بسمه تعالی

مهدی وحیدمقدم

۴۰۲۱۳۰۷۴

Mini_project1

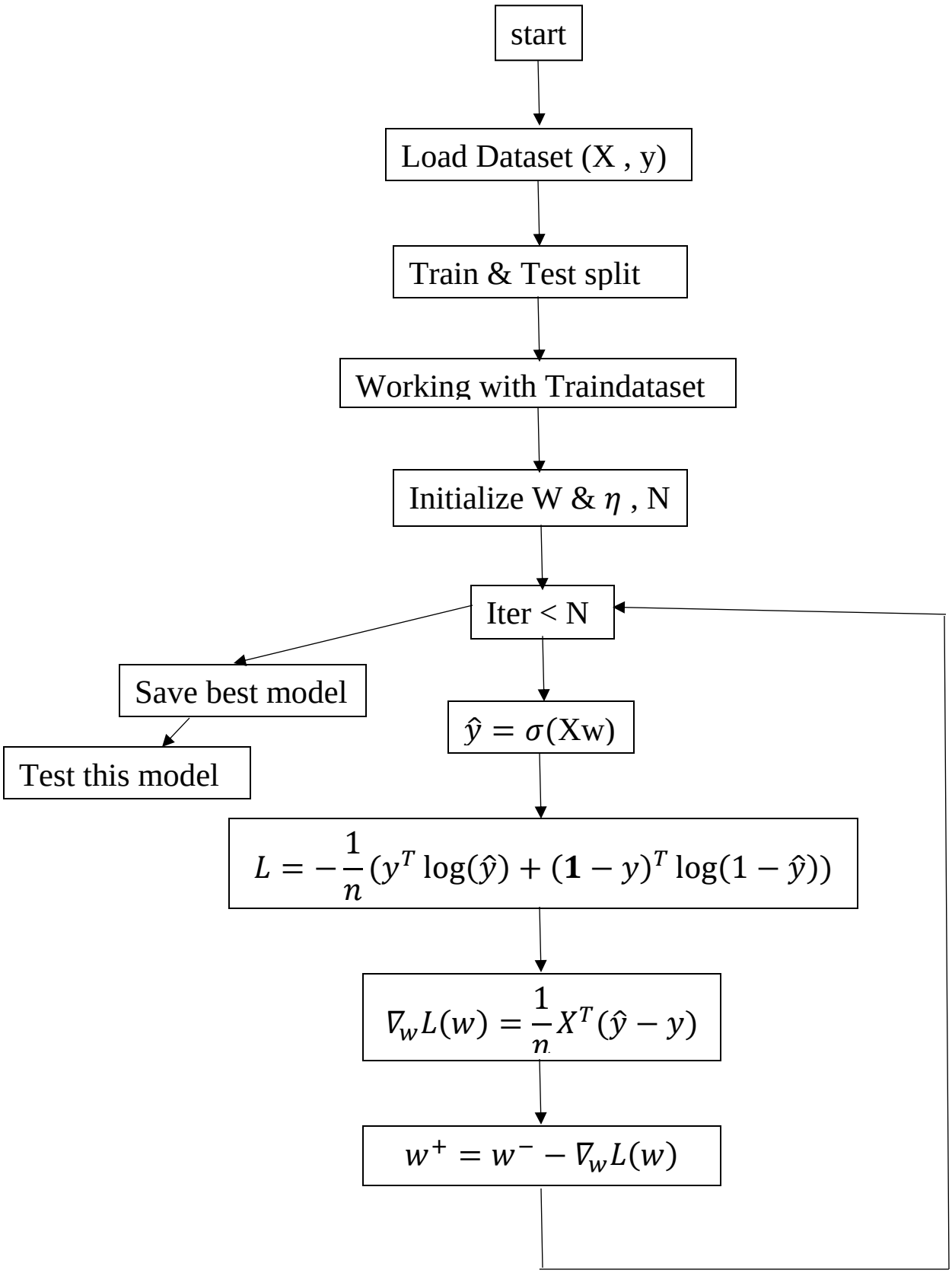
درس یادگیری ماشین

Google colab link

https://colab.research.google.com/drive/1j-AOiEI74iSmHpr5vg7YAJSR542OT371?usp=sharing

سوال ۱)

۱. فرآیند آموزش و ارزیابی یک مدل طبقه‌بند خطی را به‌صورت دیاگرامی بلوکی نمایش دهید و در مورد اجزای مختلف این دیاگرام بلوکی توضیحاتی بنویسید. تغییر نوع طبقه‌بندی از حالت دوکلاسه به چندکلاسه در کدام قسمت از این دیاگرام بلوکی تغییراتی ایجاد می‌کند؟ توضیح دهید.


start
Load Dataset (X , y)
Train & Test split
Working with Traindataset
Initialize W & η , N
Iter < N
Save best model
Test this model
$\hat{y} = \sigma(\mathrm{Xw})$
$L = -\frac{1}{n}(y^T \log(\hat{y}) + (\mathbf{1} - y)^T \log(1 - \hat{y}))$
$\nabla_w L(w) = \frac{1}{n} X^T (\hat{y} - y)$
$w^+ = w^- - \nabla_w L(w)$

توضیحات:

ابتدا یک دیتاسیت(می تواند ساختگی و یا واقعی باشد)در نظر می‌گیریم.سپس تعدادی از این دیتا را (مثلا ۸۰ درصد) به عنوان داده های train و مابقی را به عنوان داده تست در نظر می‌گیریم.بعد از آن آموزش را با استفاده از داده‌های train انجام می‌دهیم.آموزش شامل مراحل زیر است:

ابتدا وزن یا وزن های موردنظر را به صورت رندوم تعیین می‌کنیم.سپس تعداد دفعات تکرار آموزش را تعیین می‌کنیم و گام آموزشی را هم تعیین می‌کنیم.سپس در هر مرحله مقدار خروجی ماشین طراحی شده را به دست می‌آوریم،سپس با استفاده از تابع Loss مقدار اختلاف آن را با مقدار صحیح آن به دست می‌آوریم،سپس مقدار این تابع Loss را با استفاده از یک تابع بهینه‌ساز حداقل می‌کنیم و وزن یا وزن های جدید را بر اساس این مقدار بهینه شده به دست می‌آوریم.پس از N بار که این کار را تکرار کردیم،مدل نهایی را به عنوان بهترین مدل ذخیره کرده و آن را با استفاده از داده های تست موردآزمایش قرار می‌دهیم و مقدار دقت مدل پیش بینی شده را تعیین می‌کنیم.

اگر کلاس‌های موجود از ۲تا بیشتر شود،در واقع y یا target ما که مساوی تعداد کلاس ها می-باشد،دیگر برابر صفر و یک نیست و باید تعداد مقادیر بیشتری داشته باشد.همچنین در تفکیک کلاس های مختلف اگر در حالت اول با یک خط دو کلاس تفکیک می‌شدند،اینجا با توجه به افزایش کلاس ها نیاز به چند خط برای تفکیک کلاس ها داریم.نکته دیگر این است که y بعد از وارد شدن به تابع برای مثال سیگموید،احتمال تعلق داده به هر یک از کلاس‌ها را به ما می‌دهد که ما باید درایه ای که مقدار ماکسیمم این احتمالات را دارد،به عنوان کلاس آن داده تعیین کنیم.مثلا اگر y به صورت زیر داشتیم که نشان دهنده وجود ۴ کلاس است،

[0.15 0.45 0.17 0.23]

باید $y = 1$ برای داده موردنظر انتخاب شود.زیرا داده دوم بیشترین مقدار را دارا است.

۲. با استفاده از sklearn.datasets، یک دیتاست با ۱۰۰۰ نمونه، ۴ کلاس و ۳ ویژگی تولید کنید و آن را به‌صورتی مناسب نمایش دهید. آیا دیتاستی که تولید کردید چالش برانگیز است؟ چرا؟ به چه طریقی می‌توانید دیتاست تولیدشده خود را چالش‌برانگیزتر و سخت‌تر کنید؟

این دیتاست به وسیله‌ی کد زیر ایجاد می‌شود:

```
X , y = make_classification(n_samples = 1000 , n_features = 3 ,n_classes = 4
, n_redundant = 1 , n_clusters_per_class = 1 , class_sep = 3 , random_state =
74)
```

این کد یک دیتاست با ۱۰۰۰ نمونه و ۳ ویژگی و ۴ کلاس ایجاد می‌کند.۳پارامتر بعدی درباره نحوه‌ی قرارگیری این دیتاها و مقدار نزدیک بودن آن ها به یکدیگر و ... می‌باشد.همچنین random state باعث می‌شود همیشه همین دیتا تولید شود.(با ران کردن کد در دفعات بعد)

