیش بینی شده y\_train و y\_test را برمی گرداند.

سپس ضریب تعیین (R-squared) و میانگین مربعات خطا (MSE) برای مجموعه های آموزشی و آزمایشی به ترتیب با استفاده از توابع `r2_score()` و `mean_squared_error()` محاسبه می شود. R-squared میزان تناسب خط رگرسیون با داده ها را اندازه گیری می کند و MSE میانگین مجذور اختلاف بین مقادیر پیش بینی شده و واقعی را اندازه گیری می کند. در نهایت، مقادیر R-squared و MSE را برای مجموعه های آموزشی و آزمایشی چاپ می کنیم.

رگرسیون چند جمله ای 2 درجه:

```
78 poly = PolynomialFeatures(degree=2)
79 X_train_poly = poly.fit_transform(X_train)
80 X_test_poly = poly.transform(X_test)
81
82 model = LinearRegression()
83 model.fit(X_train_poly, y_train)
84 y_train_pred = model.predict(X_train_poly)
85 y_test_pred = model.predict(X_test_poly)
86 train_r2 = r2_score(y_train, y_train_pred)
87 test_r2 = r2_score(y_test, y_test_pred)
88 train_mse = mean_squared_error(y_train, y_train_pred)
89 test_mse = mean_squared_error(y_test, y_test_pred)
90
91 print(f'Training set R-squared: {train_r2:.3f}, MSE: {train_mse:.3f}\n')
92 print(f'Testing set R-squared: {test_r2:.3f}, MSE: {test_mse:.3f}\n')
93
94
95
96
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Copyright (C) 2009 Microsoft Corporation. All rights reserved.

```
PS C:\Users\dr. Behnam> & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\python.exe' 'c:\python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '55991' '--' 'c:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\python.exe'
Training set R-squared: 0.674, MSE: 0.314
```

```
Testing set R-squared: 0.721, MSE: 0.320
```

در گام اول یک شی `PolynomialFeatures` با درجه 2 ایجاد می کنیم که با ایجاد ویژگی های چند جمله ای تا درجه 2، ماتریس ویژگی اصلی `X_train` را به یک ماتریس جدید `X_train_poly` تبدیل می کند. همان تبدیل با استفاده از روش `transform()` روی داده های تست اعمال می شود. از شیء `PolynomialFeatures`.

سپس، یک مدل رگرسیون خطی ایجاد می شود و بر روی داده های آموزشی تبدیل شده ( $X_{train\_poly}$ ) و متغیر هدف مربوطه ( $y_{train}$ ) آموزش داده می شود. سپس این مدل برای پیش بینی متغیر هدف برای داده های آموزشی ( $y_{train\_pred}$ ) و داده های آزمون ( $y_{test\_pred}$ ) استفاده می کنیم.

در نهایت، عملکرد مدل با استفاده از دو معیار - R-squared ( $r2\_score$ ) و میانگین مربعات خطا ( $mean\_squared\_error$ ) ارزیابی می شوند. این معیارها برای هر دو مجموعه آموزشی و آزمایشی محاسبه شده و به ترتیب در متغیرهای  $train\_r2$ ،  $test\_r2$ ،  $train\_mse$  و  $test\_mse$  ذخیره می شوند و در ادامه عبارت های مورد نیاز را چاپ می کنیم.

رگرسیون چند جمله ای 3 درجه:

```
97 #Polynomial Regression With 3 Degrees
98 poly = PolynomialFeatures(degree=3)
99 X_train_poly = poly.fit_transform(X_train)
100 X_test_poly = poly.transform(X_test)
101
102 model = LinearRegression()
103 model.fit(X_train_poly, y_train)
104 y_train_pred = model.predict(X_train_poly)
105 y_test_pred = model.predict(X_test_poly)
106 train_r2 = r2_score(y_train, y_train_pred)
107 test_r2 = r2_score(y_test, y_test_pred)
108 train_mse = mean_squared_error(y_train, y_train_pred)
109 test_mse = mean_squared_error(y_test, y_test_pred)
110
111 print(f'Training set R-squared: {train_r2:.3f}, MSE: {train_mse:.3f}\n')
112 print(f'Testing set R-squared: {test_r2:.3f}, MSE: {test_mse:.3f}\n')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Python

```
PS C:\Users\dr. Behnam> c:; cd 'c:\Users\dr. Behnam'; & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\pyt
\vscode\extensions\ms-python.python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56010' '--'
.py'
```

```
Training set R-squared: 0.754, MSE: 0.237
```

```
Testing set R-squared: 0.726, MSE: 0.314
```

توضیحات این قسمت همانند بخش قبل است با این تفاوت که از Degree = 3 به جای 2 استفاده می کنیم.

