دانشکده مهندسي برق و کامپیوتر

نام و نام خانوادگی :

امیر اسماعیل زاده نوبری

شماره دانشجویی

40101924

درس یادگیری ماشین

مینی پروژه اول

استاد درس:

دکتر علیاری

بهار 1403


بسم الله الرحمن الرحیم

Contents

لینک GIT: https://github.com/CAmiren/Machine_learning4022

لینک colab :

https://drive.google.com/file/d/1SKr2VqhQViN8_byTM1vmXlHRIAv4nQQv/view?usp=sharing

https://drive.google.com/file/d/1Q1z9HYxpJHCJr3V-CR2Yn8MUYrY9RMcr/view?usp=sharing

https://drive.google.com/file/d/1cpSlO9KbgPJcSv2tXAt_YF8s4D8bhAwa/view?usp=sharing

سوال 1

1 .

مراحل آموزش و ارزیابی یک مدل رگرسیون لجستیک (Logistic Regression) در مسائل طبقه‌بندی (Classification) به صورت زیر است:

آموزش:

1. تهیه داده‌ها: ابتدا داده‌هایی که شامل ویژگی‌ها (متغیرهای مستقل) و برچسب‌ها (متغیر وابسته، که معمولاً یک مقدار دودویی یا چندگانه است) هستند را تهیه می‌کنیم.

2. پیش‌پردازش داده: این مرحله شامل تمیزکاری داده‌ها، مقیاس‌بندی ویژگی‌ها، و شاید تبدیل ویژگی‌ها به فرمتی که مناسب برای مدل لجستیک رگرسیون (استانداردسازی ویژگی‌ها) .

3. تقسیم داده: داده‌ها را به دو بخش تقسیم می‌کنیم، معمولاً بخش آموزش (training) و بخش ارزیابی (testing) یا اعتبارسنجی (validation).

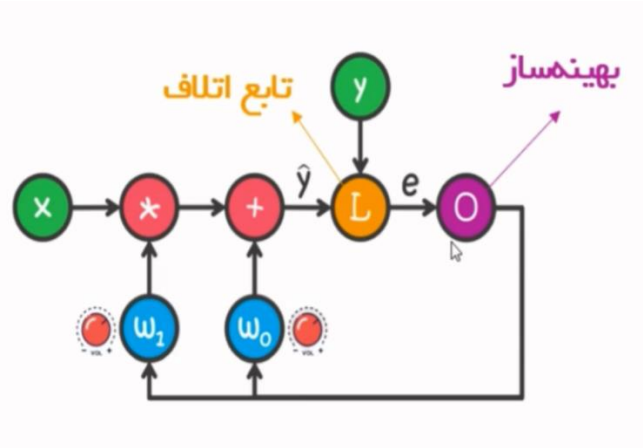
4. آموزش مدل: مدل رگرسیون لجستیک را بر روی بخش آموزش داده‌ها آموزش می‌دهیم. این شامل تخمین پارامترهای مدل است که توسط الگوریتم بهینه‌سازی (روش گرادیان کاهشی) انجام می‌شود ولی قبل از آن باید تابع اتلافی معرفی کنیم تا داده های تخمین زده شده و واقعی با هم مقایسه و از هم کم شوند تا بتوانیم با الگوریتم بهینه سازی آن را به حداقل مقدار خود برسانیم.

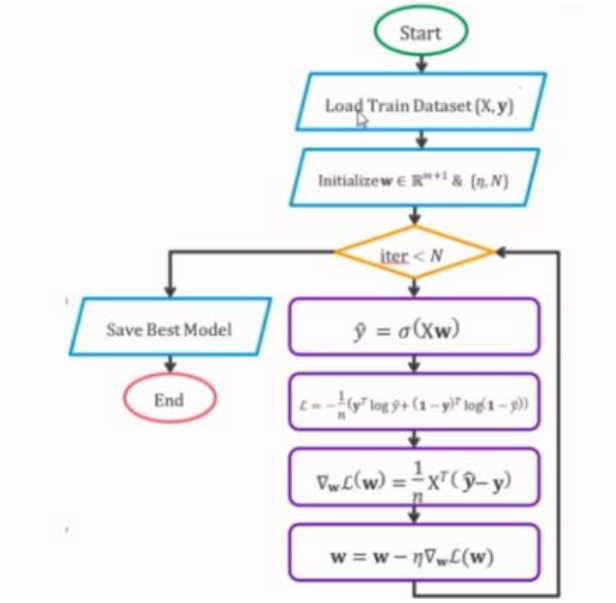
ارزیابی:

1. پیش‌بینی: با استفاده از مدل آموزش دیده، پیش‌بینی‌هایی برای بخش ارزیابی داده‌ها انجام می‌دهیم.

2. محاسبه معیارهای ارزیابی: اکنون از معیارهایی مانند دقت (Accuracy)، دقت مثبت (Precision)، بازخوانی (Recall)، اندیس F1 و ماتریس درهم‌ریختگی (Confusion Matrix) برای ارزیابی عملکرد مدل استفاده می‌کنیم.

3. تنظیم پارامترها : اگر عملکرد مدل نسبت به ارزیابی ناکام بود، می‌توانیم پارامترهای مدل یا فرضیات مسئله را تغییر دهیم و مراحل آموزش و ارزیابی را مجدداً انجام دهیم.





به دلیل کمبود وقت از بلوک دیاگرام های درس داده شده استفاده کردم

پیش‌پردازش داده (Data Preprocessing):

در این مرحله، داده‌های ورودی به مدل آماده می‌شوند. برای رگرسیون لجستیک دوکلاسه و چندکلاسه، این مرحله ممکن است در هر دو نوع مدل وجود داشته باشد. تفاوت اصلی ممکن است در تعیین شیوه کدگذاری برچسب‌ها (برای مدل چندکلاسه) و یا تفکیک داده‌های مربوط به هر کلاس (برای مدل دوکلاسه) مشخص شود.

انتخاب مدل (Model Selection):

در این مرحله، معمولاً باید مشخص شود که مدل رگرسیون لجستیک دوکلاسه یا چندکلاسه استفاده شود. این تفاوت ممکن است در بلوک انتخاب مدل دیده شود.

آموزش مدل (Model Training):

در مرحله آموزش مدل، الگوریتم‌های یادگیری برای هر کدام از مدل‌ها (دوکلاسه و چندکلاسه) ممکن است متفاوت باشند. این تفاوت معمولاً در بلوک آموزش مدل قابل مشاهده است، که ممکن است الگوریتم‌های مختلف یادگیری و پارامترهای متفاوت مدل‌ها برای هر کدام از مدل‌ها اعمال شود.

ارزیابی مدل (Model Evaluation):

در این مرحله، معیارهای ارزیابی مدل بررسی می‌شوند. این معیارها ممکن است برای مدل‌های دوکلاسه و چندکلاسه متفاوت باشند. این تفاوت ممکن است در بلوک ارزیابی مدل دیده شود.

به طور کلی، تفاوت‌های بین رگرسیون لجستیک دوکلاسه و چندکلاسه در بلوک‌های مختلفی از فرآیند آموزش، انتخاب مدل و ارزیابی مدل مشخص می‌شوند.

.2

با دستور make_classification() 1000 نمونه داده با 4 کلاس و 3 ویژگی تولید می کنم .
با توجه به دو رقم پایانی شماره دانشجویی شماره 24 را برای random_state انتخاب کردم.

پارامتر های متفاوتی برای این دستور وجود دارد مانند n_sample (تعداد نمونه) ، n_feature (تعداد ویژگی) ، n_classes (تعداد کلاس ها) ، n_redundant (تعداد ویژگی های رداندنت) ، class_sep که میزان فاصله ی کلاس ها ازهم را مشخص میکند، و همچنین پارامتر های دیگر که برای تولید دیتا ست استفاده میشود.

برای تولید دیتاستی که چالش‌برانگیزتر و سخت‌تر باشد، می‌توانید روش‌های مختلفی را به کار روش هایی از قبیل افزایش تنوع داده‌ها ، کاهش نمونه‌های لیبل دار ، افزودن نویز به داده‌ها ، امتزاج ویژگی‌ها ، متعادل نکردن دیتاست ، دستکاری برچسب‌ها ، استفاده از داده‌های انحرافی و...

برای چالش انگیز تر کردن داده با استفاده از دستور make_classification میتوان از تغییر پارامتر های زیر استفاده کرد :

n_informative: تعداد ویژگی‌های اطلاعاتی. افزایش این عدد می‌تواند دیتاست را آسان‌تر کند، در حالی که کاهش آن باعث می‌شود استخراج اطلاعات مفید از ویژگی‌ها دشوارتر شود.

n_redundant: تعداد ویژگی‌های اضافی که ترکیب خطی از ویژگی‌های اطلاعاتی هستند. افزایش این پارامتر باعث می‌شود دیتاست حاوی اطلاعات تکراری بیشتری باشد، که می‌تواند فرایند یادگیری را پیچیده‌تر کند.

flip_y: این پارامتر درصد نمونه‌هایی را تعیین می‌کند که برچسب آن‌ها به طور تصادفی تغییر می‌کند. افزایش این مقدار می‌تواند سطح نویز در برچسب‌ها را افزایش دهد و دیتاست را دشوارتر می‌کند.

class_sep: معیاری برای جداسازی کلاس‌ها. مقادیر پایین‌تر این پارامتر باعث تداخل بیشتر بین کلاس‌ها می‌شود، که می‌تواند تشخیص کلاس‌ها را سخت‌تر کند.

weights: توزیع کلاس‌ها. با تنظیم این پارامتر برای ایجاد توزیع نامتوازن کلاس‌ها، می‌توانیم دیتاستی تولید کنیم که در آن یک یا چند کلاس به نسبت سایرین نادرتر باشند، که این می‌تواند چالش‌هایی را در یادگیری ایجاد کند.

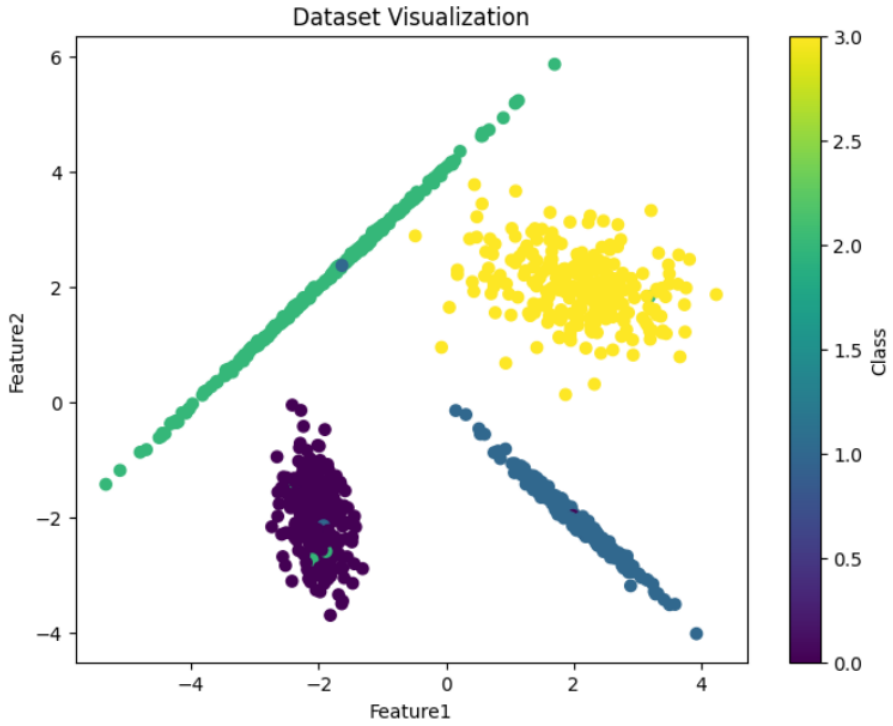
n_clusters_per_class: تعداد خوشه‌ها در هر کلاس. با افزایش تعداد خوشه‌ها در هر کلاس، داده‌ها در فضای ویژگی پیچیده‌تر و دارای توزیع‌های محلی متفاوتی می‌شوند، که تشخیص الگو را سخت‌تر می‌کند.

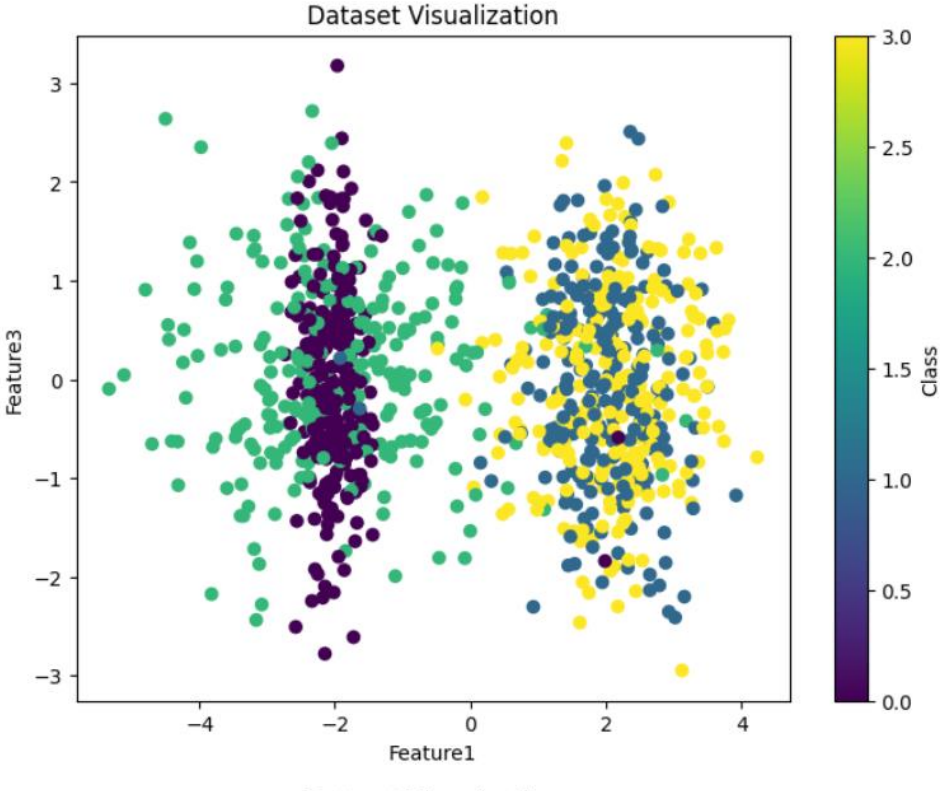
تنظیم این پارامترها به شیوه‌ای که متناسب با هدف آزمایشی ما باشد، می‌تواند به تولید دیتاست‌هایی منجر شود که چالش هایی برای مدل‌های یادگیری ماشین به ارمغان آورد.