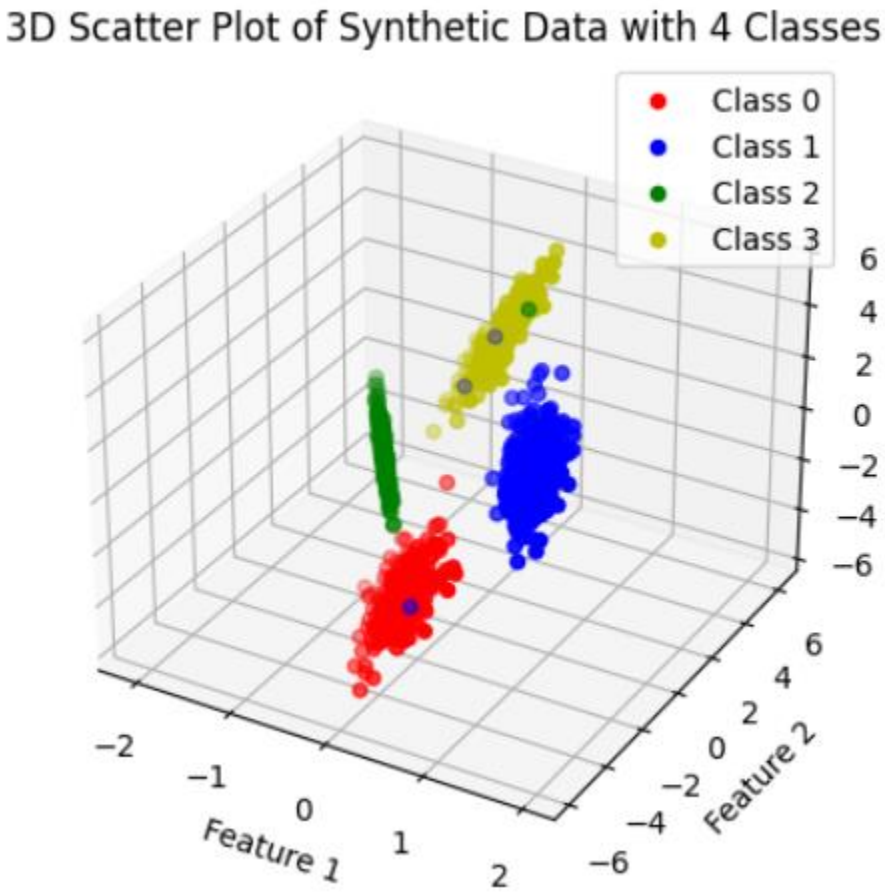
این دیتاست خیلی ساده نیست اما چالش آن هم خیلی زیاد نیست.(البته دیتای سه بعدی قطعا چالش بیشتری نسبت به دیتای دو بعدی دارد اما این جا منظور از چالش سختی کلاس بندی دیتا می‌باشد.)

برای این که چالش این دیتا بیشتر شود،می‌توان پارامترهای ۳ و ۴ و ۵ و دیگر پارامترهایی که اینجا تنظیم نشده و به صورت دیفالت است،به نوعی تنظیم شوند تا دیتاها طوری قرار گیرند تا کلاس بندی آن ها سخت تر شود.

نمایش داده:

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
colors = ['r', 'b', 'g', 'y']
for class_index in range(4):
    ax.scatter(X[y == class_index, 0], X[y == class_index, 1], X[y ==
class_index, 2], c=colors[class_index], label=f'Class {class_index}')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.set_title('3D Scatter Plot of Synthetic Data with 4 Classes')
ax.legend()
plt.show()
```

همان طور که مشاهده می‌شود در یک شکل سه بعدی دیتاهای موجود را رسم کرده ایم.هر یک از این ۴ کلاس با یک رنگ رسم شده اند.در واقع چون در این جا سه ویژگی داشتیم،رسم داده ها به صورت سه بعدی است.شکل رسم شده‌ی دیتاها به صورت زیر است:



---

۳. با استفاده از حداقل دو طبقه‌بند خطی آمادهٔ پایتون (در sklearn.linear_model) و در نظر گرفتن فراپارامترهای مناسب، چهار کلاس موجود در دیتاست قسمت قبلی را از هم تفکیک کنید. ضمن توضیح روند انتخاب فراپارامترها (مانند تعداد دورهٔ آموزش و نرخ یادگیری)، نتیجهٔ دقت آموزش و ارزیابی را نمایش دهید. برای بهبود نتیجه از چه تکنیک‌هایی استفاده کردید؟

اولین مدل تعریف شده به صورت زیر است:

```
model_0_1 = SGDClassifier( max_iter = 200 , alpha = 0.1 , random_state = 74)
```

اولین مدل ما SGDClassifier است که از مدل‌های خطی است.در این مدل تعداد ۲۰۰ دوره آموزش و نرخ یادگیری ۰.۱ در نظر گرفته شده است.نحوه انتخاب این پارامتر ها به این صورت است که ابتدا مقادیر معمولی تر(تعداد دوره آموزش کمتر و نرخ یادگیری بزرگتر)برای مدل در نظر میگیریم و عملکرد مدل را مشاهده می‌کنیم و اگر عملکرد مدل خوب نبود،کمی این مقادیر را تغییر می‌دهیم.چون مثلا هر چقدر تعداد دوره آموزش کمتر باشد،مدل سریعتر خواهد بود.همچنین هر چه نرخ یادگیری بزرگتر باشد،گام های ما برای رسیدن به بهترین جواب بزرگتر است و سریعتر به آن جواب خواهیم رسید.اگر دیدیم این مدل جواب نداد،گام ها را کوچکتر و تعداد دوره آموزش را بیشتر می‌کنیم تا ببینیم مدل عملکرد بهتری دارد یا خیر.گاهی اوقات با تغییر این پارامترها بهبود چندانی حاصل نمی‌شود و باید classifier را تغییر دهیم تا به نتیجه بهتری برسیم.

در ادامه مدل را اعمال و نتایج را مشاهده می‌کنیم:

```
x_train , x_test , y_train , y_test = train_test_split(X ,
                                                       y ,
                                                       test_size = 0.2)
# fitting the model_0
model_0_1.fit(x_train , y_train)

# see the first 35 parameters predicted
y_pred1 = model_0_1.predict(x_test)
print(f"(model_0_1) our first 35 y_test is {y_test[:35]}\n            our
first 35 y_pred is {y_pred1[:35]}")

# accuracy for this model(train data)
print(f"our first model score is {model_0_1.score(x_train , y_train)}")
```

ابتدا داده‌ی خود را به دو بخش train و test با نسبت ۸۰ به ۲۰ تقسیم می‌کنیم.سپس مدل را با داده‌های train،fit می‌کنیم.سپس با استفاده از predict داده های تست خود را پیش بینی می‌کنیم.سپس ۳۵تا داده اول پیش بینی شده را با ۳۵ برچسب اصلی تست نمایش داده و مقایسه می‌کنیم.سپس با استفاده از دستور score میزان داده های درست پیش بینی شده را می‌بینیم.نتیجه به صورت زیر است:

```
(model_0_1) our first 35 y_test is [3 3 0 1 2 3 0 3 0 2 0 0 0 0 0 2 2 3 1 2 1
2 1 1 2 1 1 1 3 1 2 0 2 3 2]
```

```
            our first 35 y_pred is [3 3 0 1 2 3 0 3 0 2 0 0 0 0 0 2 2 3 1 2 1
2 1 1 2 1 1 1 3 3 2 0 2 3 2]
our first model score is 0.99125
```

همان طور که مشاهده می‌شود در ۳۵ پیش بینی اول،فقط یک پیش بینی غلط وجود دارد.همچنین در کل حدود ۹۹ درصد از داده ها درست پیش بینی شده است که این یک عملکرد خیلی خوب برای مدل است.

اگر مدل همان مدل قبلی و با تغییرات زیر باشد:

```
model_0_2 = SGDClassifier( max_iter = 1000 , alpha = 0.01 , random_state =
74)
```

همان طور که مشاهده می‌شود تعداد دوره آموزش را به ۱۰۰۰ افزایش و نرخ یادگیری را برابر ۰.۰۱ قرار دادیم.

نتایج در این جا به صورت زیر است:

```
(model_0_2) our changed first 35 y_test is [3 3 0 1 2 3 0 3 0 2 0 0 0 0 0 2 2
3 1 2 1 2 1 1 2 1 1 1 3 1 2 0 2 3 2]
            our changed first 35 y_pred is [3 3 0 1 2 3 0 3 0 2 0 0 0 0 0 2 2
3 1 2 1 2 1 1 2 1 1 1 3 3 2 0 2 3 2]
our changed first model score is 0.99
```

همان طور که مشاهده می‌شود عملکرد مدل تقریبا تفاوتی نکرده است.حتی مقدار کمی بدتر شده است که نشان می‌دهد همیشه افزایش دوره های آموزش و کوچک کردن نرخ یادگیری به بهبود مدل کمک نمی‌کند.

حال با یک classifier دیگر همین کار را انجام می‌دهیم.این classifier به صورت زیر است:

```
model_1_1 = RidgeClassifier( max_iter = 200 , alpha = 0.1 , random_state =
74)
```

در این جا از مدل خطی RidgeClassifier استفاده کرده ایم.نتایج برای این مدل به صورت زیر است:

```
(model_1_1) our first 35 y_test is [0 0 1 1 0 1 2 2 2 2 0 3 2 0 0 0 2 2 2 2 1
0 2 3 2 2 1 3 1 0 3 2 1 0 2]
            our first 35 y_pred is [0 0 1 1 0 3 2 2 2 2 0 3 2 0 0 0 2 2 2 2 1
0 2 3 2 2 1 3 1 0 3 2 3 0 2]
our second model score is 0.985
```

همان طور که مشاهده می‌شود accuracy در این جا حدود ۹۸ درصد است که کمی کمتر از مدل قبلی است.

در کل با توجه به این که دیتاست ما خیلی پیچیده نیست،فرق بین این مدل‌ها خیلی معلوم نمی-شود و تقریبا همه ی مدل ها نتیجه خوبی در پی خواهند داشت.

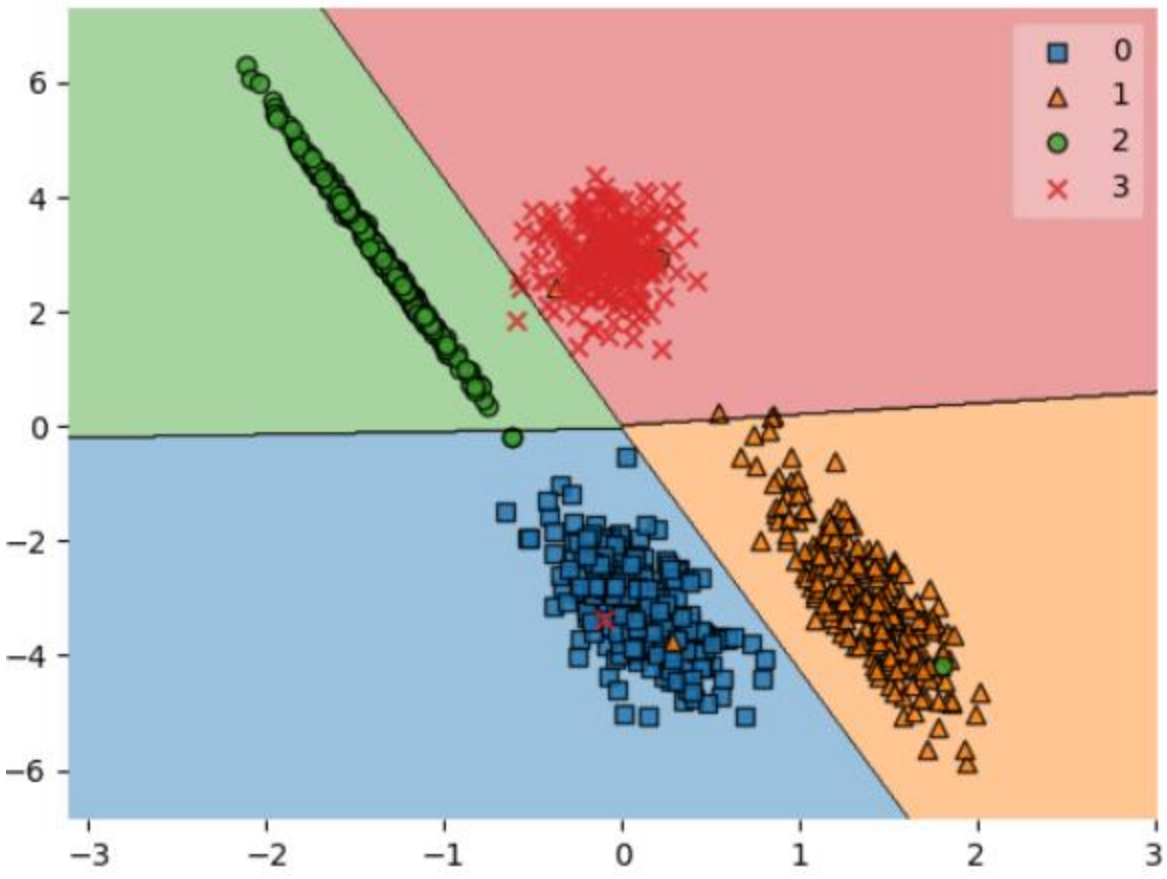
---

۴. مرز و نواحی تصمیم‌گیری برآمده از مدل آموزش‌دیدهٔ خود را به همراه نمونه‌ها در یک نمودار نشان دهید. اگر می‌توانید نمونه‌هایی که اشتباه طبقه‌بندی شده‌اند را با شکل و رنگ متفاوت نمایش دهید.

برای کشیدن مرز تصمیم گیری ابتدا داده ها را بر روی صفحه x وy آورده و سپس برای آن ها این مرز را رسم می‌کنیم.کد به صورت زیر است:

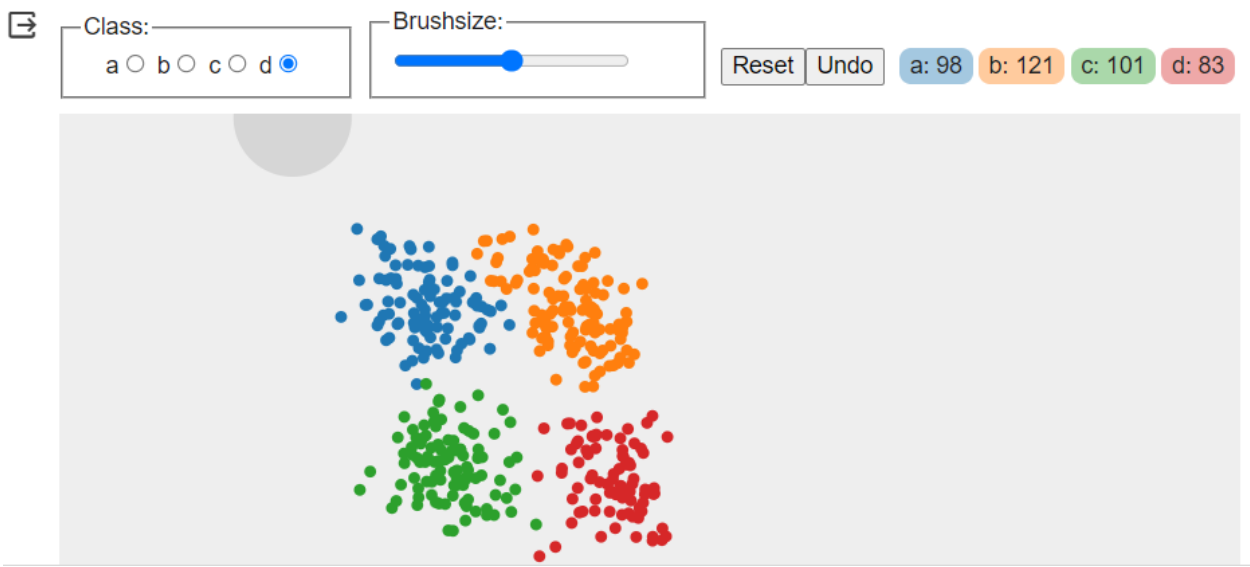
```
X_for_db = X[: , 0:2]
model_1_2.fit(X_for_db , y)
plot_decision_regions(X_for_db , y , clf = model_1_2)
```

خروجی به صورت زیر است:



۵. فرآیندی مشابه قسمت «۲» را با تعداد کلاس و ویژگی دلخواه؛ اما با استفاده از ابزار drawdata تکرار کنید. قسمت‌های «۳» و «۴» را برای این داده‌های جدید تکرار و نتایج را به‌صورتی مناسب نشان دهید.

دیتای رسم شده با این ابزار به صورت زیر است:



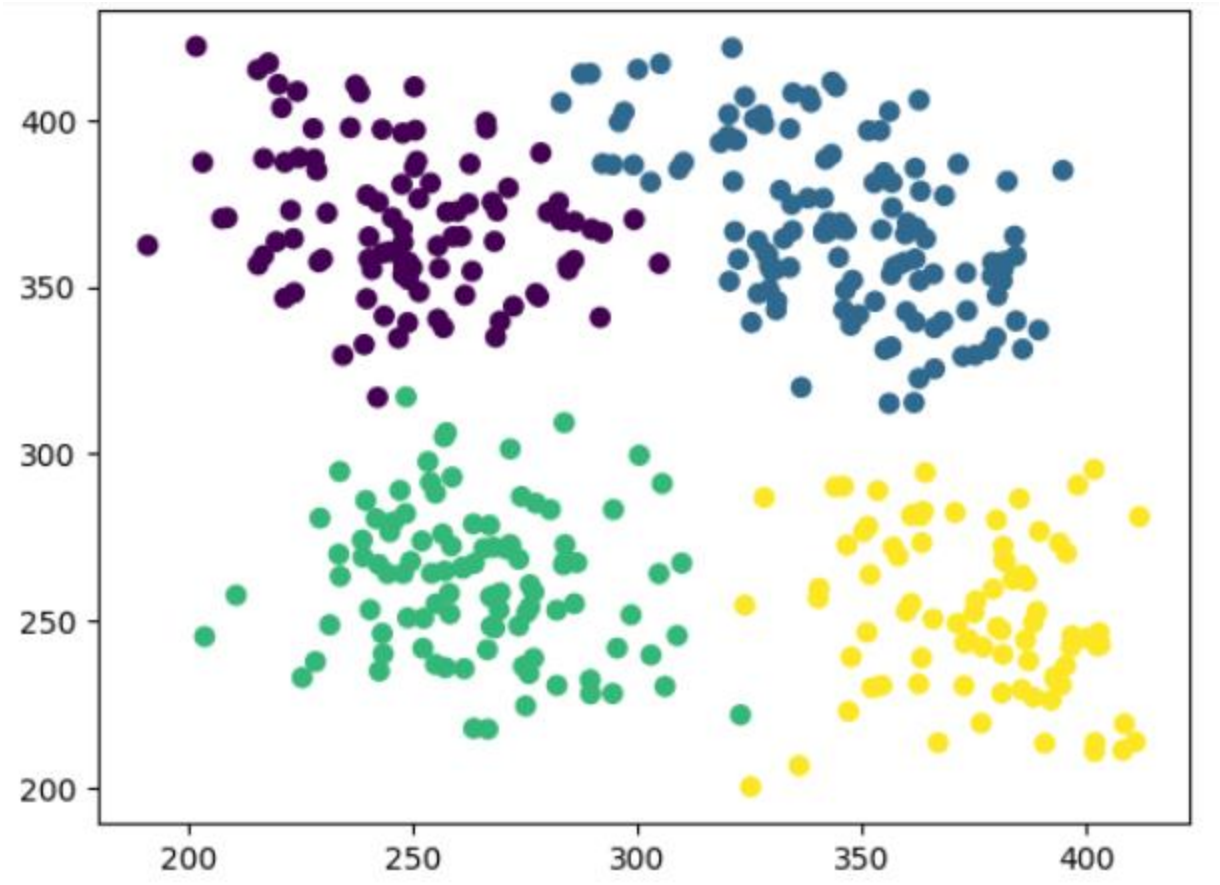
سپس با استفاده از کد زیر این دیتا را با استفاده از دو classifier مختلف کلاس بندی کرده و در آخر مرز تصمیم گیری را برای آن رسم می‌کنیم:

```
X_draw = widget.data_as_pandas[['x' , 'y']].values
color1 = widget.data_as_pandas[['label']].values.tolist()
color = widget.data_as_pandas[['label']].values.tolist()
for i in range(403):
  if color1[i] == ['a']:
    color1[i] = 1
  if color1[i] == ['b']:
    color1[i] = 2
  if color1[i] == ['c']:
   color1[i] = 3
  if color1[i] == ['d']:
    color1[i] = 4
color = np.array(color)
color1 = np.array(color1)
plt.scatter(X_draw[: , 0] ,
            X_draw[: , 1],
            c = color1)
x_draw_train , x_draw_test , y_draw_train , y_draw_test =
train_test_split(X_draw ,
                                                    color1 ,
                                                    test_size = 0.2)
model_0_1.fit(x_draw_train , y_draw_train)

# see the first 35 parameters predicted
y_draw_pred1 = model_0_1.predict(x_draw_test)
print(f"(model_0_1) our first 35 y_test is {y_draw_test[:35]}\n
our first 35 y_pred is {y_draw_pred1[:35]}")

# accuracy for this model(train data)
```

```
print(f"our first model score is {model_0_1.score(x_draw_test ,
y_draw_test)}")


model_1_1 = RidgeClassifier( max_iter = 200 , alpha = 0.1 , random_state =
74)


# fitting the model_1_1
model_1_1.fit(x_draw_train , y_draw_train)

# see the first 35 parameters predicted
y_draw_pred2 = model_1_1.predict(x_draw_test)
print(f"(model_1_1) our first 35 y_test is {y_draw_test[:35]}\n
our first 35 y_pred is {y_draw_pred2[:35]}")

# accuracy for this model(train data)
print(f"our second model score is {model_1_1.score(x_draw_test ,
y_draw_test)}")
```

در این کد ابتدا را به صورت یک دیتای عددی در می‌آوریم.دو ستون اول این دیتا که ویژگی های آن هستند را داخل X_draw و برچسب ها را داخل color1 ذخیره می‌کنیم.سپس هر جا این برچسب ها به ترتیب a و b و c و d هستند آن را به ۱ و ۲ و ۳ و ۴ تبدیل می‌کنیم.زیرا برچسب باید به صورت رنگ یا عدد باشد.

سپس با این داده های عددی دیتا را رسم کردیم که به صورت زیر است:



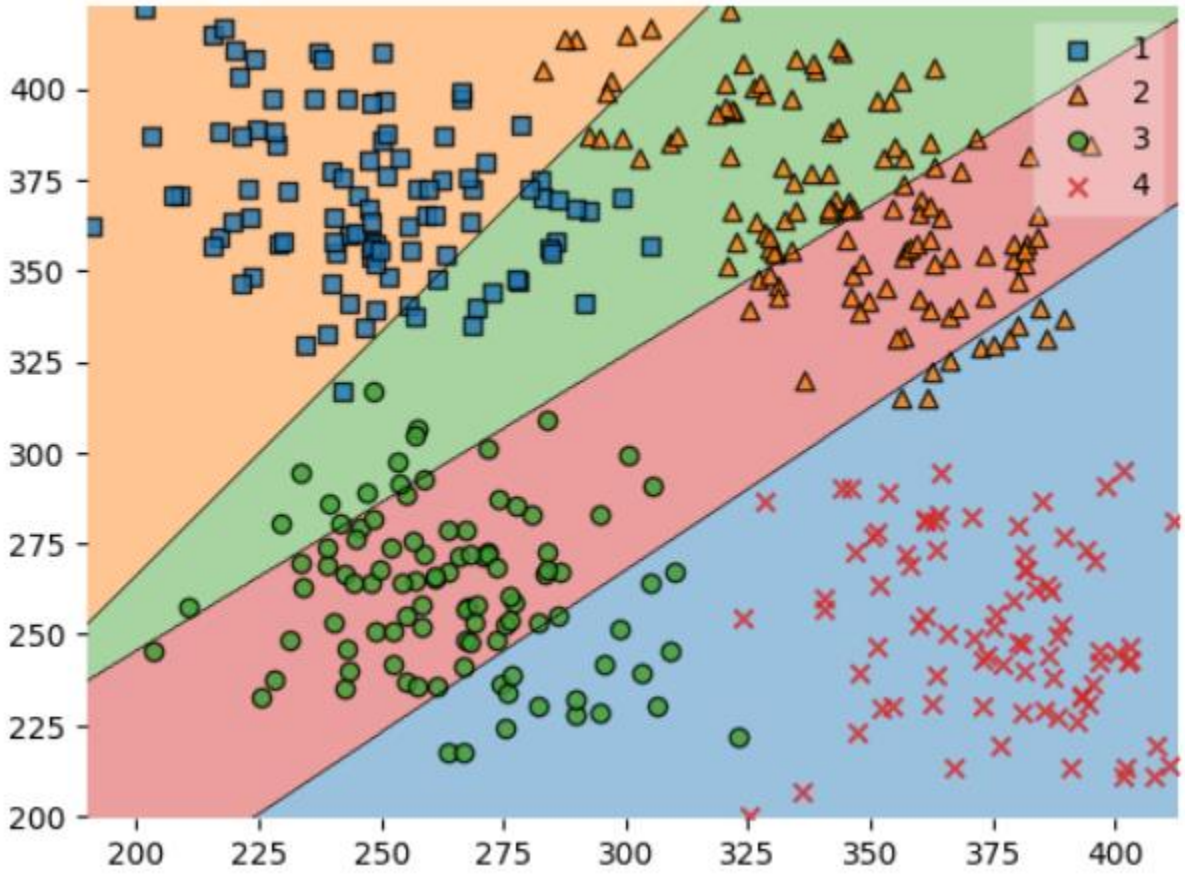
همان طور که مشاهده می‌شود داده ها به درستی رسم شده اند.

سپس این داده ها به دو بخش train و test با نسبت ۸۰ به ۲۰ تقسیم کرده و مدل را با مانند قسمت‌های قبل با داده‌های train آموزش می‌دهیم.نتایج به صورت زیر است:

```
(model_0_1) our first 35 y_test is [2 1 2 2 4 2 4 3 2 4 1 2 2 2 2 1 3 3 1 3 3
3 2 2 1 3 2 3 3 4 1 2 1 3 4]
            our first 35 y_pred is [1 1 2 2 4 2 4 2 2 4 1 2 2 2 2 1 2 2 1 2 2
4 2 2 1 2 2 2 4 4 1 2 1 4 4]
our first model score is 0.6419753086419753
(model_1_1) our first 35 y_test is [2 1 2 2 4 2 4 3 2 4 1 2 2 2 2 1 3 3 1 3 3
3 2 2 1 3 2 3 3 4 1 2 1 3 4]
            our first 35 y_pred is [2 2 2 2 4 2 4 3 2 4 1 2 2 2 2 1 3 3 1 3 3
3 2 2 1 3 2 3 3 4 1 2 1 3 4]
our second model score is 0.9753086419753086
```

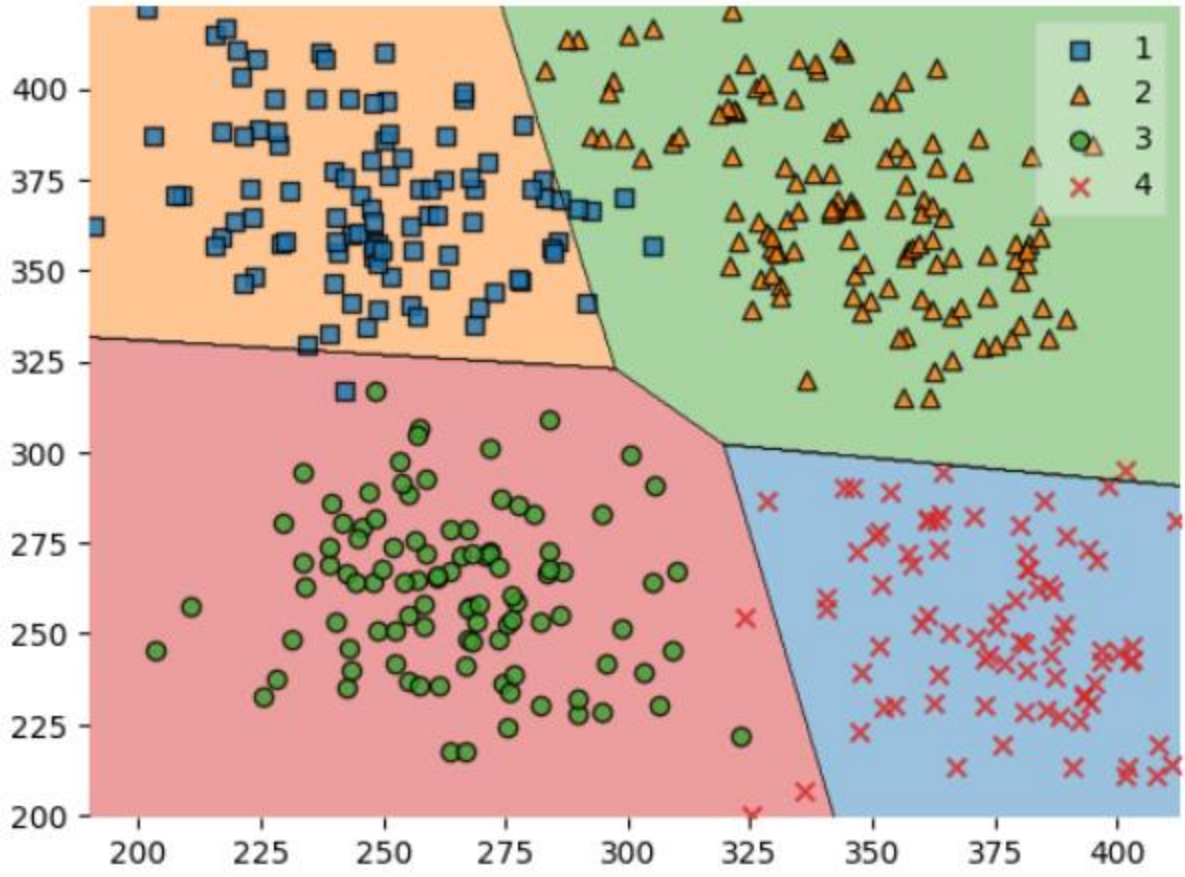
همان طور که مشاهده می‌شود،مدل اول که همان SGDClassifier است،خیلی عملکرد خوبی نداشته است.در عوض مدل دوم که RidgeClassifier بود عملکرد خیلی خوبی داشته است.پس با پیچیده تر شدن دیتا،می‌بینیم که تفاوت classifier ها به صورت چشم‌گیر تر مشاهده می‌شود.

مرز تصمیم گیری با مدل اول به صورت زیر است:



همان طور که مشاهده می‌شود،داده‌های کلاس های مختلف اصلا به خوبی از یکدیگر جدا نشده اند.

حال تفکیک داده ها با classifier دوم را در شکل زیر می‌بینیم:



همان طور که می‌بینیم،در این حالت کلاس بندی داده ها تقریبا با دقت خوبی انجام شده است.داده هایی که به درستی کلاس بندی نشده اند،بیشتر به خاطر محدودیت classifier های خطی می‌باشد.

سوال ۲)

۱. با مراجعه به صفحهٔ دیتاست CWRU Bearing با یک دیتاست مربوط به حوزهٔ «تشخیص عیب» آشنا شوید. با جستجوی آن در اینترنت و مقالات، توضیحاتی از اهداف، ویژگی‌ها و حالت‌های مختلف این دیتاست ارائه کنید.
در ادامه، ابتدا به صفحهٔ داده‌های سالم مراجعه کنید و داده‌های کلاس سالم (Normal_X[1]) را دریافت کنید. سپس، به صفحهٔ داده‌های عیب در حالت 12k مراجعه کرده و داده‌های کلاس عیب (IR007_X) را دریافت کنید.

مهم ترین بخش ماشین‌آلات دوار،یاتاقان های غلتشی هستند. یافتن عیوب بلبرینگ به موقع می تواند از تأثیرگذاری بر عملکرد کل تجهیزات جلوگیری کند. تشخیص خطا مبتنی بر داده فناوری یاتاقان ها اخیراً به یک کانون تحقیقاتی تبدیل شده است و نقطه شروع تحقیق اغلب به دست آوردن سیگنال های ارتعاشی است. مجموعه داده های عمومی زیادی برای یاتاقان های نورد وجود دارد. در میان آنها، پرکاربردترین مجموعه داده عمومی، مرکز باربری دانشگاه Case Western Reserve (CWRU) است.

داده‌های موجود در این دیتاست شامل DE (Drive end accelerometer data یا داده های شتاب سنج انتهای درایو)،FE (fan end accelerometer data یا داده های شتاب سنج انتهای فن)،BA (base accelerometer data یا داده های شتاب سنج پایه)، time series data (یا داده های سری‌های زمانی) و RPM در طول تست هستند که به دو دسته true و fault تقسیم می‌شوند.

---

۲. برای تشکیل دیتاست مراحل زیر را انجام دهید:

آ) از هر کلاس M نمونه با طول N جدا کنید (M حداقل ۱۰۰ و N حداقل ۲۰۰ باشد). یک ماتریس از داده‌های هر دو کلاس به همراه برچسب مربوطه تشکیل دهید. می‌توانید پنجره‌ای به طول N در نظر بگیرید و در نهایت یک ماتریس $M \times N$ از داده‌های هر کلاس استخراج کنید.