**دسته بندی:**

درخت تصمیم:



```

79 #Decision Tree
80 clf = DecisionTreeClassifier(criterion='entropy')
81 clf.fit(X_train, y_train)
82 y_pred = clf.predict(X_test)
83 training_accuracy = accuracy_score(y_train, clf.predict(X_train))
84 testing_accuracy = accuracy_score(y_test, y_pred)
85 print("Training Accuracy:", training_accuracy)
86 print("Testing Accuracy:", testing_accuracy)
87
88

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Windows PowerShell

Copyright (C) 2009 Microsoft Corporation. All rights reserved.

```

PS C:\Users\dr. Behnam> & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\python.exe' 'c:\Users\dr. Behnam\python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\../..debugpy\launcher' '56186' '--' 'c:\Users\dr. Behnam\Office
Training Accuracy: 0.9746376811594203
Testing Accuracy: 0.7695652173913043
PS C:\Users\dr. Behnam>

```

در اینجا یک طبقه‌بندی درخت تصمیم را بر روی برخی از داده‌های ورودی با استفاده از معیار آنتروپی برای اندازه‌گیری کیفیت هر تقسیم آموزش می‌دهیم. داده‌های ورودی به مجموعه‌های آموزشی و آزمایشی تقسیم می‌شوند، جایی که  $X_{train}$  و  $y_{train}$  به ترتیب ویژگی‌ها و برچسب‌های مجموعه آموزشی را نشان می‌دهند، در حالی که  $X_{test}$  نشان دهنده ویژگی‌های مجموعه آزمایشی است. هنگامی که طبقه‌بندی‌کننده با استفاده از  $clf.fit(X_{train}, y_{train})$  آموزش داده شد، برای پیش‌بینی برچسب‌های مجموعه آزمایشی با استفاده از  $y_{pred} = clf.predict(X_{test})$  استفاده می‌شود. دو خط کد بعدی دقت طبقه‌بندی‌کننده را در مجموعه‌های آموزشی و آزمایشی با استفاده از تابع  $accuracy\_score$  از `sklearn.metrics` محاسبه می‌کنند و این مقادیر را چاپ می‌کنند. خروجی چاپ شده به ترتیب دقت طبقه‌بندی‌کننده را در مجموعه‌های آموزشی و آزمایشی نمایش می‌دهد.

جنگل تصادفی:

ابتدا یک شی طبقه‌بندی‌کننده `clf` را به‌عنوان `RandomForestClassifier` با معیار پارامتر روی «Entropy» مقداردهی اولیه می‌کنیم. این پارامتر کیفیت تقسیم در الگوریتم درخت تصمیم مورد استفاده توسط جنگل تصادفی

در ادامه با استفاده از تابع `accuracy_score()` از `scikit-learn`، دقت مدل را روی داده‌های آموزش و آزمایش محاسبه می‌کنیم..

در نهایت، دقت آموزش و دقت تست چاپ می شود. این مقادیر به ترتیب نشان دهنده میزان عملکرد خوب مدل بر روی داده های آموزش و آزمایش هستند.

```
PS C:\Users\dr. Behnam> & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\python.exe' 'c:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56186' '--' 'c:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56186'
Training Accuracy: 0.9746376811594203
Testing Accuracy: 0.7695652173913043

PS C:\Users\dr. Behnam> c:; cd 'c:\Users\dr. Behnam'; & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\python.exe' 'c:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56186' '--' 'c:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56186'
Training Accuracy: 0.9746376811594203
Testing Accuracy: 0.7478260869565218
```

: KNN

```

78
79
80
81 #KNN algorithm for k=3, 5, and 7
82 for k in [3, 5, 7]:
83     knn = KNeighborsClassifier(n_neighbors=k)
84     knn.fit(X_train, y_train)
85
86     # Evaluate the accuracy of the model on both training and testing sets
87     train_acc = knn.score(X_train, y_train)
88     test_acc = knn.score(X_test, y_test)
89     print(f"k={k}, Training Accuracy: {train_acc:.3f}, Testing Accuracy: {test_acc:.3f}")

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

Windows PowerShell

Copyright (C) 2009 Microsoft Corporation. All rights reserved.

```

PS C:\Users\dr. Behnam> & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\python.exe' 'c:\Users\c
python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56237' '--' 'c:\Users\dr. Behnam
k=3, Training Accuracy: 0.840, Testing Accuracy: 0.699
k=5, Training Accuracy: 0.792, Testing Accuracy: 0.722
k=7, Training Accuracy: 0.771, Testing Accuracy: 0.720

```

پیاده سازی آن به این صورت است که ابتدا فهرستی [3، 5، 7] حاوی سه مقدار ممکن برای تعداد همسایه‌هایی که در الگوریتم KNN در نظر گرفته می‌شود، مقداردهی اولیه می‌کند. برای هر مقدار در این لیست، یک شی KNeighborsClassifier جدید با تعداد همسایه‌های مشخص شده ایجاد می‌شود و آن را با داده‌های آموزشی X\_train و y\_train متناسب می‌کند.