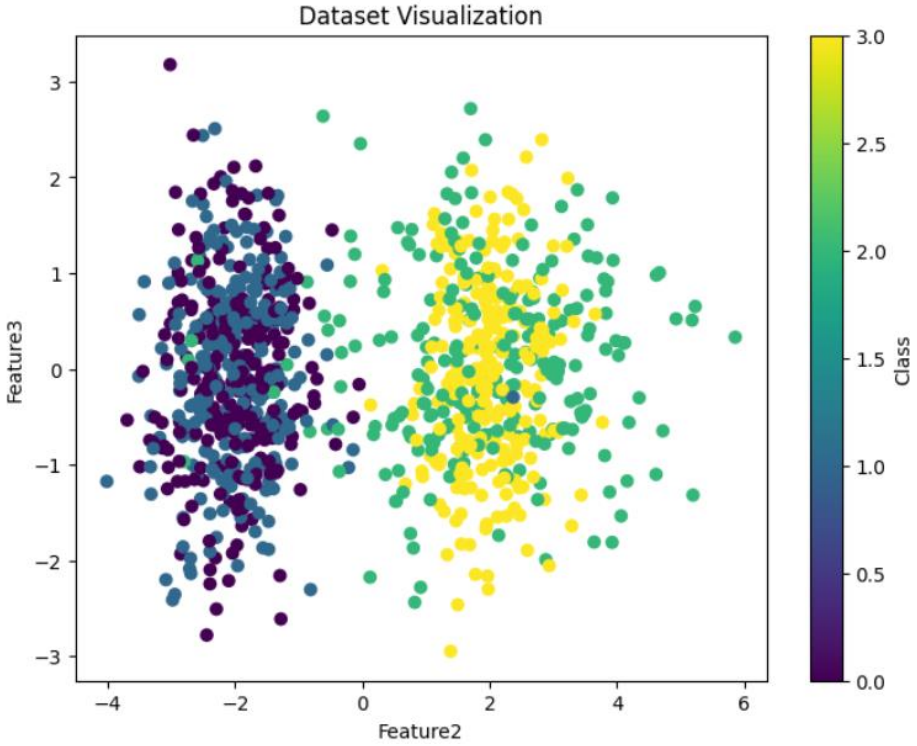
```
      Feature1  Feature2  Feature3  Target
0     2.628417 -2.698842 -0.387281       1
1     1.816352 -1.673546 -0.282533       1
2    -3.102045  0.877601 -0.847032       2
3     2.714914 -2.655495 -0.643499       1
4     1.894970 -1.957617  0.653826       1
..         ...       ...       ...     ...
995  -4.313079 -0.361263 -1.070625       2
996   2.835863 -2.667208  1.753649       1
997   0.175843  2.292155  1.846968       3
998   2.914391 -2.934094  0.462912       1
999   2.766699  1.702590 -0.798421       3

[1000 rows x 4 columns]
(1000, 3) (1000,)
```

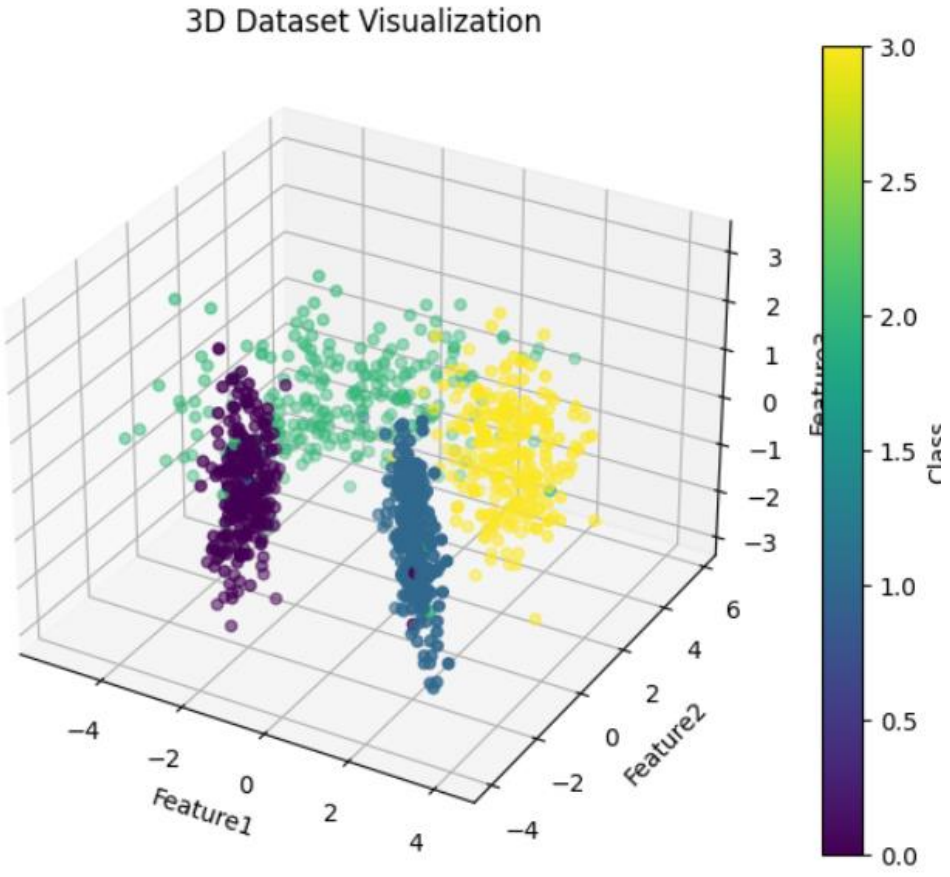
همینطور ابعاد X و y دیده میشود

چون 3 feature داریم پس فضای 3 بعدی داریم در ادامه visualization دوبعدی تک به تک feature ها و سه بعدی نمایش داده شده است:


Dataset Visualization
Feature2
Feature1
Class
6
4
2
0
−2
−4
−4
−2
0
2
4
3.0
2.5
2.0
1.5
1.0
0.5
0.0


Dataset Visualization
Feature3
Feature1
Class
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
3.0
2.5
2.0
1.5
1.0
0.5
0.0


Dataset Visualization
Feature3
Feature2
Class
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6
3.0
2.5
2.0
1.5
1.0
0.5
0.0



همانطور که پیداست داده های ما بنا بر پارمتر های تعریفی خودمان زیاد چالش برانگیز نیست.

3 .

با استفاده از classifier های logistic regression و perceptron به تفکیک دیتاست قسمت قبل می پردازیم .

ابتدا نیاز به ذکر است که کتاب خونه های استفاده شده در این سوال عبارت اند از:

Numpy , pandas , sklearn , matplotlib, mpl_toolkits.mplot3d , mlxtend

داده ها را به داده های آموزش و تست تقسیم می کنیم:

```
[ ]  x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2 , random_state=24)

     x_train.shape, y_train.shape, x_test.shape, y_test.shape ,

     ((800, 3), (800,), (200, 3), (200,))
```

الف) Logistic regression :

بر خلاف پرسپترون، رگرسیون لجستیک یک مدل آماری است که احتمالات را با استفاده از تابع لجستیک، که پیوسته است، تخمین می‌زند. این ویژگی آن را برای وظایف طبقه‌بندی دودویی مفید می‌سازد.

رگرسیون لجستیک احتمال اینکه یک ورودی داده شده به یک دسته خاص تعلق داشته باشد را مدل می‌کند. این کار با استفاده از تابع لجستیک یا تابع سیگموئید انجام می‌شود که یک مقدار بین 0 و 1 خروجی می‌دهد. تابع لجستیک بر روی ترکیب خطی از ویژگی‌های ورودی اعمال می‌شود. ضرایب معادله خطی (مشابه وزن‌ها در شبکه‌های عصبی) از داده‌های آموزشی، معمولاً با استفاده از یک روش مانند برآورد حداکثر احتمال، یاد گرفته می‌شوند.

```
18] model1 = LogisticRegression(solver='sag', max_iter=200, random_state=24)
    model1.fit(x_train, y_train)
    y_hat = model1.predict(x_test)
    print(y_hat)
    print(y_test)

    [1 0 3 0 0 2 3 2 2 1 0 0 2 3 1 0 1 0 2 3 1 3 1 0 3 2 1 2 2 0 1 1 2 0 1 1 1
     3 1 1 3 1 0 0 2 3 0 3 0 0 0 0 2 3 2 3 0 2 0 3 2 3 0 0 0 3 3 3 0 3 1 0 0 2
     3 1 2 2 1 3 2 3 2 2 2 0 3 0 0 3 1 2 0 3 3 1 2 2 0 2 3 0 2 2 1 3 2 3 1 1 1
     0 1 1 1 1 3 3 2 1 0 3 1 1 3 0 2 3 1 0 1 3 0 3 2 0 0 2 0 3 1 0 2 0 1 2 3 1
     0 1 0 2 3 1 2 1 2 2 0 3 0 2 0 3 0 3 0 0 3 2 1 0 0 3 3 2 2 2 3 2 0 3 0 3 3
     2 2 1 3 3 1 3 0 2 2 1 1 3 3 2]
    [1 0 3 0 0 2 3 2 2 1 0 0 2 3 1 0 1 0 2 3 1 3 1 0 3 2 1 2 2 0 1 1 2 0 1 1 1
     3 1 1 3 1 0 0 2 3 0 3 0 0 0 0 2 3 2 3 0 2 0 3 2 3 0 0 0 3 3 3 0 3 1 0 0 2
     2 1 2 2 1 3 2 3 2 2 2 0 3 0 0 3 1 2 0 3 3 1 2 2 0 2 3 0 2 2 1 3 2 3 1 1 1
     0 1 1 1 1 3 3 2 1 0 3 1 1 3 0 2 3 1 0 1 3 0 3 2 0 0 2 0 3 1 0 2 0 1 2 3 1
     0 1 0 2 3 1 2 0 2 2 0 3 0 2 0 3 0 3 0 0 3 2 1 0 0 3 3 2 3 2 3 2 0 3 0 3 3
     2 2 1 3 3 1 3 0 2 2 1 1 3 3 2]
```

پارامتر max_iter ماکسیمم تکرر برای solver است تا بتواند به خوبی مدل رگرسیون لجیستسک را فیت کند ، دقت پیشبینی را بهینه کند و... و خود تابع بهترین iteration را محاسبه می کند و نیازی به تنظیم آن نیست .

همجنین پارامتر solver=’sag’ ، همان الگوریتم بهینه سازی ما است که در این کد SAG یا Stochastic average gradient descent برحسب پیش فرض انتخاب شده است.

```
train_score = model1.score(x_train, y_train)
test_score = model1.score(x_test, y_test)
acc = accuracy_score(y_test,y_hat)
con = confusion_matrix(y_test,y_hat)
print(acc)
print(con)
print ("train score is :" , train_score)
print ("test score is :" , test_score)
```

```
0.985
[[54  1  0  0]
 [ 0 43  0  0]
 [ 0  0 48  1]
 [ 0  0  1 52]]
train score is : 0.97125
test score is : 0.985
```

نتیجه دقت تخمین داده های تست و ترین را مشاهده می کنیم که عدد 98% و 97% است که تقریب خوبی است همچنین confusion matrix داده های تست را مشاهده میکنیم که اعداد روی قطر اصلی داده های درست تخمین زده شده برای هر کلاس است و در قطر غیر اصلی داده های اشتباه تخمین زده شده را می بینیم.

ب) Perceptron :

پرسپترون یکی از ساده‌ترین انواع شبکه‌های عصبی مصنوعی است و می‌توان آن را بلوک ساختمانی برای شبکه‌های پیچیده‌تر در نظر گرفت. پرسپترون نوعی طبقه‌بند خطی است، به این معنا که پیش‌بینی‌های خود را بر اساس یک تابع خطی انجام می‌دهد که وزن‌ها را با بردار ویژگی ترکیب می‌کند.

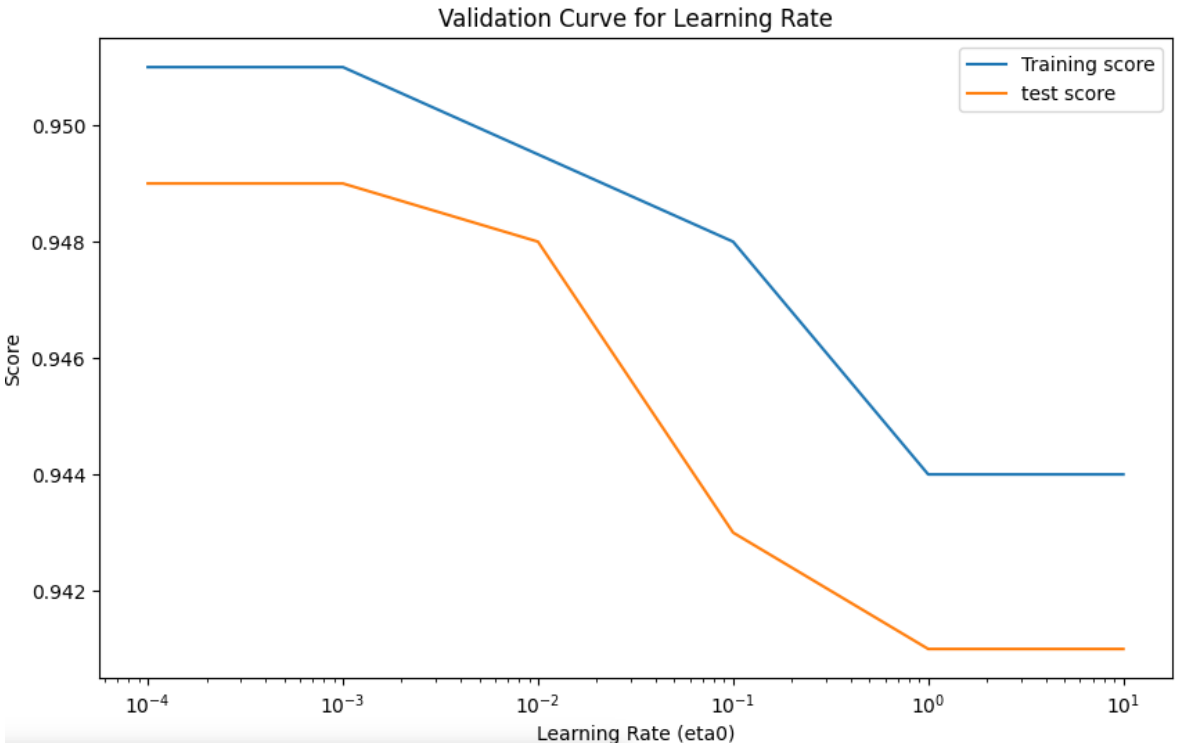
پرسپترون شامل مقادیر ورودی، وزن‌ها، یک سوگیری (یا آستانه) و یک تابع فعال‌سازی است. هر ورودی در وزن خود ضرب می‌شود و جمع این محصولات به علاوه سوگیری، ورودی به تابع فعال‌سازی را تشکیل می‌دهد. قانون یادگیری پرسپترون بر اساس خطای بدست آمده در پیش‌بینی‌ها، که تفاوت بین خروجی مورد انتظار و واقعی است، وزن‌ها و سوگیری را به‌روز می‌کند. این قانون به طور تکراری بر روی مجموعه آموزشی اعمال می‌شود.

با استفاده از دستور SGDClassifier() و loss='perceptron' ، classifier پرسپترون با الگوریتم بهینه سازی stochastic gradient descent انتخاب میکنیم .

هایپر پارامتر های ما نرخ آموزش و تعداد تکرر است که در اینجا همانند قسمت قبل بصورت max_iter امده که بدین معنا است که تعداد iteration بهینه در خود تابع انتخاب می شود و ما فقط باید تعداد حد اکثر تکرر را به اندازه ی کافی بزرگ بگیریم تا همگرایی رخ دهد .

در مورد نرخ آموزش ما میتوانیم نمودار دقت به ازای نرخ آموزش را برای داده های تست و آموزش بکشیم برای اینکار از دستور validation_curve() استفاده کرده ایم:

لازم به ذکر است که در این تابع از تقسیم تست و ترین درون خود تابع استفاده شده و با split ما کاری ندارد.