```
import pandas as pd
import numpy as np
data = pd.read_csv('/content/data_forq2.csv')
X_true = data[['x']].values
y_fault = data[['y']].values
X_true_set = np.array(X_true[:20000])
y_fault_set = np.array(y_fault[:20000])
X_true_set = X_true_set.reshape(100 , 200)
y_fault_set = y_fault_set.reshape(100 , 200)
```

در این کد ابتدا فایل csv مربوط به دیتا که قبلا با دستور gdown آورده شده خوانده می‌شود.سپس ۲۰۰۰۰ دیتا از ستون true و ۲۰۰۰۰ دیتا از ستون fault در یک آرایه ذخیره می‌شود.سپس هر یک از این ۲۰۰۰۰ دیتا به یک ماتریس ۱۰۰ در ۲۰۰ تبدیل می‌شود.

ب) در مورد اهمیت استخراج ویژگی در یادگیری ماشین توضیحاتی بنویسید. سپس، با استفاده از حداقل ۸ عدد از روش‌های ذکرشده در جدول ۱، ویژگی‌های دیتاست قسمت «۲ـآ» را استخراج کنید و یک دیتاست جدید تشکیل دهید.

جدول ۱: ویژگی‌های پیشنهادی برای استخراج از دیتاست.

Feature	Formula	Feature	Formula
Standard Deviation	$x_{std} = \sqrt{\frac{\sum_{i=1}^{N}(x(i)-\bar{x})^2}{N}}$	Shape Factor	$SF = \frac{x_{rms}}{\frac{1}{N}\sum_{i=1}^{N}\lvert x(i)\rvert}$
Peak	$x_p = \max\lvert x(i)\rvert$	Impact Factor	$IF1 = \frac{x_p}{\frac{1}{N}\sum_{i=1}^{N}\lvert x(i)\rvert}$
Skewness	$x_{ske} = \frac{\frac{1}{N}\sum_{i=1}^{N}(x(i)-\bar{x})^3}{x_{std}^3}$	Square Mean Root	$x_{smr} = \left(\frac{1}{N}\sum_{i=1}^{N}\sqrt{\lvert x(i)\rvert}\right)^2$
Kurtosis	$x_{kur} = \frac{\frac{1}{N}\sum_{i=1}^{N}(x(i)-\bar{x})^4}{x_{std}^4}$	Mean	$Mean = \frac{1}{n}\sum_{i=1}^{n} x_i$
Crest Factor	$CF = \frac{x_p}{x_{rms}}$	Absolute Mean	$Abs\ Mean = \frac{1}{n}\sum_{i=1}^{n}\lvert x_i\rvert$
Clearance Factor	$CLF = \frac{x_p}{x_{smr}}$	Root Mean Square	$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$
Peak to Peak	Maximum - Minimum	Impulse Factor	$IF2 = \frac{AbsMax}{\frac{1}{n}\sum_{i=1}^{n}\lvert x_i\rvert}$

هر مجموعه‌ای از داده ها که داریم،زمانی می‌توانیم آن ها را تفکیک کنیم که از نظر بعضی پارامترها تفاوت داشته باشند.مثلا میانگین،ماکزیمم و ... .در نتیجه با استفاده از داده های خام نمیتوان عملا طبقه بندی انجام داد.چون ممکن است برای کلاس های مختلف تفاوت ها در این داده های خام زیاد مشخص نشود.اما با استخراج ویژگی این تفاوت ها بهتر مشخص می‌شوند.

دلیل مهم دیگر برای استخراج ویژگی کاهش ابعاد دیتاست موردنظر است.مثلا در همین جا ما یک دیتاست ۲۰۰ در ۲۰۰ داریم(هر دو کلاس با هم)اما پس از استخراج ۱۰ ویژگی به ابعاد ۲۰۰ در ۱۰ کاهش پیدا می‌کند که حافظه کمتری مصرف می‌شود و این موضوع برای ما دارای اهمیت است.

تابع هایی که برای استخراج ویژگی نوشته شده اند به صورت زیر می‌باشند:

```
def Peak(x):
  return np.max(np.abs(x))
def Standard_deviation(x):
  sum = 0
  for i in range(len(x)):
    sum += np.power((x[i] - np.mean(x)) , 2)
  return(np.sqrt(sum/len(x)))
def Skewness(x):
  sum = 0
  for i in range(len(x)):
    sum += (np.power((x[i] - np.mean(x)) , 3))/len(x)
```

```
  return sum/(np.power(Standard_deviation(x) , 3))
def Kurtosis(x):
  sum = 0
  for i in range(len(x)):
    sum += (np.power((x[i] - np.mean(x)) , 4))/len(x)
  return sum/(np.power(Standard_deviation(x) , 4))
def RMS(x):
  sum = 0
  for i in range(len(x)):
    sum += np.power(x[i] , 2)
  return np.sqrt((1/len(x))*sum)
def Crest_Factor(x):
  return((Peak(x))/(RMS(x)))
def SMR(x):
  sum = 0
  for i in range(len(x)):
    sum += np.sqrt(np.abs(x[i]))
  return (np.power(sum/len(x) , 2))
def Clearance_Factor(x):
  return((Peak(x))/(SMR(x)))
def Peak_to_Peak(x):
  return (np.max(x) - np.min(x))
def Mean(x):
  return np.mean(x)
def feature(x):
  X_feature = []
  for i in range(100):
    X_feature.append([Peak(x[i]) , Standard_deviation(x[i]) , Skewness(x[i])
, Kurtosis(x[i]) , RMS(x[i]) , Crest_Factor(x[i]) , SMR(x[i]) ,
Clearance_Factor(x[i]) , Peak_to_Peak(x[i]) , Mean(x[i])])
  X_feature = np.array(X_feature)
  X_feature = X_feature.reshape(100 , 10)
  return X_feature
```

۱۰ تابع اول مطابق جدول بالا،۱۰ ویژگی از دیتا را به ما می‌دهند.تابع آخر(feature) یک دیتاست را از ما به عنوان ورودی می‌گیرد و ویژگی های آن را استخراج کرده و دیتاست ویژگی ها را به عنوان خروجی می‌دهد.

ج) ضمن توضیح اهمیت فرآیند بُرزدن (مخلوط کردن)[۲]، داده‌ها را مخلوط کرده و با نسبت تقسیم دلخواه و معقول به دو بخش «آموزش» و «ارزیابی» تقسیم کنید.

مخلوط کردن داده ها از این نظر حائز اهمیت است که در آموزش یک ماشین نباید هیچ نظم خاصی داشته باشند تا در فرایند یادگیری سوگیری خاصی اتفاق نیفتد.مخلوط کردن داده ها باعث می‌شود ترتیب قرارگیری داده ها به صورت تصادفی باشد و یادگیری ماشین با صحت بیشتری انجام شود و در نتیجه پیش بینی های ماشین بهتر باشد.

برای این کار از کد زیر استفاده می‌کنیم:

```
feature_X_true = feature(X_true_set)
feature_X_with_label = np.append(feature_X_true , one_col , axis = 1)
feature_y_fault = feature(y_fault_set)
feature_y_with_label = np.append(feature_y_fault , zero_col , axis = 1)
feature_data_with_label = np.vstack((feature_X_with_label ,
feature_y_with_label))
X_shuffle , y_shuffle = shuffle(feature_data_with_label[: , :10] ,
feature_data_with_label[: , 10])
y_shuffle = y_shuffle.reshape(200 , 1)
shuffled_data = np.append(X_shuffle , y_shuffle , axis = 1)
X_train , X_test , y_train , y_test = train_test_split(shuffled_data[: , :10]
,
                                                       shuffled_data[: , 10]
,
                                                       test_size = 0.2 ,
                                                       random_state = 74)
```

در این کد ابتدا ویژگی های داده ها را استخراج کرده سپس برچسب گذاری می‌کنیم.سپس داده‌ها را مخلوط کرده و مجدد برچسب ها را به داده مربوطه دوباره میزنیم.سپس آن را با نسبت ۸۰ به ۲۰ به دو دسته آموزش و ارزیابی تقسیم می‌کنیم.

د) حداقل دو روش برای نرمال‌سازی داده‌ها را با ذکر اهمیت این فرآیند توضیح دهید و با استفاده از یکی از این روش‌ها، داده‌ها را نرمال کنید. آیا از اطلاعات بخش «ارزیابی» در فرآیند نرمال‌سازی استفاده کردید؟ چرا؟

اهمیت نرمال سازی از این جهت است که ما گاهی با دیتاست‌هایی روبرو هستیم که فاصله ی داده‌ها در آن ها زیاد است.با نرمال سازی فاصله این داده ها کم می‌شود.

روش اول:Min-Max scaling

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

روش دوم:Z-score Standardization

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

$$\sigma = standard\ deviation\ of\ data\ , \qquad \mu = mean$$

روش سوم:Robust Scaling

$$X_{robust} = \frac{X - median(X)}{IQR(X)}$$

For Datasets with Odd Numbers:

$$\text{median} = \left(\frac{n+1}{2}\right)^{th} \text{term}$$

For Datasets with Even Numbers:

$$\text{median} = \frac{\left(\frac{n}{2}\right)^{th} \text{term} + \left(\frac{n}{2}+1\right)^{th} \text{term}}{2}$$

$$IQR = Q3 - Q1$$

- IQR = interquartile range
- Q3 = 3rd quartile or 75th percentile
- Q1 = 1st quartile or 25th percentile

کدی که برای نرمال سازی وجود دارد به صورت زیر است:(روش Min_Max)

```
scaler = MinMaxScaler()
X_tr_normal = scaler.fit_transform(X_train)
X_te_normal = scaler.fit_transform(X_test)
X_trl_normal = np.append(X_tr_normal , y_train , axis = 1)
X_tel_normal = np.append(X_te_normal , y_test , axis = 1)
```

ابتدا داده ها را نرمال و سپس برچسب می‌زنیم.

نرمال سازی هم روی داده های آموزش و هم روی داده های ارزیابی باید انجام شود.زیرا اگر فقط روی یکی انجام شود،خطا در پیش بینی داده های تست خواهیم داشت.(اگر فقط داده های آموزش را نرمال کنیم)

۳. بدون استفاده از کتابخانه‌های آمادهٔ پایتون، مدل طبقه‌بند، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کدنویسی کنید تا دو کلاس موجود در دیتاست به خوبی از یکدیگر تفکیک شوند. نمودار تابع اتلاف را رسم کنید و نتیجهٔ ارزیابی روی داده‌های تست را با حداقل ۲ شاخصه محاسبه کنید. نمودار تابع اتلاف را تحلیل کنید. آیا می‌توان از روی نمودار تابع اتلاف و قبل از مرحلهٔ ارزیابی با قطعیت در مورد عمل‌کرد مدل نظر داد؟ چرا و اگر نمی‌توان، راه‌حل چیست؟

کد به صورت زیر است:

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def logistic_regression(x, w):
    y_hat = sigmoid(x @ w)
    return y_hat
def bce(y, y_hat):
    loss = -(np.mean(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)))
    return loss
def gradient(x, y, y_hat):
    grads = (x.T @ (y_hat - y)) / len(y)
    return grads
def gradient_descent(w, eta, grads):
    w -= eta*grads
    return w
def accuracy(y, y_hat):
    acc = np.sum(y == np.round(y_hat)) / len(y)
    return acc
```