پس از برآزش مدل، با استفاده از روش score () شی KNeighborsClassifier دقت آن را در مجموعه‌های آموزشی و آزمایشی ارزیابی می‌کند. دقت آموزش معیاری است از میزان تناسب مدل با داده‌های آموزشی، در حالی که دقت تست تخمینی از تعمیم مدل به داده‌های جدید و نادیده می‌دهد.

در نهایت، کد نتایج را خروجی می‌دهیم، مقدار k و همچنین دقت آموزش و آزمایش مربوطه را برای هر k چاپ می‌کند.


: SVM

```

81 #SVM algorithm in linear mode
82 svm_linear = SVC(kernel='linear')
83 svm_linear.fit(X_train, y_train)
84 train_acc_linear = svm_linear.score(X_train, y_train)
85 test_acc_linear = svm_linear.score(X_test, y_test)
86 print(f"Linear SVM: Training Accuracy: {train_acc_linear:.3f}, Testing Accuracy: {test_acc_linear:.3f}")
87
88 #SVM algorithm in non-linear mode
89 svm_nonlinear = SVC(kernel='rbf')
90 svm_nonlinear.fit(X_train, y_train)
91 train_acc_nonlinear = svm_nonlinear.score(X_train, y_train)
92 test_acc_nonlinear = svm_nonlinear.score(X_test, y_test)
93 print(f"Non-Linear SVM: Training Accuracy: {train_acc_nonlinear:.3f}, Testing Accuracy: {test_acc_nonlinear:.3f}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

 Python Debug Console

Windows PowerShell

Copyright (C) 2009 Microsoft Corporation. All rights reserved.

```

PS C:\Users\dr. Behnam> & 'C:\Users\dr. Behnam\AppData\Local\Programs\Python\Python38-32\python.exe' 'c:\Users\dr. Behnam\.vscode\ext
python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56253' '--' 'c:\Users\dr. Behnam\Official2.py'
Linear SVM: Training Accuracy: 0.594, Testing Accuracy: 0.596
Non-Linear SVM: Training Accuracy: 0.630, Testing Accuracy: 0.642
PS C:\Users\dr. Behnam>

```

این کد دو طبقه‌بندی‌کننده ماشین بردار پشتیبان (SVM) را بر روی یک مجموعه داده را آموزش و ارزیابی می‌کند. اولین مدل SVM با استفاده از هسته خطی آموزش داده می‌شود و دقت آموزش و تست آن محاسبه و چاپ می‌شود. سپس مدل دوم SVM با استفاده از هسته غیرخطی (تابع پایه شعاعی) آموزش داده می‌شود و دقت آموزش و تست آن نیز محاسبه و چاپ می‌شود.

چند خط اول کد نمونه ای از کلاس SVC برای SVM با هسته خطی ایجاد می‌کند و آن را با داده های آموزشی متناسب می‌کند. سپس دقت آموزش و تست این مدل با استفاده از روش `score` (محاسبه شده و به ترتیب در متغیرهای `train_acc_linear` و `test_acc_linear` ذخیره می‌شود. سپس این دقت ها با چاپ می‌کنیم. خطوط بعدی نمونه دیگری از کلاس SVC را برای SVM با هسته تابع پایه شعاعی (RBF) ایجاد کرده و آن را به داده های آموزشی برازش می‌دهند. دقت آموزش و تست این مدل به طور مشابه محاسبه شده و در نهایت چاپ می‌شود.

به طور کلی، این کد راهی برای مقایسه عملکرد مدل های SVM با هسته های مختلف در یک مجموعه داده ارائه می‌کند.

### دسته بندی داده ها با استفاده از یادگیری عمیق:

در ابتدا ستون `pricelevel` را به صورت زیر `one hot encode` می‌کنیم:

```

83 # Encode the labels as integers using LabelEncoder
84 le = LabelEncoder()
85 y = le.fit_transform(y)
86
87 # Perform one-hot encoding on the integer-encoded labels
88 y = to_categorical(y)
89
90 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
91 #print("Training set shape:", X_train.shape)
92 #print("Testing set shape:", X_test.shape)
93
94

```

در ادامه مدل و شبکه ی عصبی را می سازیم و آن را آموزش و در نهایت مورد ارزیابی قرار می دهیم:

```

95 # Build the model
96 model = Sequential()
97 model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
98 model.add(Dropout(0.2))
99 model.add(Dense(32, activation='relu'))
100 model.add(Dropout(0.2))
101 model.add(Dense(4, activation='softmax'))
102
103 # Compile the model
104 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
105
106 # Train the model
107 opt = Adam(learning_rate=0.001)
108 history=model.fit(X_train, y_train, epochs=12000, batch_size=256, validation_data=(X_test, y_test))
109
110 # Evaluate the model
111 train_acc=model.evaluate(X_train, y_train)
112 print('Training accuracy:', train_acc[1])
113 test_acc=model.evaluate(X_test, y_test)
114 print('Test accuracy:', test_acc[1])
115

```

قسمت اول کد یک مدل ترتیبی ایجاد می کند. این بدان معنی است که لایه ها به صورت متوالی روی هم قرار می گیرند. در این مورد، سه لایه متراکم با عملکرد فعال سازی `relu`، و دو لایه حذفی وجود دارد که به جلوگیری از برازش بیش از حد کمک می کند. شکل ورودی به صورت `(X_train.shape[1])` تعریف می شود، به این معنی که مدل انتظار ورودی هایی با ویژگی های `X_train.shape[1]` را دارد.

مرحله بعدی کامپایل مدل با مشخص کردن تابع ضرر، بهینه ساز و متریک ارزیابی است. در این مورد، تابع تلفات متقابل آنتروپی طبقه ای استفاده می شود، زیرا این یک مشکل طبقه بندی چند کلاسه است، از بهینه ساز `Adam` استفاده می شود و متریک دقت در طول آموزش ردیابی می شود.

پس از کامپایل مدل، با استفاده از روش `fit()` آموزش داده می شود. داده های آموزشی (`y_train, X_train`) به همراه برخی از فرآیندها مانند تعداد دوره ها، اندازه دسته و داده های اعتبارسنجی به مدل منتقل می شود. بهینه ساز را نیز می توان در این مرحله مشخص کرد. در این حالت از بهینه ساز Adam با نرخ یادگیری 0.001 استفاده می شود.

در نهایت، مدل آموزش دیده شده بر روی داده های آموزشی و آزمون با استفاده از روش ارزیابی (`evaluate()`) ارزیابی می شود. دقت آموزش و دقت تست بر روی کنسول چاپ می شود.

همچنین کد مربوط به ماتریس درهم ریختگی نیز به صورت زیر می باشد:

```
116 # Predict the test data and print the confusion matrix
117 y_pred = model.predict(X_test)
118 cm = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1))
119 print('Confusion matrix:\n', cm)
```

پس از تست کردن انواع هایپرپارامترها بهترین نتایج و همچنین ماتریس درهم ریختگی به صورت زیر می باشد:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
11/11 [=====] - 0s 5ms/step - loss: 0.6525 - accuracy: 0.7370 - val_loss: 0.8422 - val_accuracy: 0.7058
87/87 [=====] - 0s 1ms/step - loss: 0.5528 - accuracy: 0.7844
Training accuracy: 0.7844203114509583
22/22 [=====] - 0s 1ms/step - loss: 0.8422 - accuracy: 0.7058
Test accuracy: 0.7057970762252808
Confusion matrix:
[[108  6 22 28]
 [ 2 149  3 31]
 [ 15 10 110 35]
 [ 12 24 15 120]]
```

از `learning rate` با مقدار بسیار کمی استفاده کردیم زیرا نرخ یادگیری کمتر منجر به همگرایی کندتر می شود، اما به طور بالقوه می تواند منجر به عملکرد بهتر شود زیرا به روز رسانی ها دقیق تر هستند و مقدار آن را 0.001 که کمترین مقدار است قرار دادیم.

از `softmax` به جای `sigmoid` استفاده شده است زیرا برای طبقه بندی چند کلاسه بسیار مناسب تر و کاربردی تر است.

از batch size و epochs زیادی استفاده شده است به طوری هر چه مقدار آنها بیشتر شد دقت بیشتری نیز بدست آمد. با epochs ۱۲۰۰۰ و batch size ۲۵۶ بهترین نتیجه حاصل شد و با مقادیر کمتری نیز تست شدند اما دقت کمتر از بهترین عملکرد بود و حتی زمانی که تعداد epochs را ۲۰ و تعداد batch size را ۳۲ قرار داده ام. دقت در بازه ۶۰ قرار گرفت که خیلی از بهترین عملکرد فاصله دارد. در نتیجه باید نرخ یادگیری را کم یا حتی کمترین مقدار در نظر بگیریم تا دقت و آموزش بالاتر رود و تعداد batch size و epochs را نیز باید مقادیر بالا و زیادی در نظر بگیریم تا عملکرد بسیار خوبی حاصل شود.