می بینیم که نرخ آموزش 100.0 عدد مطلوبی است .

```
model3 = SGDClassifier(loss="perceptron"
                       , eta0=0.001
                       , learning_rate="constant"
                       , penalty=None
                       , random_state=24
                       , max_iter=20
                       )
model3.fit(x_train , y_train)
y_hat = model3.predict(x_test)
print(y_hat)
print(y_test)
```

```
[1 0 3 0 2 2 3 2 2 1 0 0 2 3 1 0 1 0 2 3 1 3 1 0 3 2 1 2 2 0 1 1 2 0 1 1 1
 3 1 1 3 1 0 0 2 3 0 3 0 2 0 0 2 3 2 3 0 2 0 2 2 3 0 0 0 3 3 3 0 3 1 0 0 2
 3 1 2 2 1 3 2 3 2 2 2 0 3 0 0 3 1 2 0 3 3 1 2 2 0 2 3 0 2 2 1 3 2 3 1 1 1
 0 1 1 1 1 3 3 2 1 0 3 1 1 3 0 2 3 1 0 1 3 0 3 2 0 0 2 0 3 1 0 2 0 1 2 3 1
 0 1 0 2 3 1 2 1 2 2 0 3 0 2 0 3 0 3 0 0 3 2 1 0 0 3 3 2 2 2 3 2 0 3 0 3 3
 2 2 1 3 3 1 3 0 2 2 1 1 3 3 2]
[1 0 3 0 0 2 3 2 2 1 0 0 2 3 1 0 1 0 2 3 1 3 1 0 3 2 1 2 2 0 1 1 2 0 1 1 1
 3 1 1 3 1 0 0 2 3 0 3 0 0 0 0 2 3 2 3 0 2 0 3 2 3 0 0 0 3 3 3 0 3 1 0 0 2
 2 1 2 2 1 3 2 3 2 2 2 0 3 0 0 3 1 2 0 3 3 1 2 2 0 2 3 0 2 2 1 3 2 3 1 1 1
 0 1 1 1 1 3 3 2 1 0 3 1 1 3 0 2 3 1 0 1 3 0 3 2 0 0 2 0 3 1 0 2 0 1 2 3 1
 0 1 0 2 3 1 2 0 2 2 0 3 0 2 0 3 0 3 0 0 3 2 1 0 0 3 3 2 3 2 3 2 0 3 0 3 3
 2 2 1 3 3 1 3 0 2 2 1 1 3 3 2]
```

```
train_score = model3.score(x_train, y_train)
test_score = model3.score(x_test, y_test)
acc = accuracy_score(y_test,y_hat)
con = confusion_matrix(y_test,y_hat)
print(acc )
print(con)
print ("train score is :" , train_score)
print ("test score is :" , test_score)
```

```
0.97
[[52  1  2  0]
 [ 0 43  0  0]
 [ 0  0 48  1]
 [ 0  0  2 51]]
train score is : 0.9375
test score is : 0.97
```

همچنین دقت تست و آموزش را مشاهده می کنیم که97% و 937% اند که تقریب خوبی است و همچنین confusion matrix را نیز نشان داده ایم.

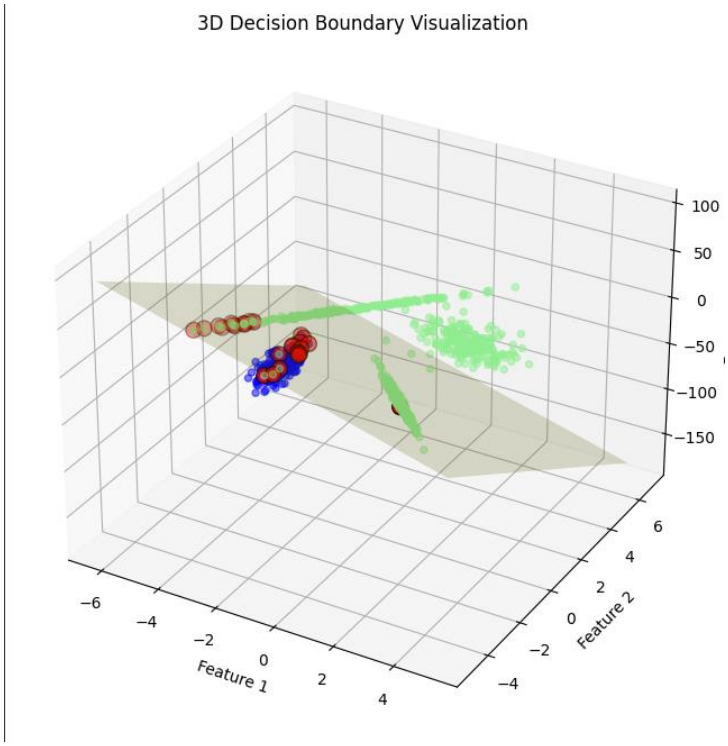
4 .

چون 3 ویژگی داریم ، برای رسم مرز و نواحی تصمیم گیری شمای سه بعدی نیاز داریم بسیار پیچیده می شود .

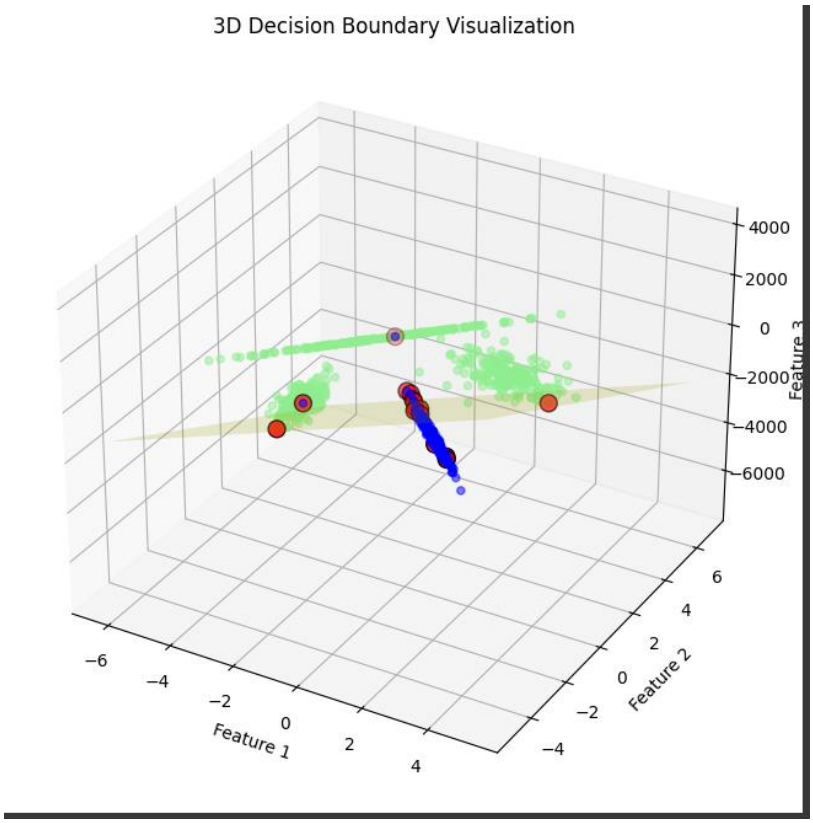
برای همین ما یک بار رسم سه بعدی برای نواحی ،رسم سه بعدی یک کلاس نسبت به باقی کلاس ها و یک بار با PCA کاهش ابعاد داده و دو بعدی را با داده های misclassed هایلایت شده و نشده رسم کرده ایم:

الف) Logistic Regression

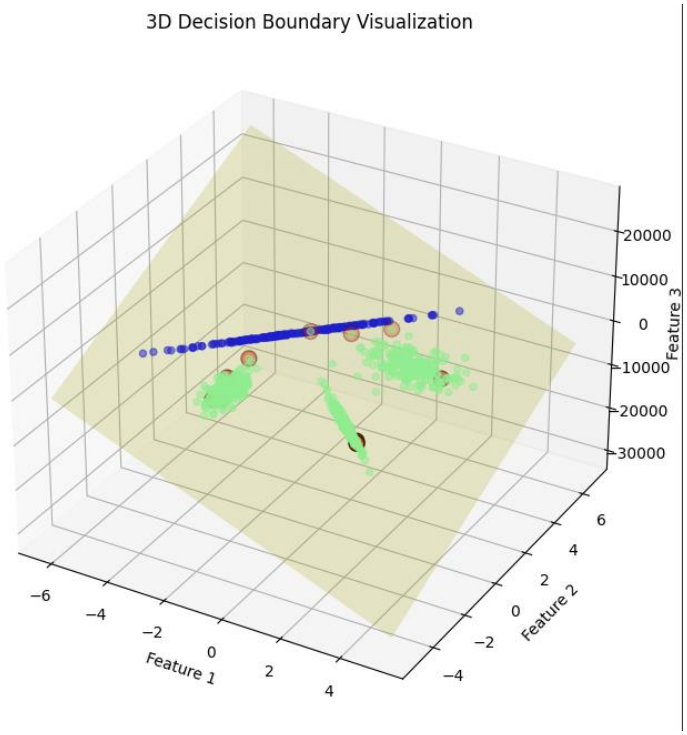
کلاس 1



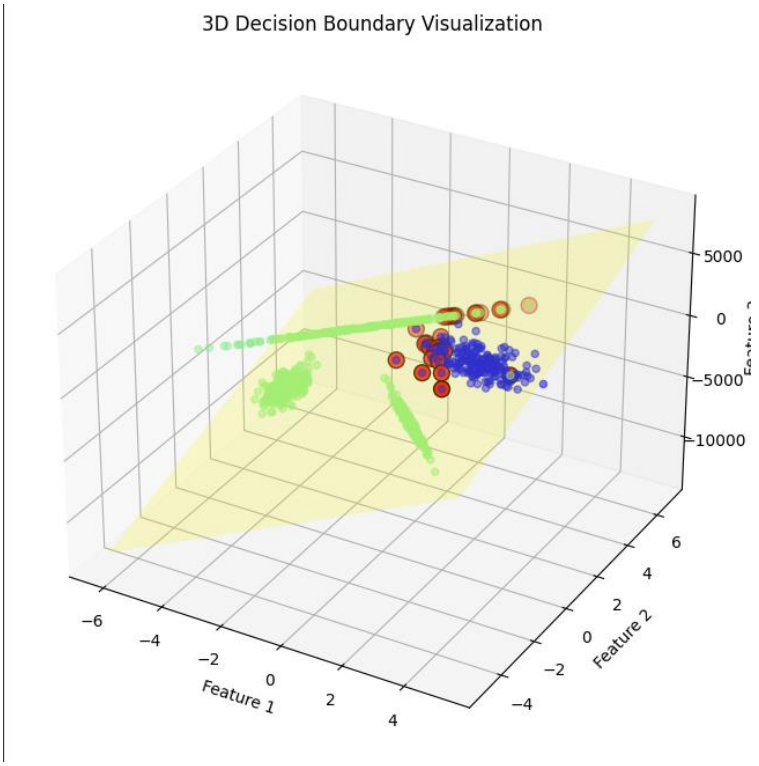
کلاس 2


3D Decision Boundary Visualization
Feature 1
Feature 2
Feature 3

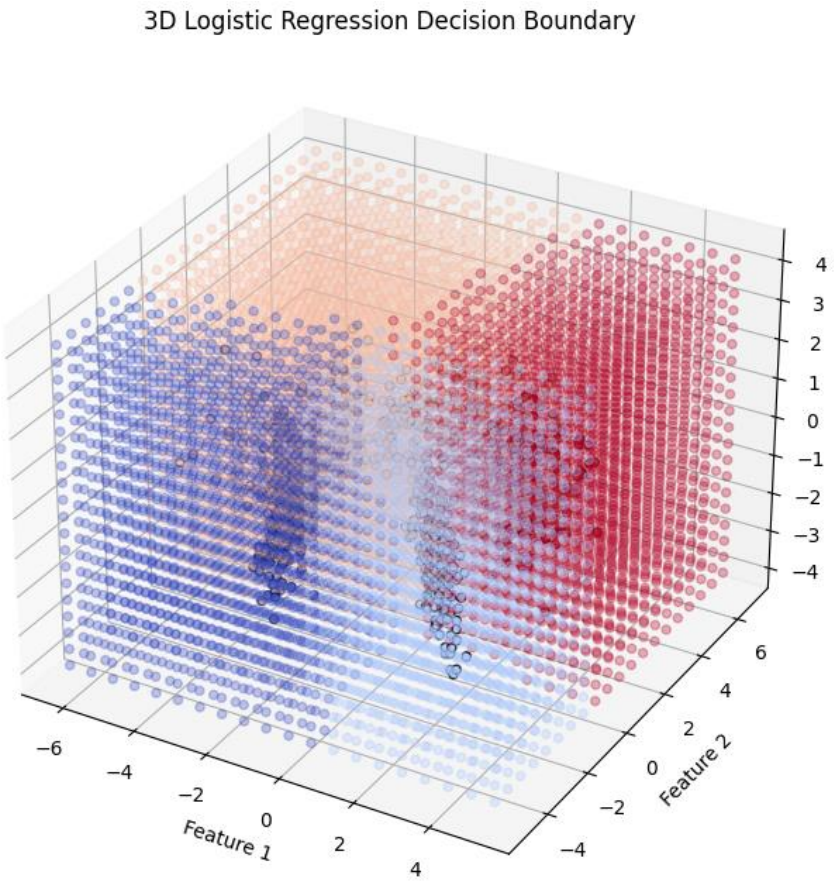
کلاس 3


3D Decision Boundary Visualization
Feature 1
Feature 2
Feature 3

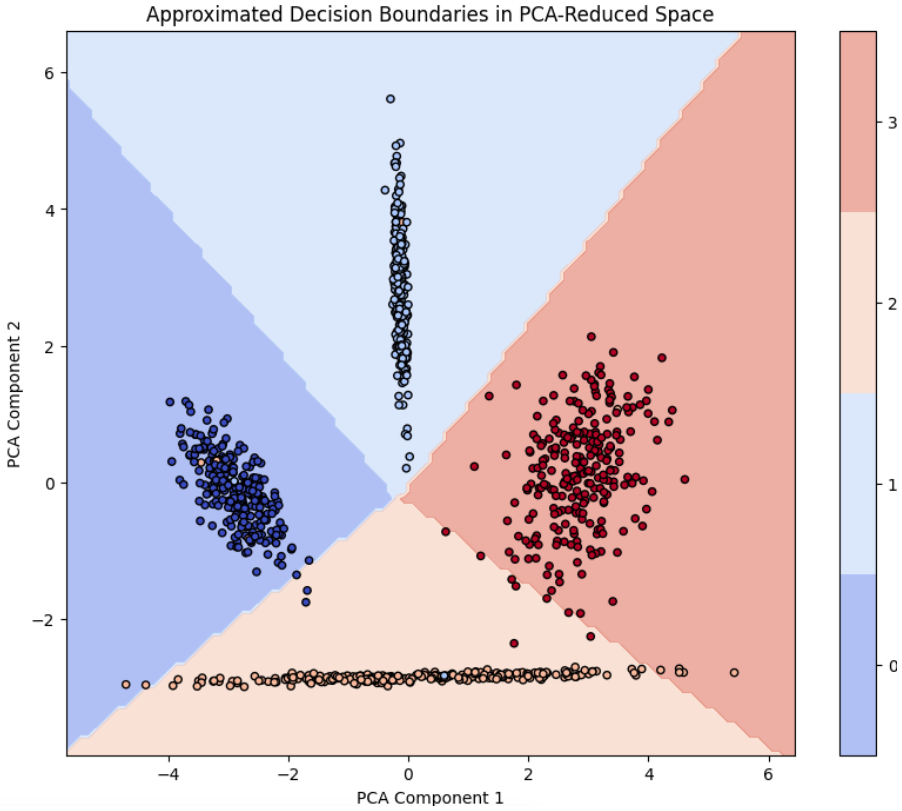
کلاس 4



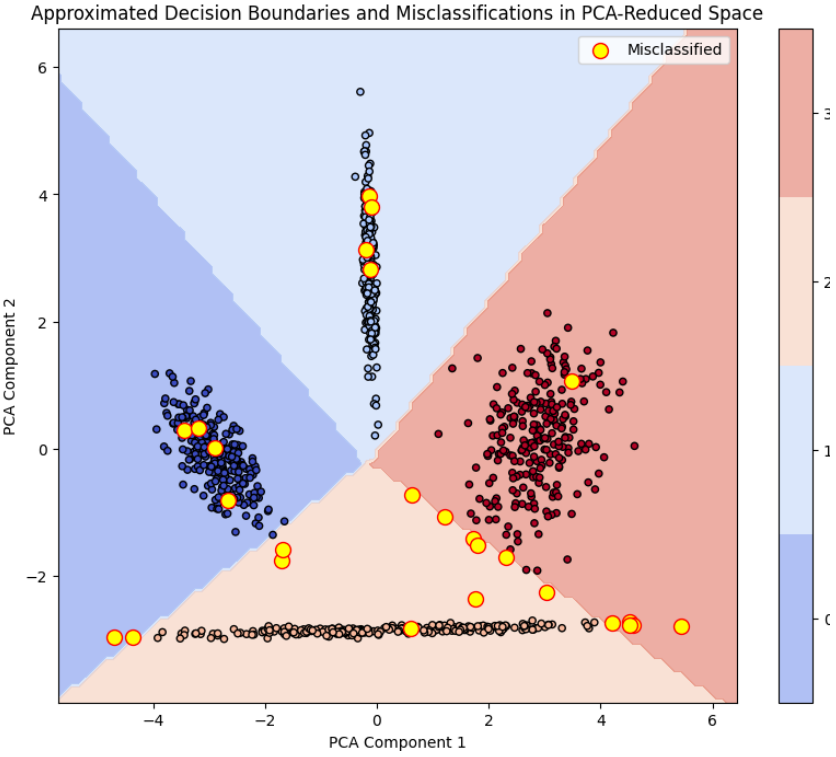
تمام کلاس ها



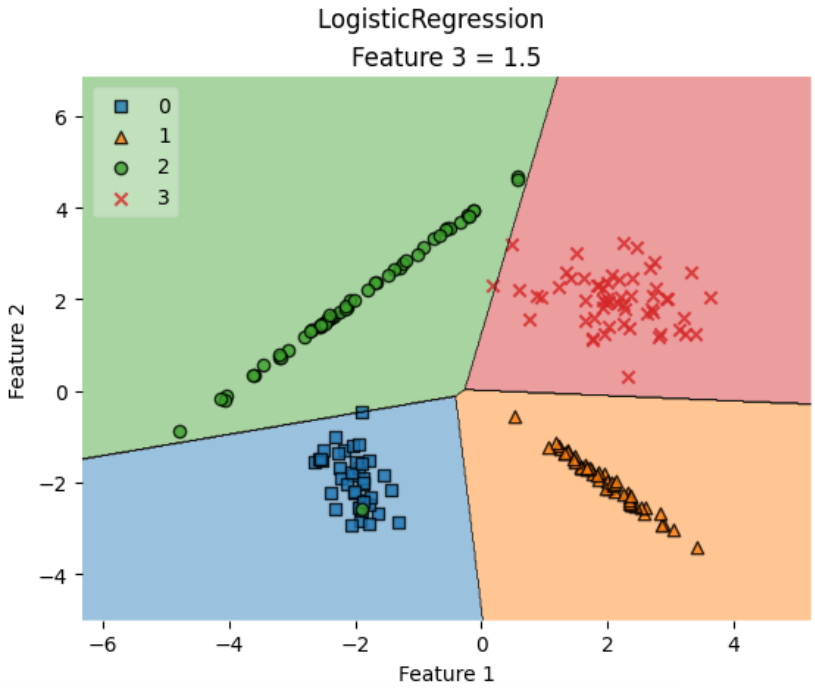
رسم دو بعدی مرز تصمیم گیری (ویژگی 1 و 2)



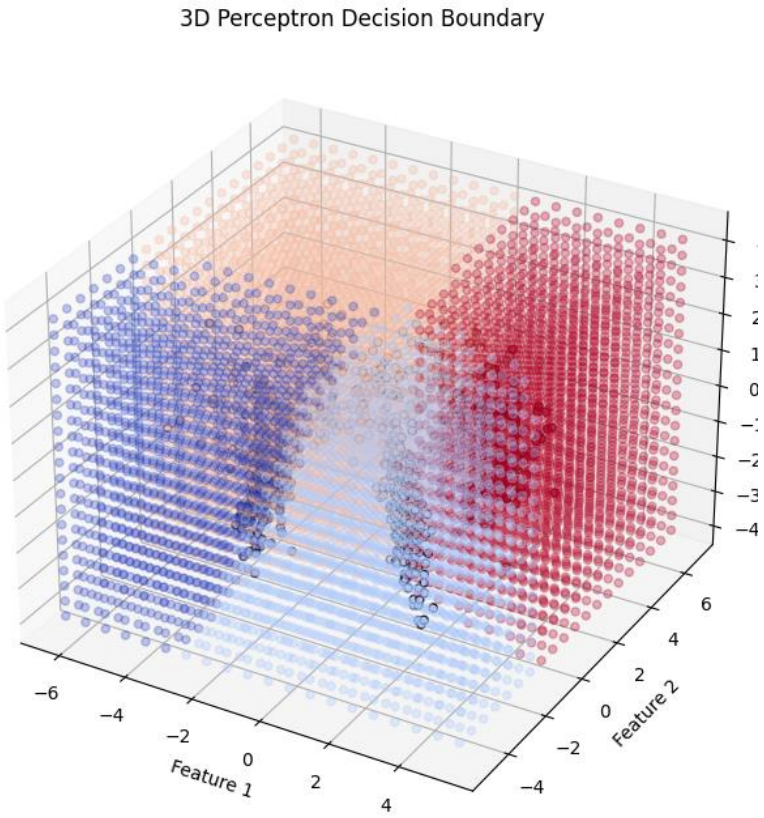
حال داده های misclassed را هایلایت کردیم:



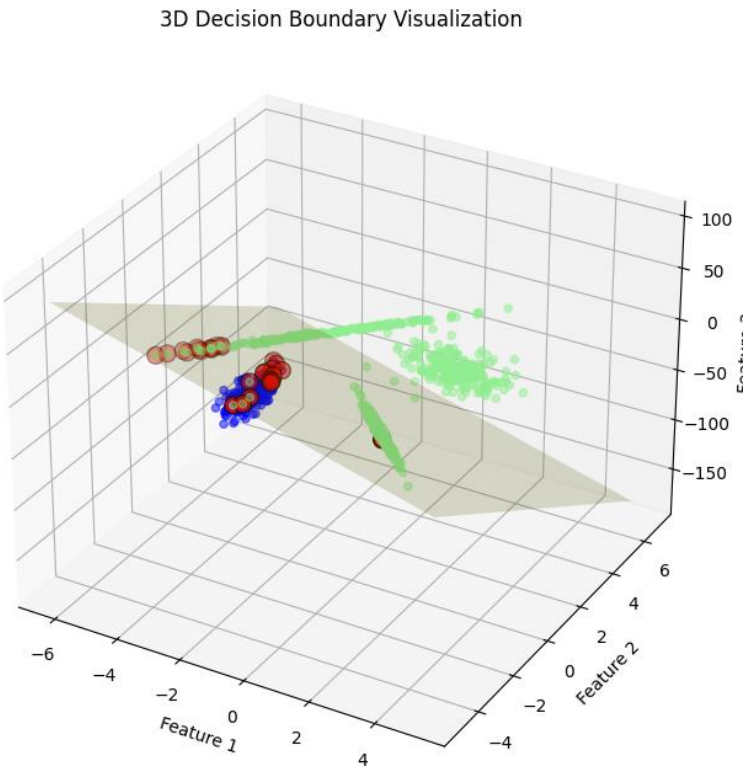
با استفاده از دستور mlxtend :



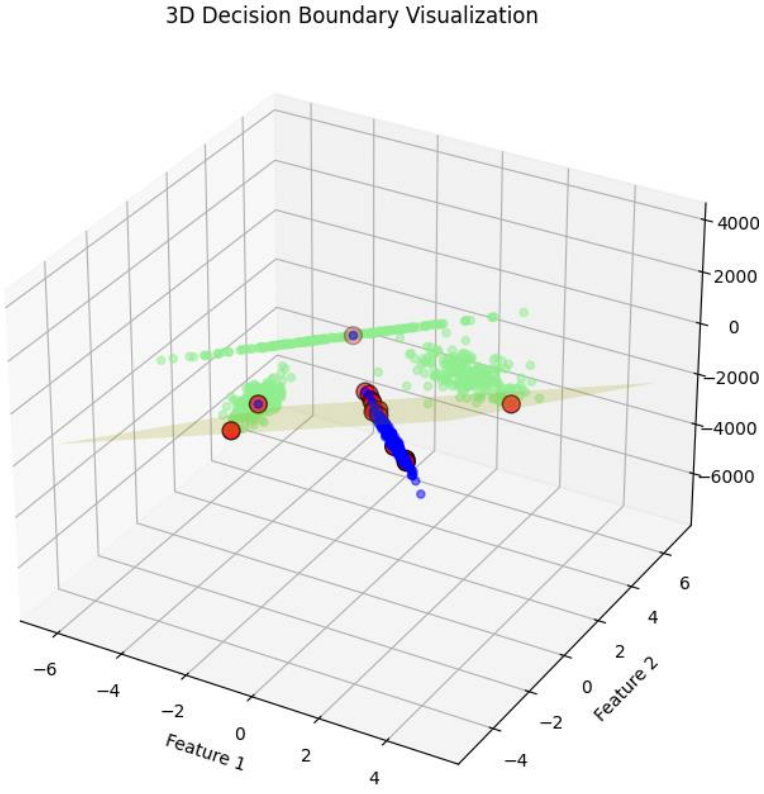
ب) perceptron :



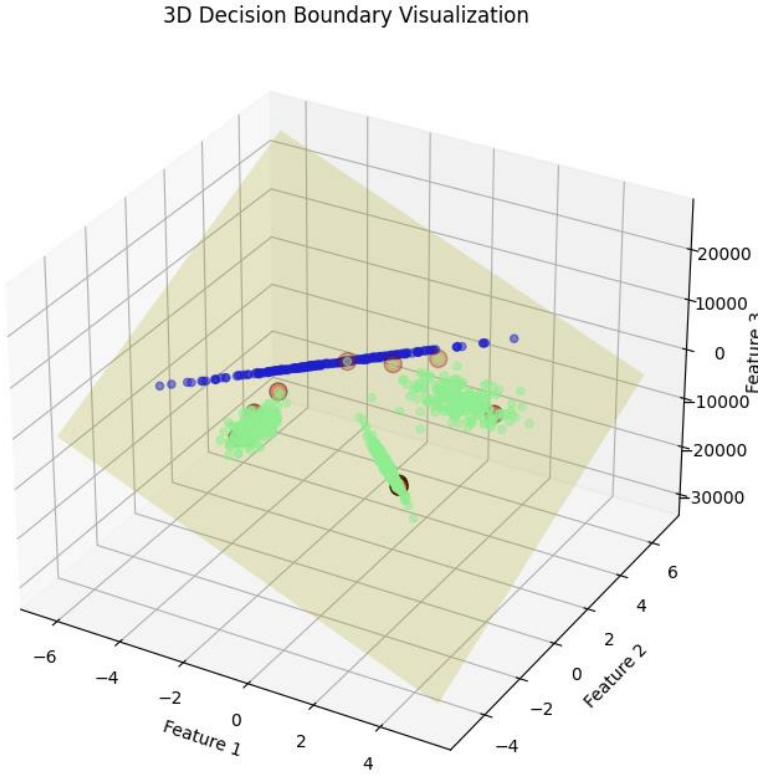
کلاس 1


3D Decision Boundary Visualization
100
50
0
−50
−100
−150
−6
−4
−2
0
2
4
Feature 1
6
4
2
0
−2
−4
Feature 2

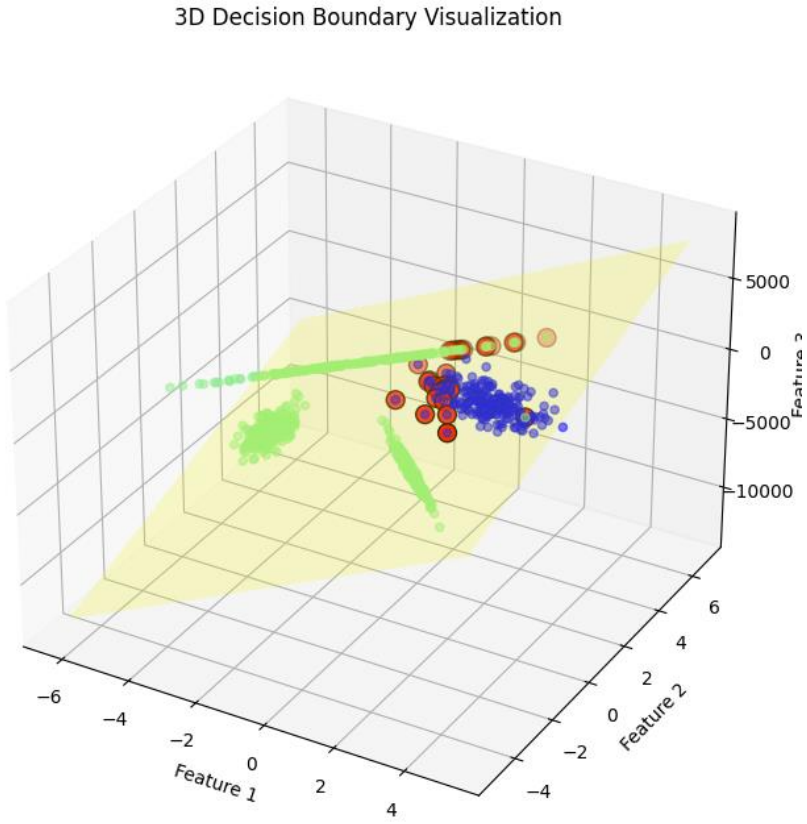
کلاس 2


3D Decision Boundary Visualization
4000
2000
0
−2000
−4000
−6000
−6
−4
−2
0
2
4
Feature 1
6
4
2
0
−2
−4
Feature 2