ابتدا توابع موردنیاز برای آموزش نوشته شده اند.

```
w = np.random.randn(11, 1)
eta = 0.1
n_epochs = 25
```

وزن‌ها را ابتدا به صورت رندوم در نظر می‌گیریم و نرخ یادگیری را برابر ۰.۱ و تعداد دوره‌های آموزش را ۲۵ درنظر میگریم.

```
error_hist = []

for epoch in range(n_epochs):
    # predictions
    y_hat = logistic_regression(X_trl_normal, w)

    # loss
    e = bce(y_train, y_hat)
    error_hist.append(e)

    # gradients
    grads = gradient(X_trl_normal, y_train, y_hat)

    # gradient descent
    w = gradient_descent(w, eta, grads)

    if (epoch+1) % 10 == 0:
        print(f'Epoch={epoch}, \t E={e:.4},\t w={w.T[0]}')
```

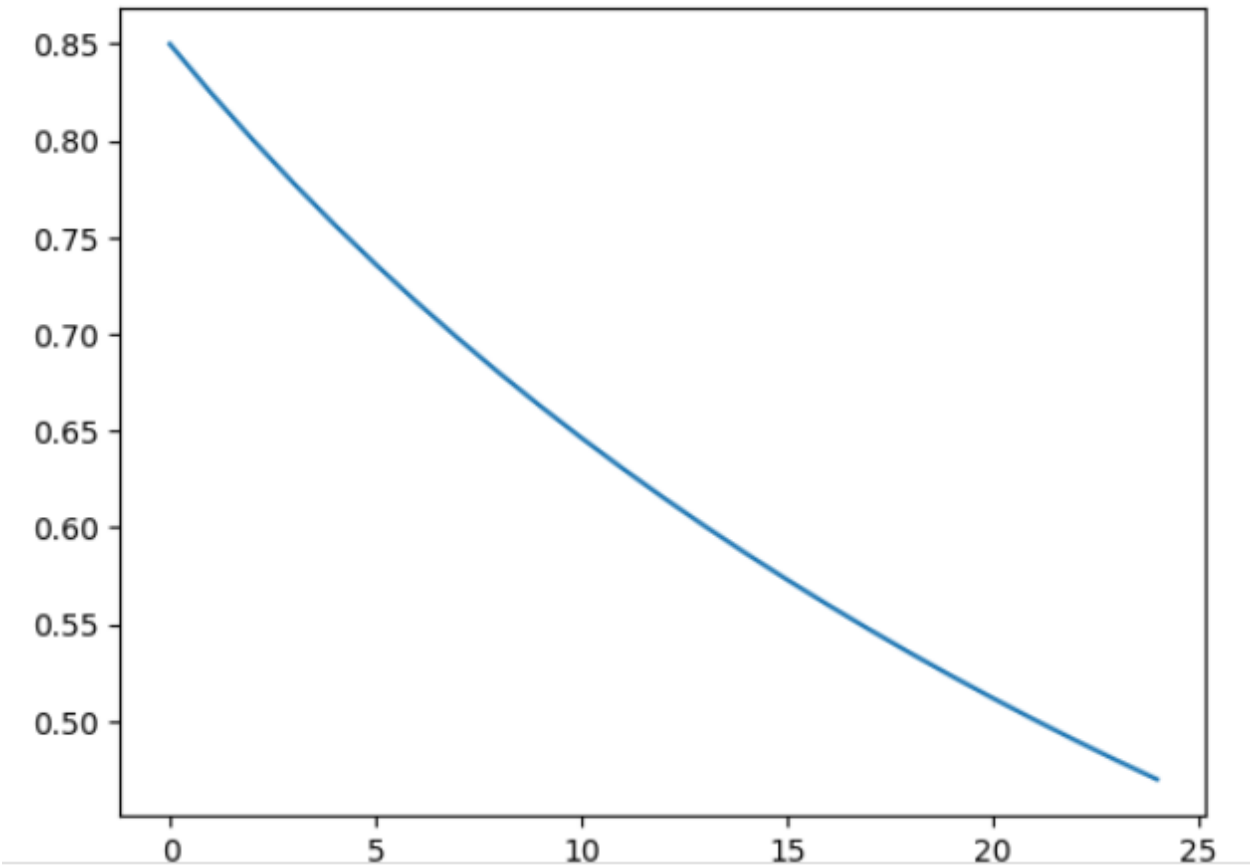
کد بالا مربوط به آموزش مدل است.در هر حلقه ابتدا با وزن تصادفی که در نظر گرفتیم پیش بینی را انجام می‌دهیم.سپس مقدار اتلاف را حساب می‌کنیم.بعد از آن با توجه به اختلاف با مقدار درست وزن ها را در هر حلقه آپدیت می‌کنیم.این کار همان طور که گفته شد ۲۵ بار انجام می‌شود.همچنین در آخر هر ۱۰ دوره یک بار مقدار اتلاف و وزن‌های آپدیت شده نمایش داده می‌شود که نتیجه به صورت زیر است:

```
Epoch=9,     E=0.6629,    w=[-0.00948261 -0.49251315 -1.67021071  1.35322248 -
0.50993579  0.97304956
  0.05474591 -0.79913267 -0.06003181  0.95728802 -0.24350252]
Epoch=19,    E=0.5234,    w=[-0.0989444  -0.58540813 -1.61571402  1.30900695 -
0.60369901  0.94080944
 -0.03172728 -0.83983484 -0.15699666  1.05203053  0.03287543]
```

همان طور که مشاهده می‌شود در حلقه ۹ و ۱۹ مقدار اتلاف و وزن‌های جدید را داریم.

```
import matplotlib.pyplot as plt
plt.plot(error_hist)
```

در این کد اتلاف را برای این مدل رسم کرده ایم که به صورت زیر است:



همان طور که مشاهده می‌شود مقدار اتلاف در طول آموزش کاهشی بوده است که نشان می‌دهد عملکرد مدل خوب بوده است.اما لزوما دلیل بر نتیجه خوب نمی‌باشد.زیرا کاهشی بودن اتلاف وقتی خوب است که در نهایت بهترین وزن ها را به ما بدهد.ممکن است اتلاف کاهش یابد اما در هر دوره به اندازه ای کاهش نیابد که با پایان دوره های آموزش وزن خیلی خوبی را بدهد.در واقع باید این نمودار به نزدیکی صفر برسد تا بفهمیم مدل به خوبی آموزش دیده است.

```
y_hat = logistic_regression(X_tel_normal, w)
accuracy(y_test, y_hat)
```