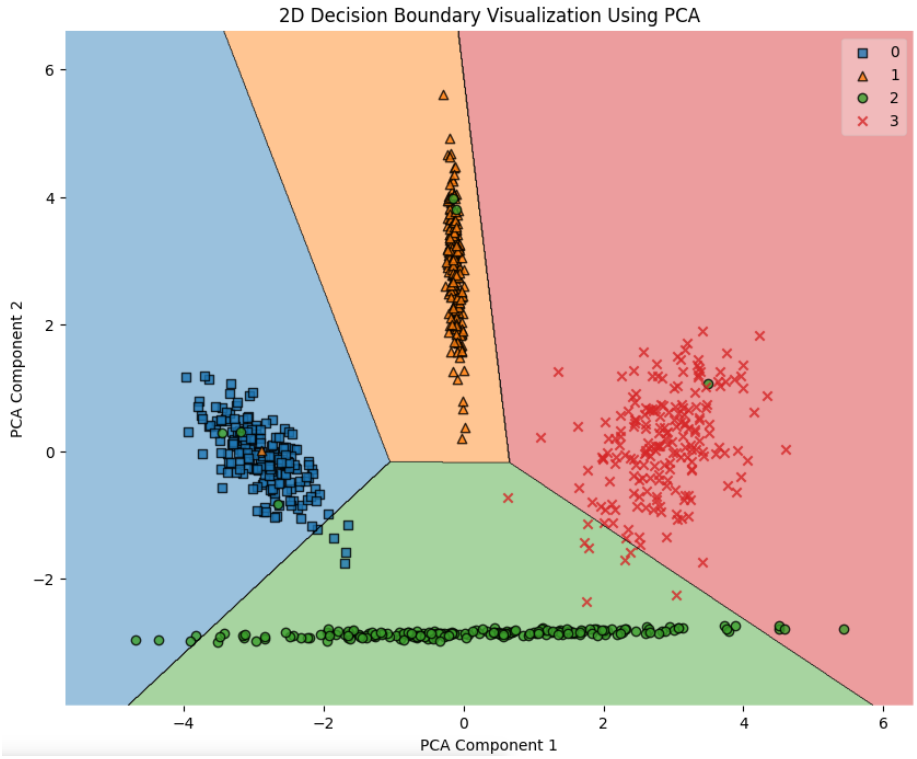
کلاس 3


3D Decision Boundary Visualization
20000
10000
0
10000
20000
30000
Feature 3
−6
−4
−2
0
2
4
Feature 1
6
4
2
0
−2
−4
Feature 2

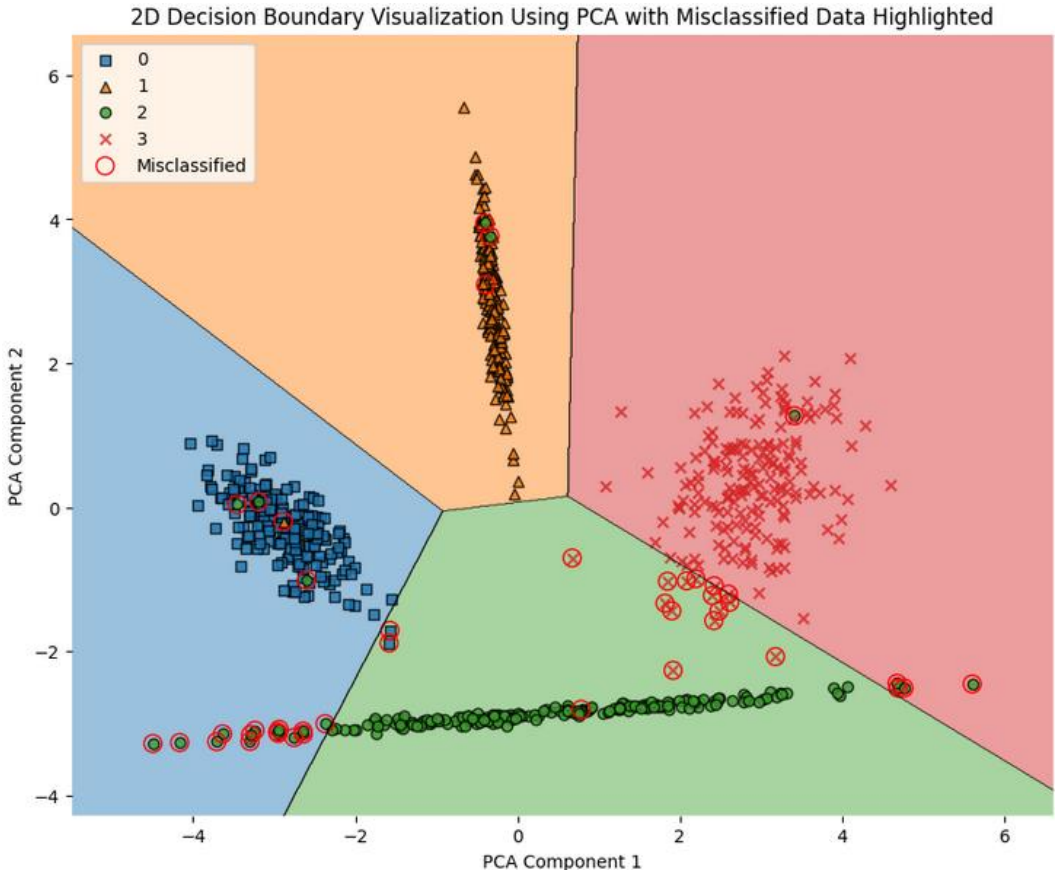
کلاس 4


3D Decision Boundary Visualization
5000
0
−5000
−10000
Feature 3
−6
−4
−2
0
2
4
Feature 1
6
4
2
0
−2
−4
Feature 2

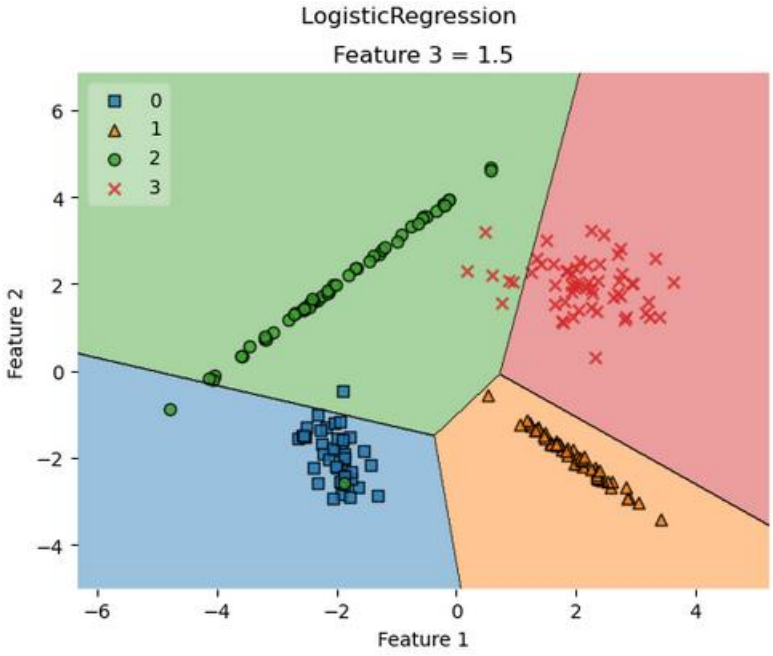
رسم دو بعدی :



با داده های misclassedهایلایت شده:



با mlxtend :



5 .

توسط کتابخانه ی drawdata و دستور draw_scatter() دیتاستی تولید می کنیم (متاسفانه عکسش پاک شده وقتی دوباره باز کردم)

دیتاست را فرا میخوانیم و لیبل های abc را به 0و1و2 تبدیل کرده وسپس با دستور standardscalar() آن ها را استاندارد سازی می کنیم و همینطور با دستور shuffle داده ها را بر میزنیم

```
(array([[299.99263049, 417.58265835],
        [345.42433552,  85.74487736],
        [288.39522559, 245.45239811],
        ...,
        [179.83421648,  59.36045824],
        [393.11871805, 128.4647334 ],
        [296.73186941, 125.66792505]]),
 array([[ 0.20295074,  1.39198802],
        [ 0.57488633, -1.23561846],
        [ 0.1080063 ,  0.029001  ],
        ...,
        [-0.78074976, -1.44453943],
        [ 0.96534577, -0.89734784],
        [ 0.17625588, -0.91949394]]))
```

. سپس با داده هارا به تست وآموزش با ضریب 0.2 تقسیم می کنیم .

```
x_train.shape, y_train.shape, x_test.shape, y_test.shape ,
```

```
((717, 2), (717,), (180, 2), (180,))
```

با استفاده از مدل logistic regression به تفکیک داده ها می پردازیم:

```
model4 = LogisticRegression(solver='saga', max_iter=200, random_state=24)
model4.fit(x_train, y_train)


y_hat = model4.predict(x_test)
print(y_hat)
print(y_test)
```

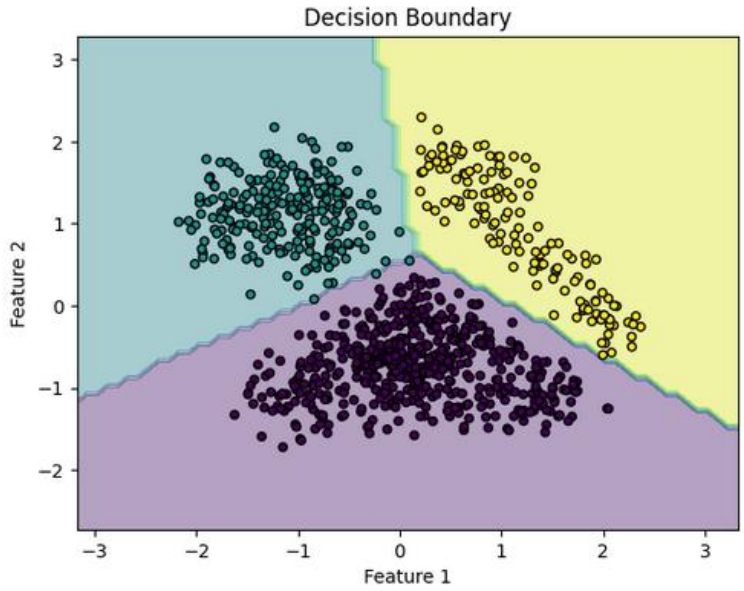
```
[0 0 0 2 1 0 0 1 1 1 0 2 2 1 0 0 1 2 2 0 0 0 2 0 0 0 0 0 1 0 2 0 1 1 0 2 0
 0 0 1 2 2 2 0 0 0 2 0 0 0 1 2 0 0 2 1 1 1 1 0 0 0 0 1 1 1 0 2 0 0 0 0 0 0
 0 0 0 2 2 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 2 1 0 1 2 0 2 1 1 0 0 0 0 1 1 0 1
 0 0 0 1 0 1 1 0 0 0 0 0 2 1 0 0 0 1 1 1 0 0 2 1 0 0 1 2 0 0 0 0 0 0 0 0 0
 0 0 0 2 2 0 0 0 0 0 2 1 0 0 2 0 0 1 0 1 0 1 1 0 0 1 1 2 0 0 0 0]
[0 0 0 2 1 0 0 1 1 1 0 2 2 1 0 0 1 2 2 0 0 0 2 0 0 0 0 0 1 0 2 0 1 1 0 2 0
 0 0 1 2 2 2 0 0 0 2 0 0 0 1 2 0 0 2 1 1 1 1 0 0 0 0 1 1 1 0 2 0 0 0 0 0 0
 0 0 0 2 2 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 2 1 0 1 2 0 2 1 1 0 0 0 0 1 1 0 1
 0 0 0 1 0 1 1 0 0 0 0 0 2 1 0 0 0 1 1 1 0 0 2 1 0 0 1 2 0 0 0 0 0 0 0 0 0
 0 0 0 2 2 0 0 0 0 0 2 1 0 0 2 0 0 1 0 1 0 1 1 0 0 1 1 2 0 0 0 0]
```

نتیجه :

```
1.0
[[103   0   0]
 [  0  49   0]
 [  0   0  28]]
train score is : 0.99581589958159
test score is : 1.0
```

میبینیم که دقت تست و آموزش تقریبا بوده و همینطور که از confusion matrix نیز پیداست تمام داده ها به خوبی تخمین زده شده اند و این به دلیل مدل ساده بود.

رسم مرز تصمیم گیری:



همچنین برای perceptron داریم:

```
model5 = SGDClassifier(loss="perceptron"
                       , eta0=0.001
                       , learning_rate="constant"
                       , penalty=None
                       , random_state=24
                       , max_iter=200
                       )
model5.fit(x_train , y_train)

y_hat = model5.predict(x_test)
print(y_hat)
print(y_test)
```

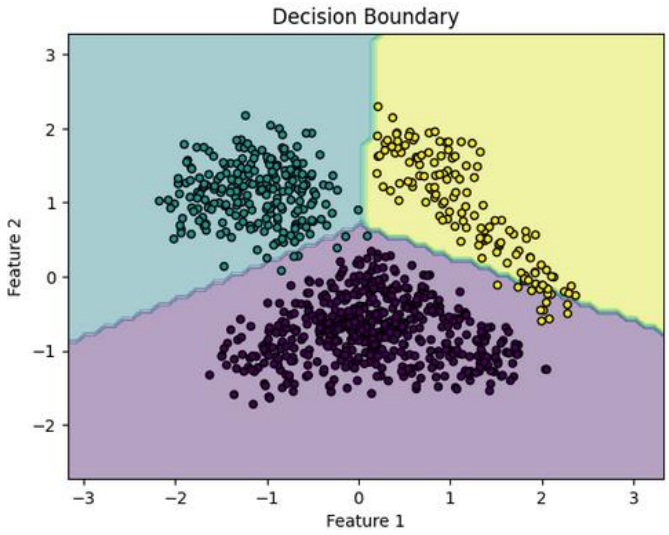
```
[0 0 0 2 1 0 0 1 1 1 0 2 2 1 0 0 1 2 2 0 0 0 2 0 0 0 0 0 1 0 2 0 1 1 0 2 0
 0 0 1 2 2 2 0 0 0 0 0 0 0 1 2 0 0 2 1 1 1 1 0 0 0 0 1 1 1 0 2 0 0 0 0 0 0
 0 0 0 2 2 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 2 1 0 1 2 0 2 1 1 0 0 0 0 1 1 0 1
 0 0 0 1 0 1 1 0 0 0 0 0 2 1 0 0 0 1 1 1 0 0 2 1 0 0 1 2 0 0 0 0 0 0 0 0 0
 0 0 0 2 2 0 0 0 0 0 2 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 2 0 0 0 0]
[0 0 0 2 1 0 0 1 1 1 0 2 2 1 0 0 1 2 2 0 0 0 2 0 0 0 0 0 1 0 2 0 1 1 0 2 0
 0 0 1 2 2 2 0 0 0 2 0 0 0 1 2 0 0 2 1 1 1 1 0 0 0 0 1 1 1 0 2 0 0 0 0 0 0
 0 0 0 2 2 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 2 1 0 1 2 0 2 1 1 0 0 0 0 1 1 0 1
 0 0 0 1 0 1 1 0 0 0 0 0 2 1 0 0 0 1 1 1 0 0 2 1 0 0 1 2 0 0 0 0 0 0 0 0 0
 0 0 0 2 2 0 0 0 0 0 2 1 0 0 2 0 0 1 0 1 0 1 1 0 0 1 1 2 0 0 0 0]
```

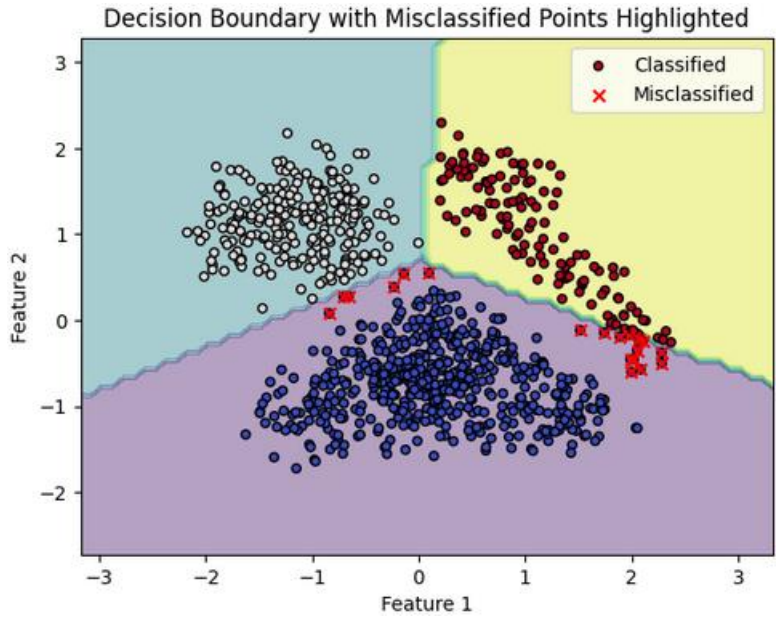
```
train_score = model5.score(x_train, y_train)
test_score = model5.score(x_test, y_test)
acc = accuracy_score(y_test,y_hat)
con = confusion_matrix(y_test,y_hat)
print(acc)
print(con)
print ("train score is :" , train_score)
print ("test score is :" , test_score)
```

```
0.9888888888888889
[[103   0   0]
 [  0  49   0]
 [  2   0  26]]
train score is : 0.9762900976290098
test score is : 0.9888888888888889
```

می بینیم که از دقت تست و ترین 99 و 97.5درصد هست.

مرز تصمیم گیری :


Decision Boundary
Feature 1
Feature 2


Decision Boundary with Misclassified Points Highlighted
Classified
Misclassified
Feature 1
Feature 2

سوال 2

1 .

داده‌های آزمایشی برای بلبرینگ‌های نرمال و معیوب را فراهم شده است . آزمایش‌ها با استفاده از یک موتور الکتریکی ۲ اسب بخار Reliance انجام شد و داده‌های شتاب در مکان‌های نزدیک و دور از بلبرینگ‌های موتور اندازه‌گیری شد. شرایط آزمایش واقعی موتور و همچنین وضعیت خطای بلبرینگ برای هر آزمایش با دقت مستند شده است.

بلبرینگ‌های موتور با استفاده از ماشین‌کاری تخلیه الکتریکی (EDM) بلبرینگ‌های موتور با استفاده از ماشین‌کاری تخلیه الکتریکی (EDM) با عیوب تخمیر شده اند. عیوب با قطرهای ۰.۰۰۷ اینچ تا ۰.۰۴۰ اینچ به طور جداگانه در مسیر ریسوایی داخلی، عنصر نورد (یعنی توپ) و مسیر ریسوایی بیرونی معرفی شده اند. بلبرینگ‌های معیوب مجدداً در موتور آزمایشی نصب شدند و داده‌های ارتعاش برای بارهای موتور ۰ تا ۳ اسب بخار (سرعت موتور ۱۷۲۰ تا ۱۷۹۷ دور در دقیقه) ثبت شد.

اطلاعات برای بلبرینگ‌های نرمال، عیب در سمت درایو نقطه‌ای و سمت فن جمع‌آوری شده اند. داده‌ها با نرخ نمونه‌برداری ۱۲،۰۰۰ نمونه در ثانیه و ۴۸،۰۰۰ نمونه در ثانیه برای آزمایش‌های بلبرینگ سمت درایو جمع‌آوری شده است. تمام داده‌های بلبرینگ سمت فن با نرخ نمونه‌برداری ۱۲،۰۰۰ نمونه در ثانیه جمع‌آوری شده اند .

فایل‌های داده به فرمت متلب است. هر فایل شامل داده‌های ارتعاشات سمت فن و درایو همراه با سرعت چرخش موتور است. برای تمام فایل‌ها، مورد زیر در نام متغیر نشان می‌دهد:

DE - داده‌های شتاب‌سنج سمت درایو

FE - داده‌های شتاب‌سنج سمت فن

BA - داده‌های شتاب‌سنج پایه

time - داده‌های سری زمانی

RPM - دور در دقیقه در طول آزمایش

داده‌های پایه نرمال ، داده‌های عیب بلبرینگ سمت درایو با نرخ k12 ، داده‌های عیب بلبرینگ ، سمت درایو با نرخ k48 ، داده‌های عیب بلبرینگ سمت فن

با توجه به خواسته های سوال از داده های نرمال با سرعت (1797rmp) وداده های فالت با قطرهای ۰.۰۰۷ اینچ مسیر رینگ داخلی استفاده میکنیم .

```
[('X097_DE_time', (243938, 1), 'double'), ('X097_FE_time', (243938, 1), 'double'), ('X097RPM', (1, 1), 'double')]
[('X105_DE_time', (121265, 1), 'double'), ('X105_FE_time', (121265, 1), 'double'), ('X105_BA_time', (121265, 1), 'double'), ('X105RPM', (1, 1), 'double')]
```

در اینجا فیچر های مرتبط نوشته شده و بنا بر خواسته ی مساله ما فقط ازDE_time وBA_time استفاده کردیم

2 .

آ)

تابعی تعریف می کنیم که دیتارا گرفته آن را بر زند و نمونه با طول N جدا کند . سپس این تابع را به دیتاست اعمال می کنیم و از برای هرکلاس یک ماتریس 100*200 تشکیل میدهیم .

```
((100, 200), (100, 200))
```

ب)

فیچر های ذکر شده در کتابخانه های numpy و scypy موجود هستند آنها را درون یک کلاس تعریف میکنیم. و به ماتریس های قبل اعمال می کنیم حال یک داده با 14 feature و 200 نمونه داریم .همچنین داده ها را با تعریف تابعی لیبل می زنیم:

```
Combined Features Shape: (200, 14)
Labels: (200,)
```

ج)

شافل کردن داده‌ها (بر زدن)در یادگیری ماشین اهمیت زیادی دارد، زیرا جلوگیری می‌کند از مدل یاد بگیرد که تشابهات اتفاقی مبتنی بر ترتیب داده‌ها را به عنوان الگوهای معتبر در نظر بگیرد. این کار به تعمیم الگوها بین دسته‌های مختلف کمک می‌کند و خطر بیش‌برازش را با وارد کردن تصادفی بودن به فرآیند آموزش کاهش می‌دهد. به علاوه، شافل کردن مطمئن می‌شود که مدل به طور ناخودآگاه الگوهای خاص به ترتیب جمع‌آوری یا ترتیب داده‌ها را یاد نمی‌گیرد. به طور کلی، این کار به بهبود عملکرد و استحکام بهتر مدل کمک می‌کند

حال داده ها را استاندارد سازی می کنیم (برای بر زدن shuffle=True) :

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

x_train, x_test, y_train, y_test = train_test_split(X,y,
    test_size = 0.2,
    shuffle = True,
    random_state = 24
)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((160, 14), (40, 14), (160,), (40,))
```

د)

1 نرمال سازی مینیمم ماکسیمم:

این روش یکی از ساده‌ترین روش‌های نرمال‌سازی است و به این صورت عمل می‌کند که هر ویژگی را بین صفر و یک (یا هر دو عدد دیگری که تعیین می‌شود) مقیاس‌بندی می‌کند.

$$X_{norm} = \frac{X - X_{min}}{X_{min} - X_{min}}$$

این روش خصوصاً مفید است وقتی می‌خواهیم داده‌ها را به نحوی محدود کنیم که تاثیر نقاط دورافتاده کاهش یابد و همه ویژگی‌ها در یک محدوده قرار گیرند.

2 استاندارد سازی

استانداردسازی یک روش دیگر است که در آن داده‌ها به گونه‌ای تغییر می‌یابند که میانگین آن‌ها صفر و انحراف معیار آن‌ها یک می‌شود.

$$X_{std} = \frac{X - \mu}{\sigma}$$

استانداردسازی به ویژه زمانی مفید است که داده‌ها توزیع نرمال داشته باشند و برای الگوریتم‌هایی که حساس به مقیاس داده‌ها هستند، مانند ماشین بردار پشتیبان (SVM) یا رگرسیون خطی، مناسب است.

بله، می‌توانیم از داده‌های ارزیابی برای نرمال‌سازی استفاده کنیم، اما باید احتیاط کنید که از همان پارامترهایی که برای نرمال‌سازی مجموعه داده اصلی استفاده شده‌اند، استفاده شود. این کار معمولاً در صورتی موثر است که معمولاً مجموعه داده اصلی و داده‌های ارزیابی از یک توزیع مشابه تولید شوند.

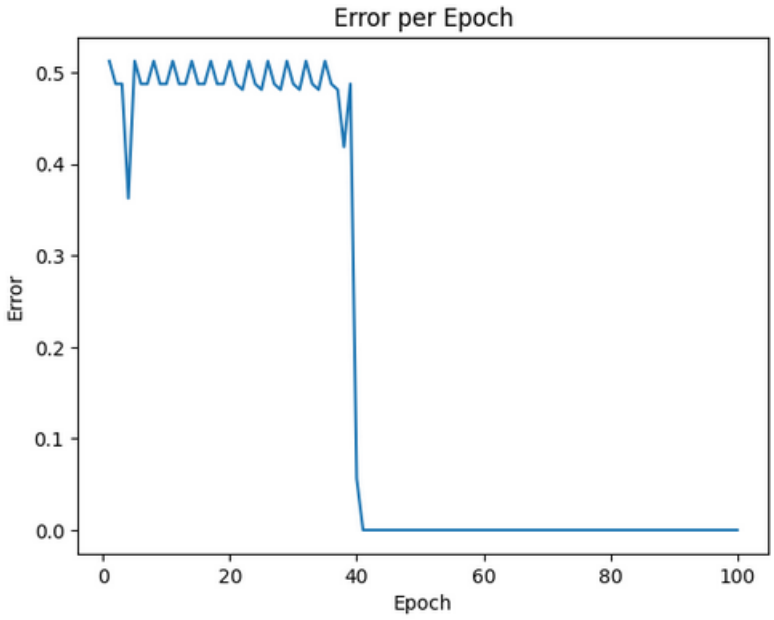
استفاده از همان پارامترهای نرمال‌سازی برای داده‌های ارزیابی که برای داده‌های آموزش استفاده شده‌اند، به ما کمک می‌کند که مدل‌های آموزش دیده بر روی داده‌های آموزش را بهتر به داده‌های ارزیابی تطبیق دهیم. این کمک می‌کند که مدل در مقابل اعمال نامعلوم (داده‌های جدید) بهتر عمل کند و عملکرد آن را بهبود بخشیم.

به طور کلی، استفاده از یک توزیع مشترک برای نرمال‌سازی داده‌های آموزش و ارزیابی، به ما کمک می‌کند تا مدلی را آموزش دهیم که به‌طور موثر بر روی داده‌های جدید کار کند.

3 .

با تعریف یک کلاس بدون استفاده از کتابخانه ،مدل logistic regression با تابع اتلافBCE والگوریتم یادگیری و ارزیابی GD در پایتون نوشتیم :

نمودار اتلاف :



برای ارزیابی عملکرد مدل‌های رگرسیون لجستیک، می‌توانید از دو شاخص متداول به نام‌های دقت (Accuracy) و ماتریس درهم‌ریختگی (Confusion Matrix) استفاده کنید.

1. دقت :(Accuracy) این شاخص نسبت تعداد پیش‌بینی‌های صحیح به کل نمونه‌ها را نشان می‌دهد :

2. ماتریس درهم‌ریختگی :(Confusion Matrix) این ماتریس نشان می‌دهد که چه تعداد نمونه از هر کلاس به درستی تشخیص داده شده‌اند و چه تعداد نمونه به اشتباه به هر کلاس تخصیص یافته‌اند.

با 3 معیار ارزیابی f1_score و دقت (Accuracy) و همچنین confusion matrix نتیجه را ارزیابی می کنیم.

```
accuracy(y_test,y_pred),confusion_matrix(y_test,y_pred),f1_score(y_test,y_pred)
```

```
(1.0,
 array([[18,  0],
        [ 0, 22]]),
 1.0)
```

می بینیم که دقت 100% شده است.

-خیر قبل از ارزیابی نمی توان به نمودار اتلاف اعتماد کرد زیرا امکان overtrain شدن وجود دارد یعنی مدل داده های اموزش را یادگرفته و شروع به یادگیری نویز مسکند و این باعث میشود که در مواجهه با داده های جدی عملکرد ضعیفی نشان دهد به همین دلیل باید از داده هی ارزیابی در کنار داده های آزمون استفاده کرد .

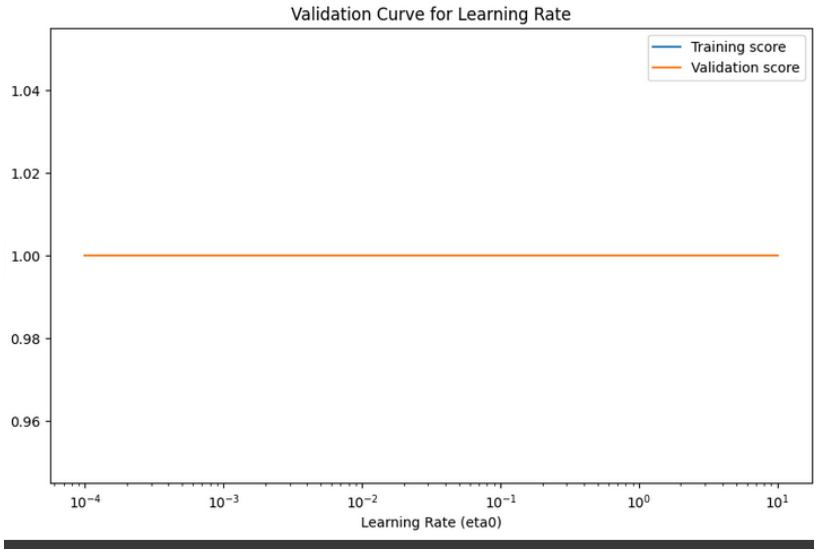
4 .

با استفاده از دستور SGDclassifier(solver =’log_loss’) میتوان همان کدی که ما نوشتیم را با استفاده از کتاب خانه نوشت.

نتایج را نشان میدهیم:

```
accuracy_score(y3_pred,y_test) , f1_score(y3_pred,y_test)
```

```
(1.0, 1.0)
```



نمایش نمودار اتلاف با استفاده از SGDclassifier() امکان پذیر نبود .

سوال 3

ابتدا داده ها را بارگزاری کرده و به فرمت مناسب در میآوریم . سپس اطلاعات مرتبط را برای فهم بهتر دیتاست استخراج می کنیم.

فیچر های ما بصورت زیر اند:

```
Index(['Formatted Date', 'Summary', 'Precip Type', 'Temperature (C)',
       'Apparent Temperature (C)', 'Humidity', 'Wind Speed (km/h)',
       'Wind Bearing (degrees)', 'Visibility (km)', 'Loud Cover',
       'Pressure (millibars)', 'Daily Summary'],
```

```
(96453, 12)
```

12 فیچر و 96453 نمونه داریم .

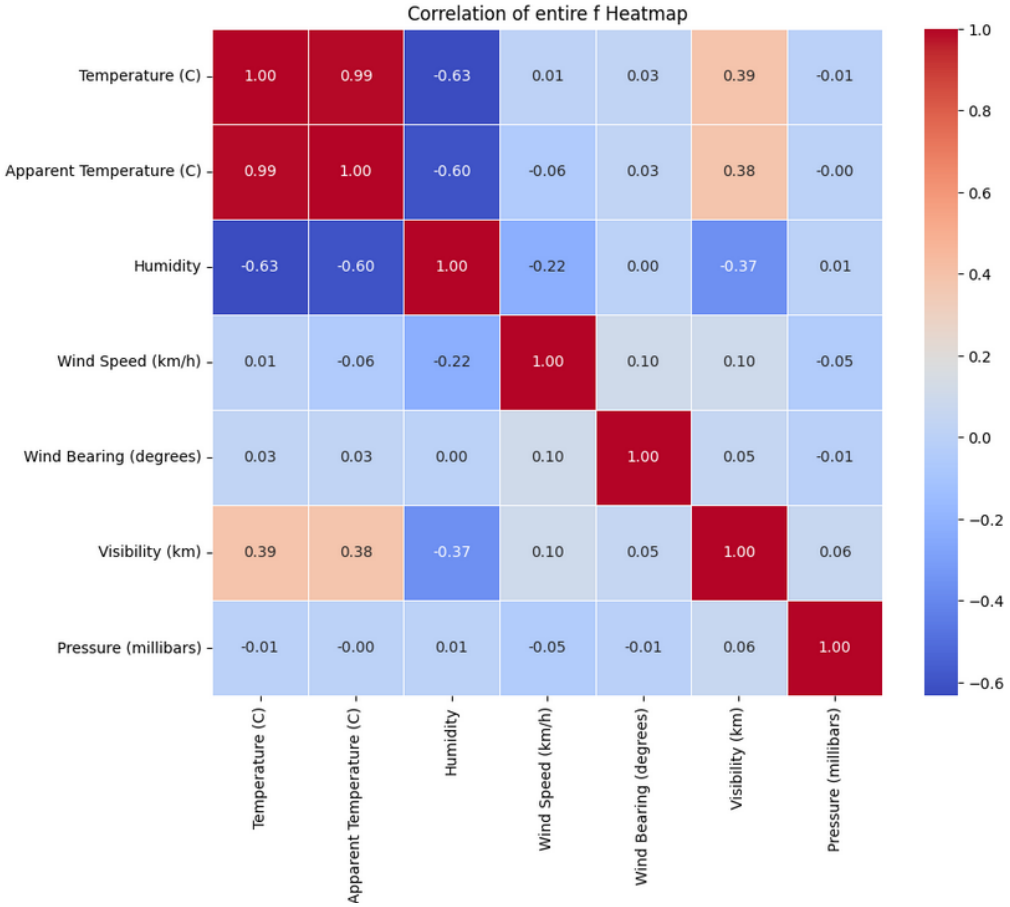
همچنین فرمت داده ها را بدست می آوریم

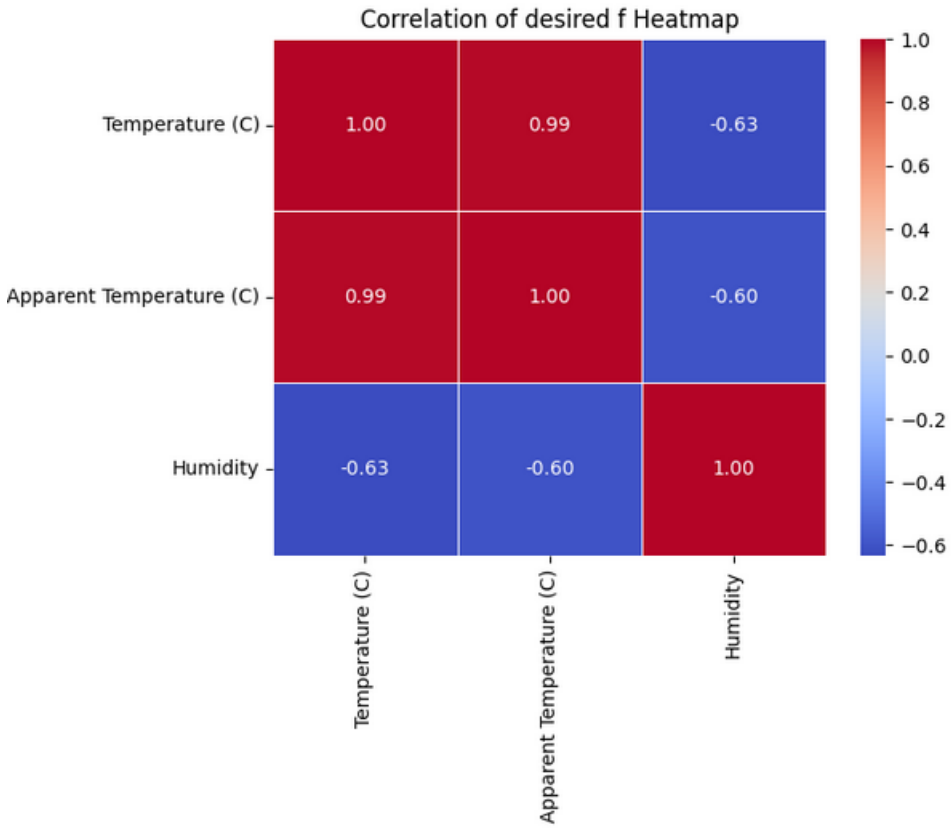
```
Type of  Formatted Date is <class 'str'>
Type of  Summary is <class 'str'>
Type of  Precip Type is <class 'str'>
Type of  Temperature (C) is <class 'numpy.float64'>
Type of  Apparent Temperature (C) is <class 'numpy.float64'>
Type of  Humidity is <class 'numpy.float64'>
Type of  Wind Speed (km/h) is <class 'numpy.float64'>
Type of  Wind Bearing (degrees) is <class 'numpy.float64'>
Type of  Visibility (km) is <class 'numpy.float64'>
Type of  Loud Cover is <class 'numpy.float64'>
Type of  Pressure (millibars) is <class 'numpy.float64'>
Type of  Daily Summary is <class 'str'>
```

ستون ها str و داده های null را حذف می کنیم .

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover
count	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.0
mean	11.932678	10.855029	0.734899	11.810640	187.509232	10.347325	0.0
std	9.551546	10.696847	0.195473	6.913571	107.383428	4.192123	0.0
min	-21.822222	-27.716667	0.000000	0.000000	0.000000	0.000000	0.0
25%	4.688889	2.311111	0.600000	5.828200	116.000000	8.339800	0.0
50%	12.000000	12.000000	0.780000	9.965900	180.000000	10.046400	0.0
75%	18.838889	18.838889	0.890000	14.135800	290.000000	14.812000	0.0
max	39.905556	39.344444	1.000000	63.852600	359.000000	16.100000	0.0

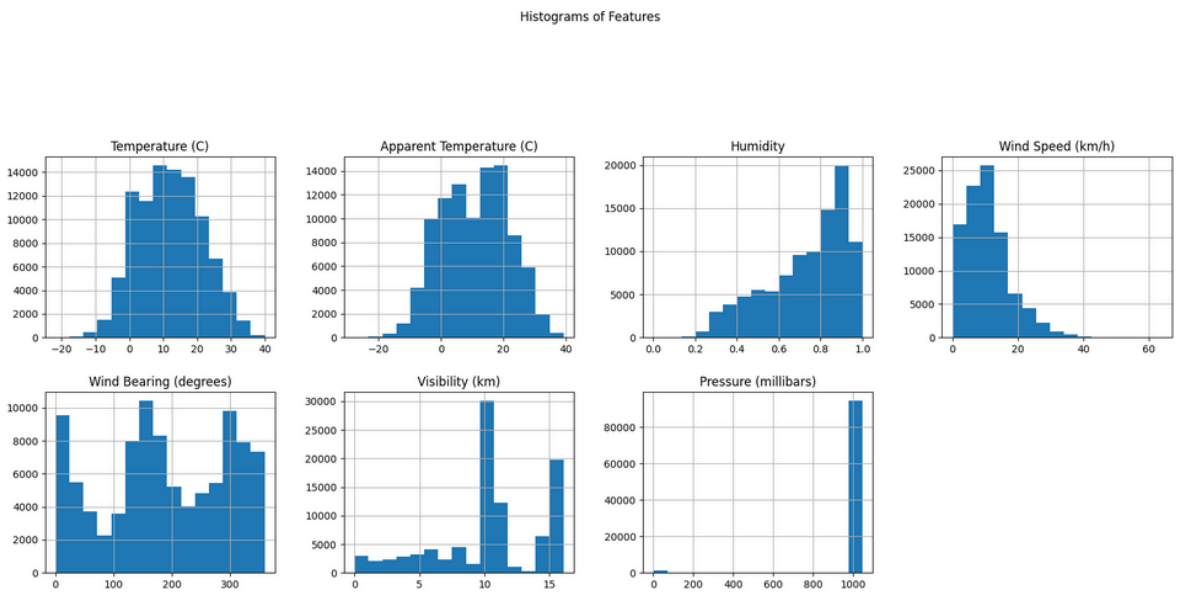
	Pressure (millibars)
count	96453.000000
mean	1003.235956
std	116.969906
min	0.000000
25%	1011.900000
50%	1016.450000
75%	1021.090000
max	1046.380000





در شکل بالا هر چقدر با توجه به بار رنگ در سمت راست و عدد در وسط میزان همبستگی نمایش داده شده است . درایه های قطر اصلی 1 هستند چون میزان همبستگی با خودش یکسان است.

همانطور که از نمودار پیداست بین feature 3 انتخابی همبستگی کمی وجود دارد . لذا تخمین یکی با دیگری امکان پذیر نیست . به همین دلیل باید از featureهای دیگر نیز استفاده کن .



هیستوگرام نوعی نمودار است که توزیع داده‌های عددی را با نشان دادن فراوانی نقاط داده در بازه‌های معین نمایش می‌دهد. این ابزار اساسی در تحلیل آماری برای مشاهده توزیع فراوانی پایه یک مجموعه داده پیوسته است که امکان ارزیابی سریع و آسان شکل، تغییرپذیری و گرایش مرکزی داده‌ها را فراهم می‌کند.

برای انتخاب بر اساس histogram باید داده هایی با پهنای بیشتر توزیع بهتر هستند.

این تصویر شامل هشت نمودار هیستوگرام است که هرکدام نمایانگر توزیع داده‌ها برای یک ویژگی مختلف متعلق به داده‌های آب و هوایی هستند. بیایید هر یک را تحلیل کنیم:

1. **دما (سانتی‌گراد)** :داده‌های دما تقریباً دارای توزیع نرمال به نظر می‌رسند، با بیشترین فراوانی در حدود 10 تا 20 درجه سانتی‌گراد. دماهای منفی و بالای 30 درجه کمتر دیده می‌شوند.
2. **دمای ظاهری (سانتی‌گراد)** :دمای ظاهری نیز توزیع مشابهی با دمای واقعی دارد، اما کمی پهن‌تر به نظر می‌رسد که نشان می‌دهد احساس دما توسط انسان‌ها ممکن است در طیف وسیع‌تری نوسان داشته باشد.
3. **رطوبت** :بیشترین فراوانی رطوبت در ارزش‌های بالا نزدیک به 1 (یا 100٪) متمرکز است، که نشان می‌دهد داده‌ها بیشتر مربوط به مناطق یا زمان‌های با رطوبت بالا هستند.
4. **سرعت باد (مایل بر ساعت)** :توزیع سرعت باد به سمت سرعت‌های پایین تمایل دارد، با بیشترین فراوانی در سرعت‌های بسیار پایین. سرعت‌های بالای باد بسیار کمتر رخ می‌دهند.
5. **جهت باد (درجه)** :این نمودار نشان می‌دهد که جهت باد در داده‌ها یک توزیع نسبتاً یکنواخت دارد، با اندکی افزایش در فراوانی در اطراف 200 تا 300 درجه.
6. **دید (کیلومتر)** :توزیع دید نشان‌دهنده تمرکز بسیار زیاد بر روی دید بسیار خوب (بالاترین فراوانی در ارزش‌های بالا) است و افت شدید در فراوانی با کاهش دید.
7. **فشار (میلی‌بار)** :توزیع فشار بیشتر در اطراف 1000 میلی‌بار متمرکز است، که یک مقدار نسبتاً معمولی برای فشار اتمسفری است. فشارهای بسیار بالا یا پایین تر نادر هستند.

بر اساس نمودار های دیده شده visibility بهترین انتخاب ما است.

2 .

حال داده ها را استاندارد سازی کرده و آنها را به داده های تست وترین تقسیم می کنیم:

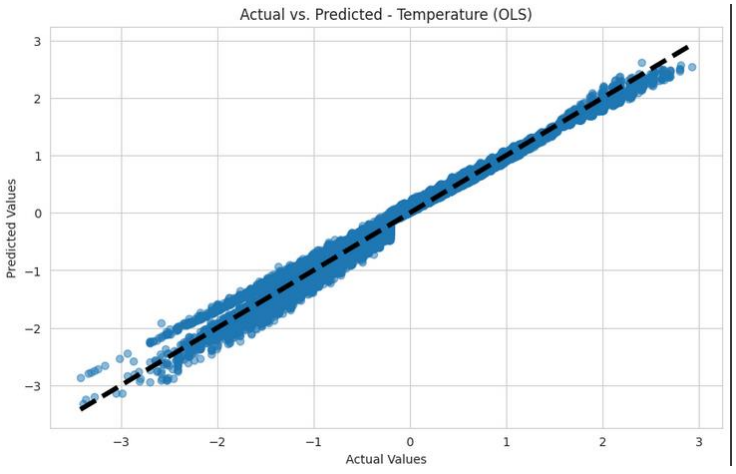
ویژگی های ما شامل Visibility و Humidity وApparent Temperature و Temperature هستند که به خواستهی سوال یکبار Apparent Temperatue و یکبار Temperature خروجی ما هستند حال LS و RLS زا اعمال می کنیم و خطای MSE آن ها را بدست می آوریم.

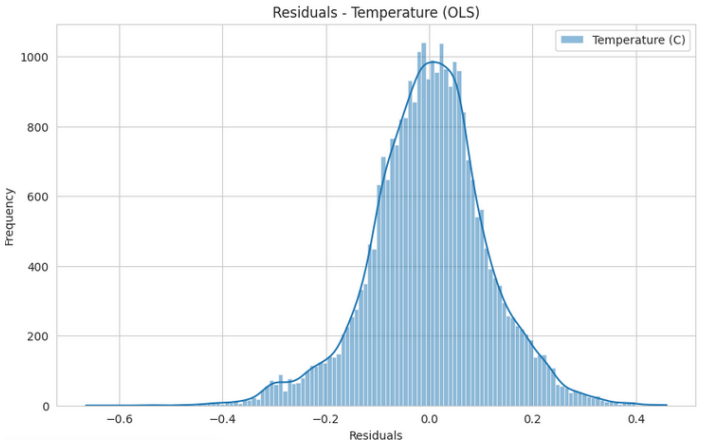
MSE:

```
Temperature Prediction MSE - Ordinary Least Squares: 0.012878511126006584
Temperature Prediction MSE - Ridge Regression: 0.012878516411735964
Apparent Temperature Prediction MSE - Ordinary Least Squares: 0.013667662366379614
Apparent Temperature Prediction MSE - Ridge Regression: 0.013667655057284438
```

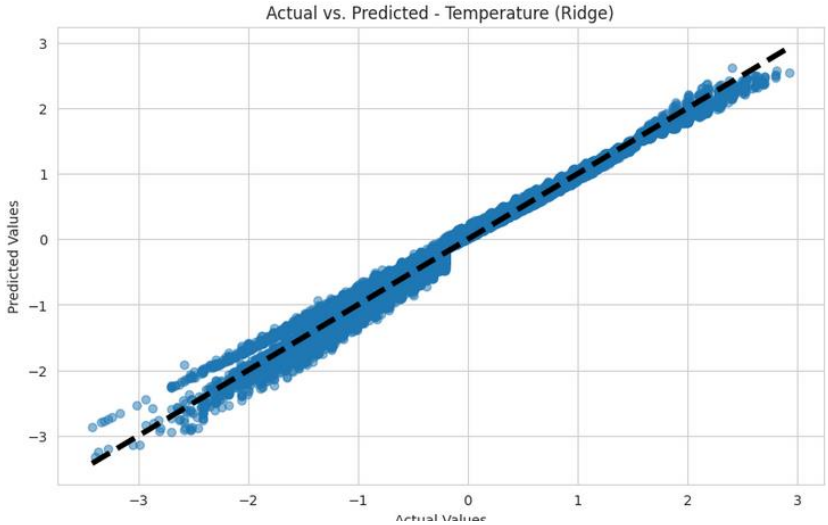
1. **نمودار نقاط داده‌های واقعی در مقابل پیش‌بینی‌ها** :این نمودار ها نشان می‌دهند که پیش‌بینی‌ها از مدل‌ها چگونه با داده‌های واقعی مقایسه می‌شوند. اگر نقاط بر روی خط قطری قرار گیرند، نشان‌دهنده دقت خوب پیش‌بینی‌های مدل است.
2. **نمودارهای باقی‌مانده‌ها** :این هیستوگرام‌ها توزیع باقی‌مانده‌ها (خطاها) را نشان می‌دهند. به طور ایده‌آل، باید توزیعی متقارن و به شکل زنگی با مرکزیت دور صفر داشته باشیم، که نشان‌دهنده توزیع نرمال خطاهای مدل است.

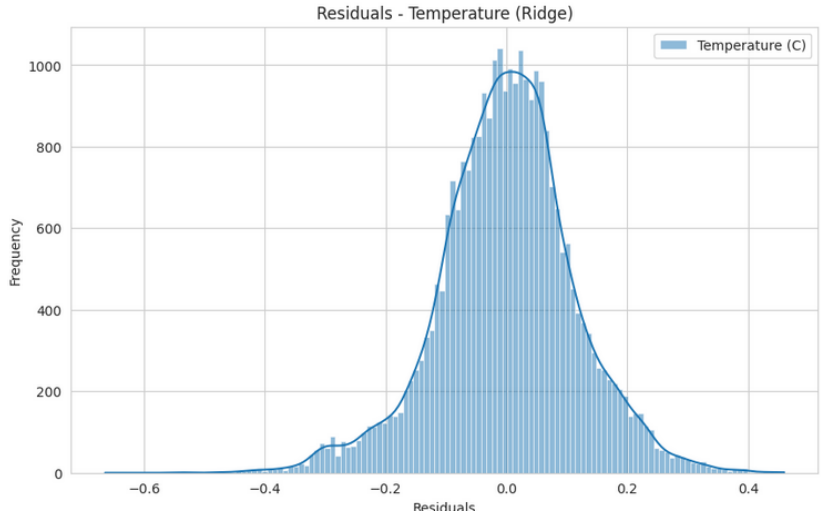
Ls برای خروجی دما



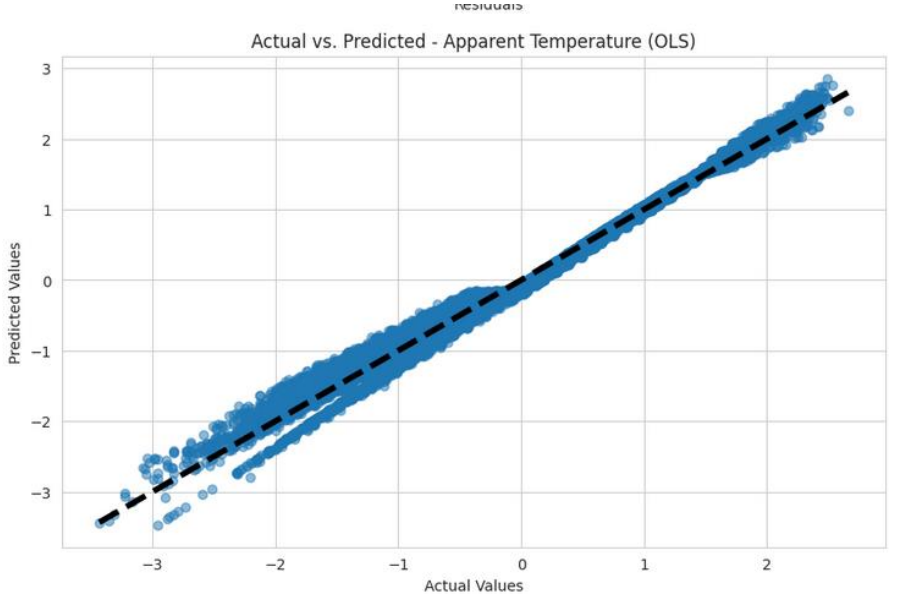


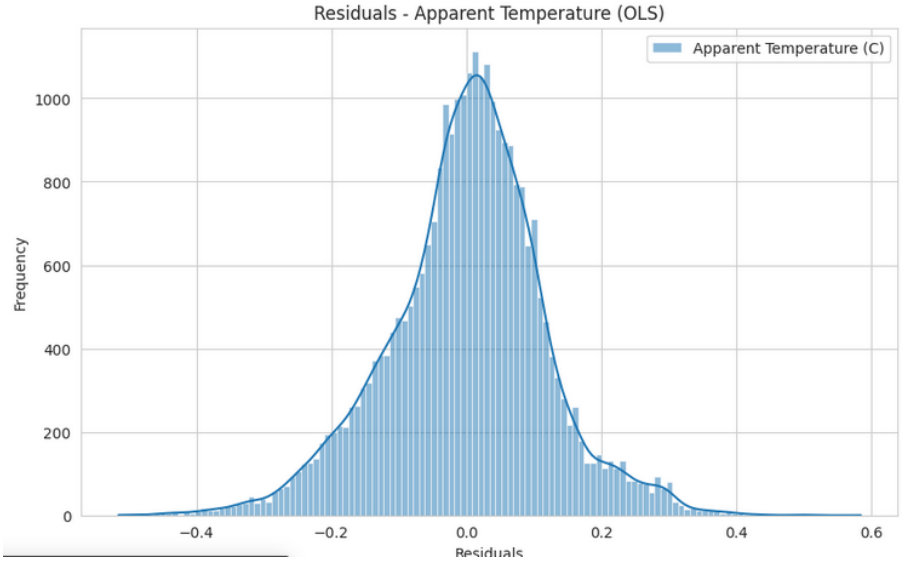
RLS برای خروجی دما



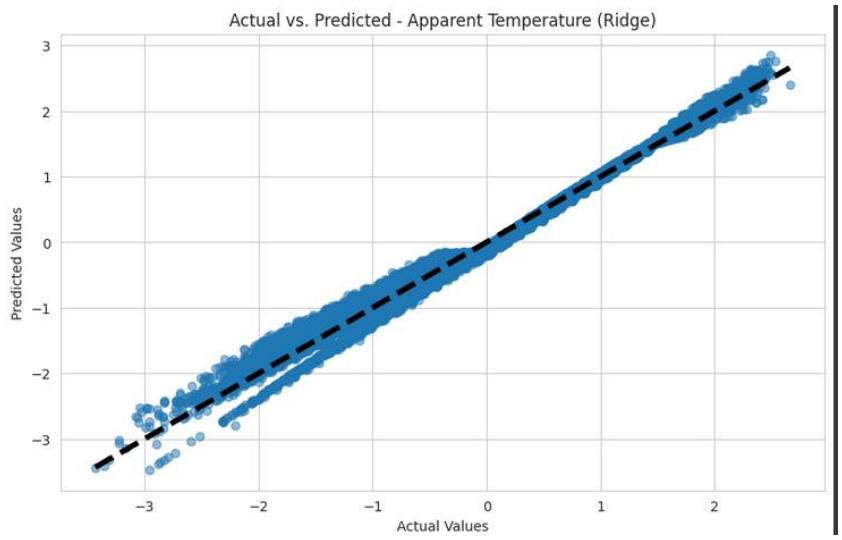


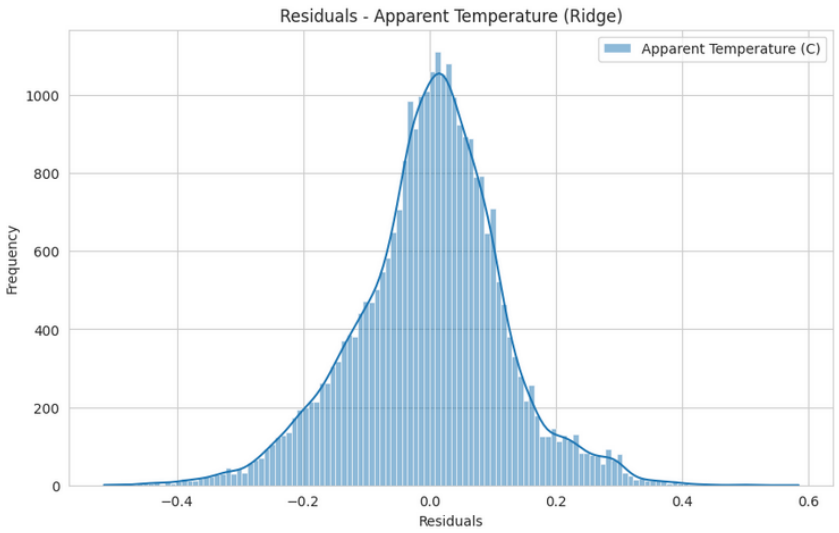
LS برای خروجی دمای آشکار

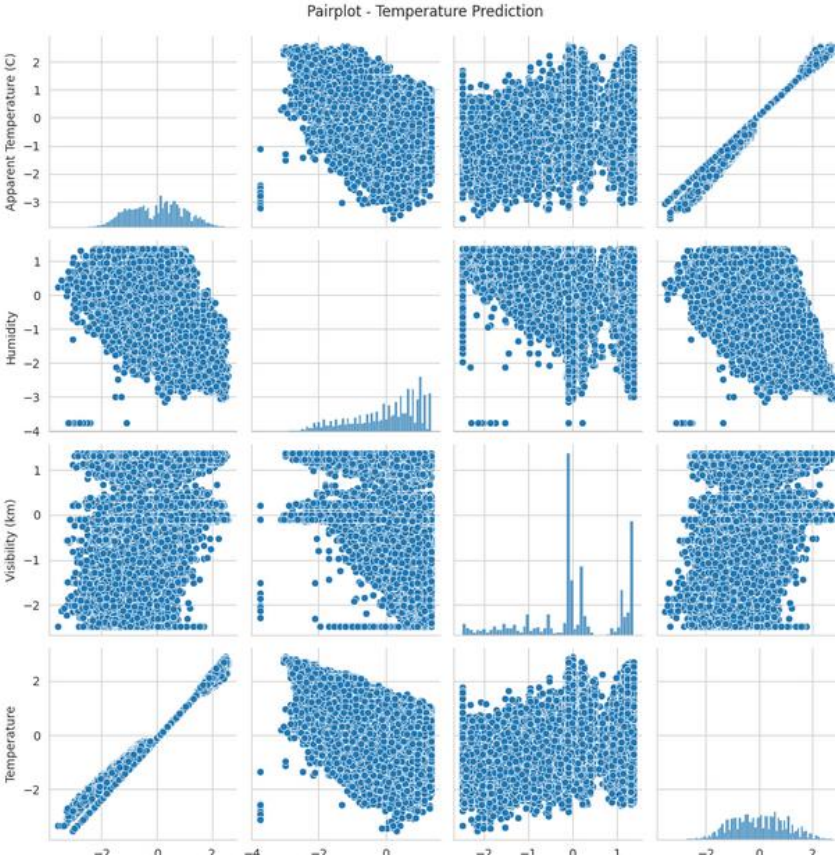


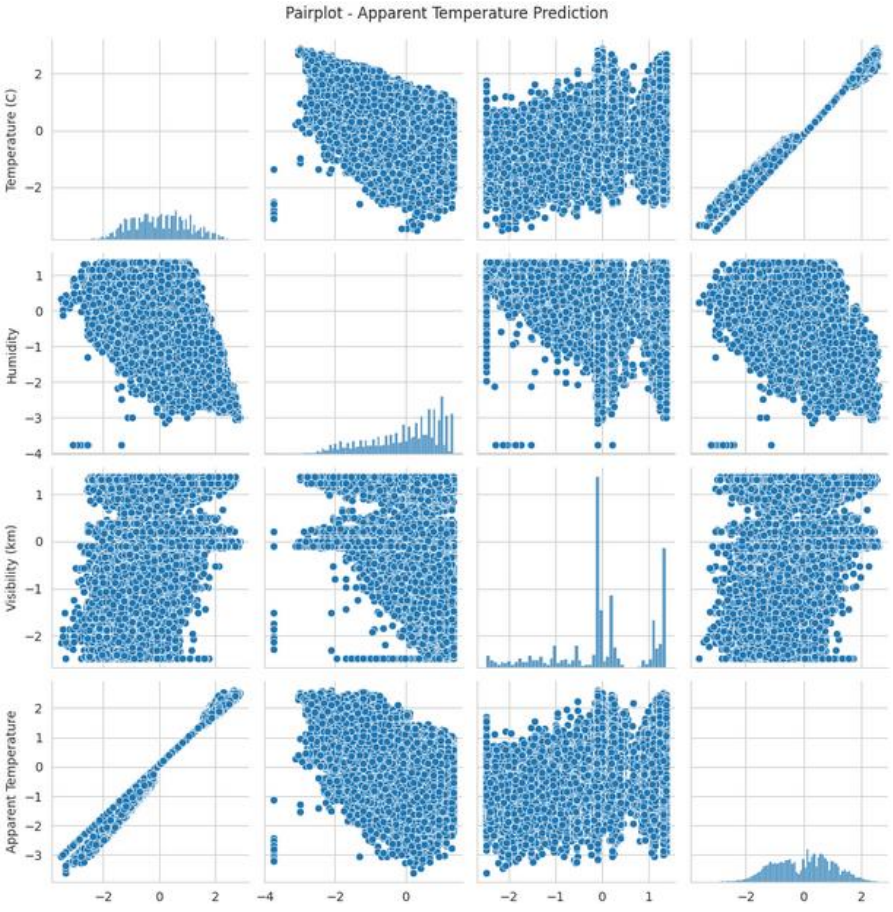


RLS برای دمای آشکار






Pairplot - Temperature Prediction
Apparent Temperature (C)
Humidity
Visibility (km)
Temperature


Pairplot - Apparent Temperature Prediction
Temperature (C)
Humidity
Visibility (km)
Apparent Temperature

3 .

:WLS

در روش کمترین مربعات وزن‌دار (Weighted Least Squares یا WLS)، به هر نقطه داده وزن خاصی اختصاص داده می‌شود که براساس اهمیت و قابلیت اطمینان آن نقطه تعیین می‌شود. این کار برای اهداف مختلفی ممکن است اعمال شود، از جمله اعمال وزن‌ها بر اساس واریانس خطاهای مرتبط با هر مشاهده. در روش WLS، معمولاً وزن‌ها بر اساس واریانس خطاها انتخاب می‌شوند. مشاهده‌هایی با واریانس بالاتر (یعنی عدم قطعیت بیشتر) وزن کمتری دارند، در حالی که مشاهده‌هایی با واریانس کمتر وزن بیشتری دارند. این امر به مدل این امکان را می‌دهد که بیشترین توجه را به مشاهداتی که معتبرتر یا کمتر نویز دارند، بدهد.

```
weights_T = 1 / (1 + 0.1*df_drop['Humidity'])
weights_AT = 1 / (1 + 0.1*df_drop['Visibility (km)'])
```

```
Temperature Prediction MSE - Weighted Least Squares: 0.012876949037476607
Apparent Temperature Prediction MSE - Weighted Least Squares: 0.013684055553901182
```

می بینیم کهMSE مطلوب درآمده است