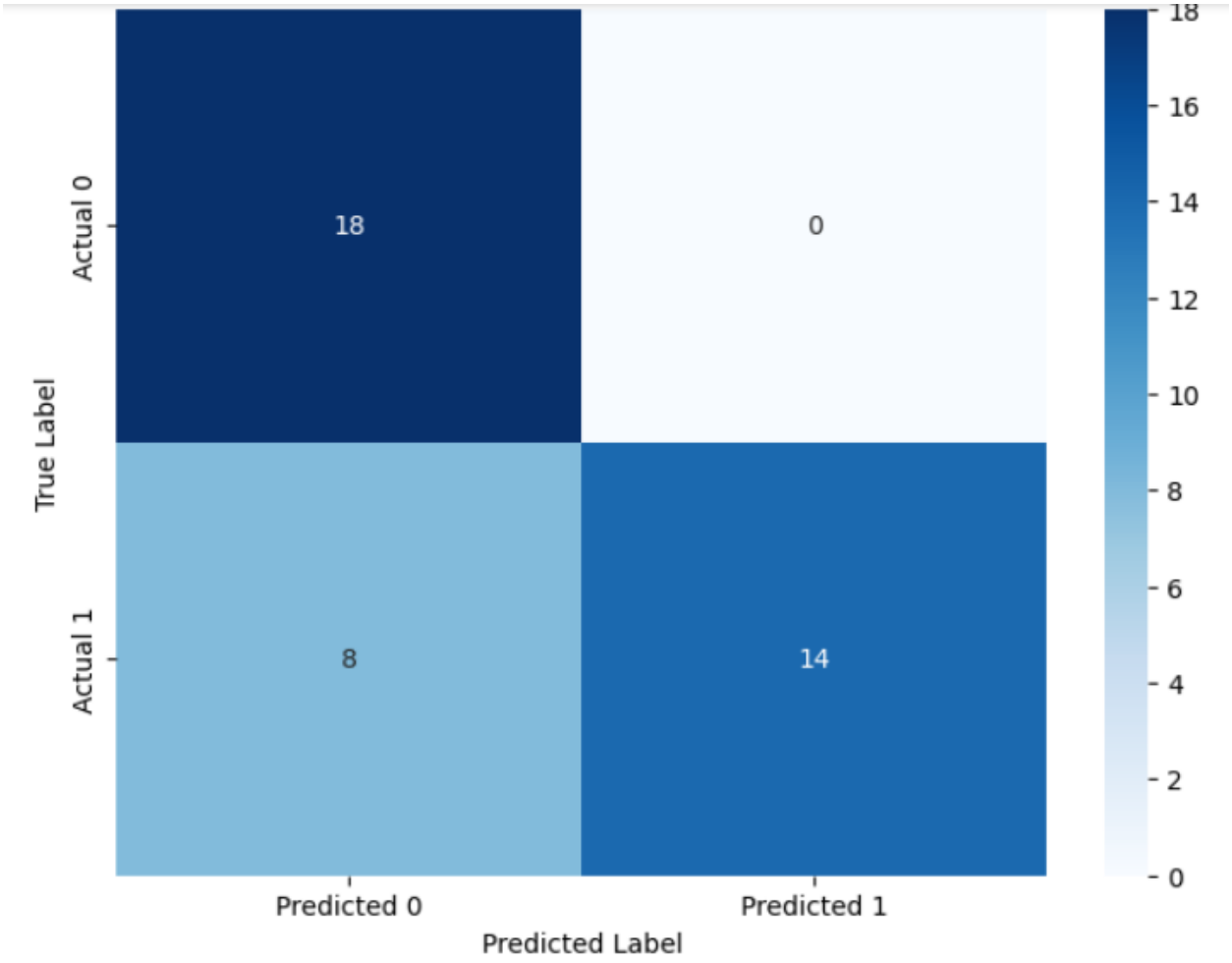
```
0.8
```

در این قسمت با استفاده از داده های تست،مدل را ارزیابی می‌کنیم.همان طور که مشاهده می‌شود accuracy برای این مدل ۸۰ درصد است که تقریبا عملکرد خوبی است.

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import accuracy_score
cm = confusion_matrix(y_test, np.round(y_hat))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

تکه کد بالا confusion matrix را برای مدل ما رسم می‌کند.داریم:



همان طور که مشاهده می‌شود مدل تمام داده هایی که برچسب صفر داشته اند را درست پیش بینی کرده است؛اما از ۲۲ داده که برچسب یک داشته اند،۱۴ تا را تشخیص داده است.

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test , np.round(y_hat))
f1
```

```
0.7777777777777778
```

کد بالا f1 score را برای این مدل حساب می‌کند که برابر حدود ۰.۷۸ می‌باشد.

طبق تحلیلی که روی نمودار اتلاف داشتیم،اگر این نمودار کاهشی باشد و به سمت صفر برود،این مدل عملکرد خوبی دارد.البته اگر این کاهش قوس رو به داخل داشته باشد،نشان می‌دهد سریع تر این مدل توانسته اتلاف را کاهش دهد.اما بهترین روش برای ارزیابی مدل،همان مرحله ارزیابی می‌باشد.

۴. فرآیند آموزش و ارزیابی را با استفاده از یک طبقه‌بند خطی آمادهٔ پایتون (در sklearn.linear_model) انجام داده و نتایج را مقایسه کنید. در حالت استفاده از دستورات آمادهٔ سایکیت‌لرن، آیا راهی برای نمایش نمودار تابع اتلاف وجود دارد؟ پیاده‌سازی کنید.

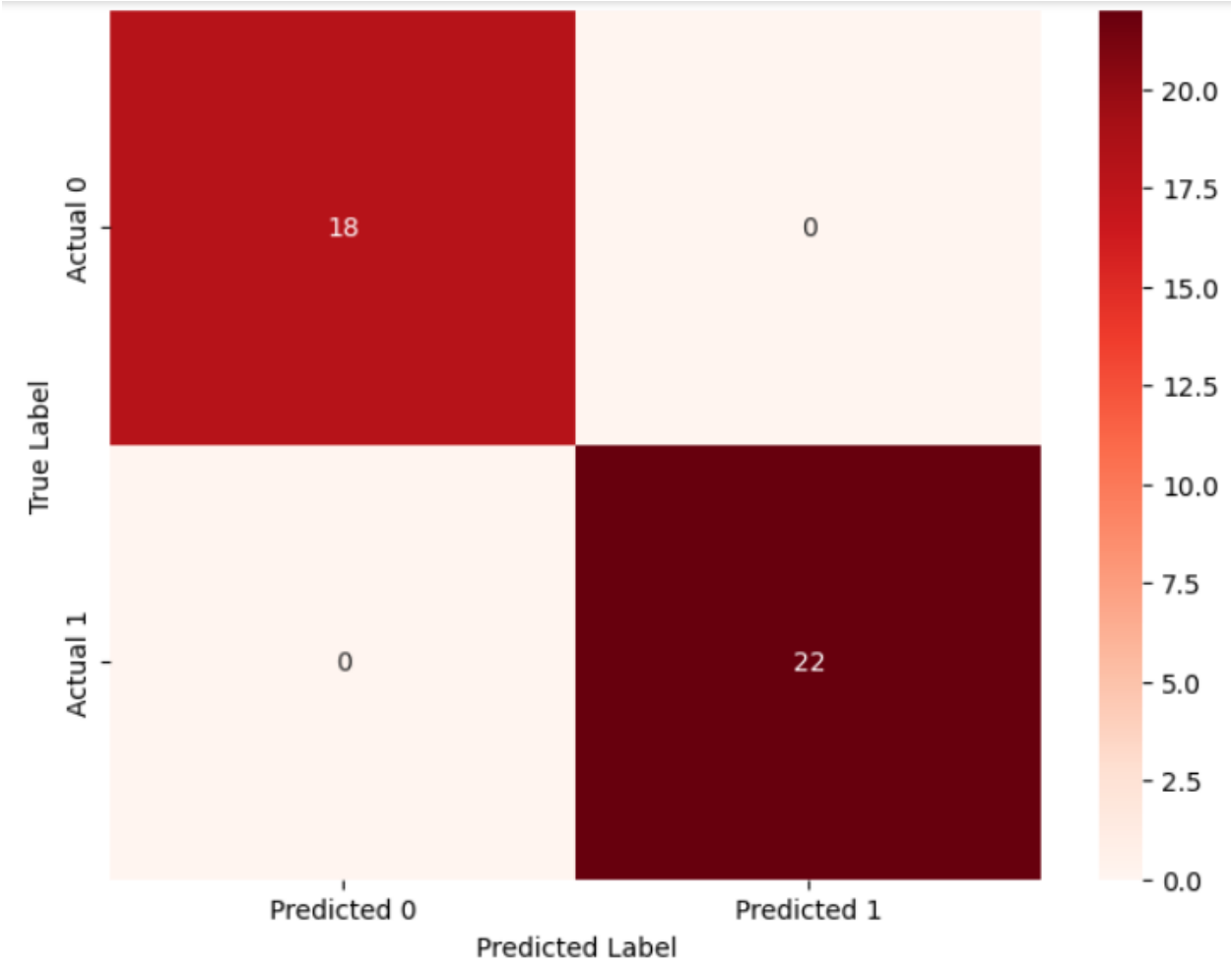
```
model_1 = SGDClassifier(max_iter = 25 , alpha = 0.01 , random_state = 74)
model_1.fit(X_tr_normal , y_train)
y_pred_normal = model_1.predict(X_te_normal)
# first 10 samples
print(f"first 10 predict is:{y_pred_normal[:10]}\nfirst 10 true data
is:{y_test[:10]}\n")
model_1_score = model_1.score(X_te_normal , y_test)
print(f"our model score is:{model_1_score}")
cm2 = confusion_matrix(y_test, model_1.predict(X_te_normal))
plt.figure(figsize=(8, 6))
sns.heatmap(cm2, annot=True, cmap='Reds', fmt='g',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

در این کد با استفاده از مدل آماده‌ی SGDClassifier فرایند آموزش و ارزیابی انجام شده است.

نتایج به صورت زیر است:

```
first 10 predict is:[0. 0. 0. 1. 1. 1. 1. 0. 1. 1.]
first 10 true data is:[[0.]
 [0.]
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [0.]
 [1.]
 [1.]]

our model score is:1.0
```

همان طور که مشاهده می‌شود عملکرد این مدل بهتر از مدل قبلی است و accuracy برابر یک است.حال confusion matrix را مشاهده می‌کنیم:



که همان طور که مشاهده می‌شود همه ی داده ها درست تشخیص داده شده اند.

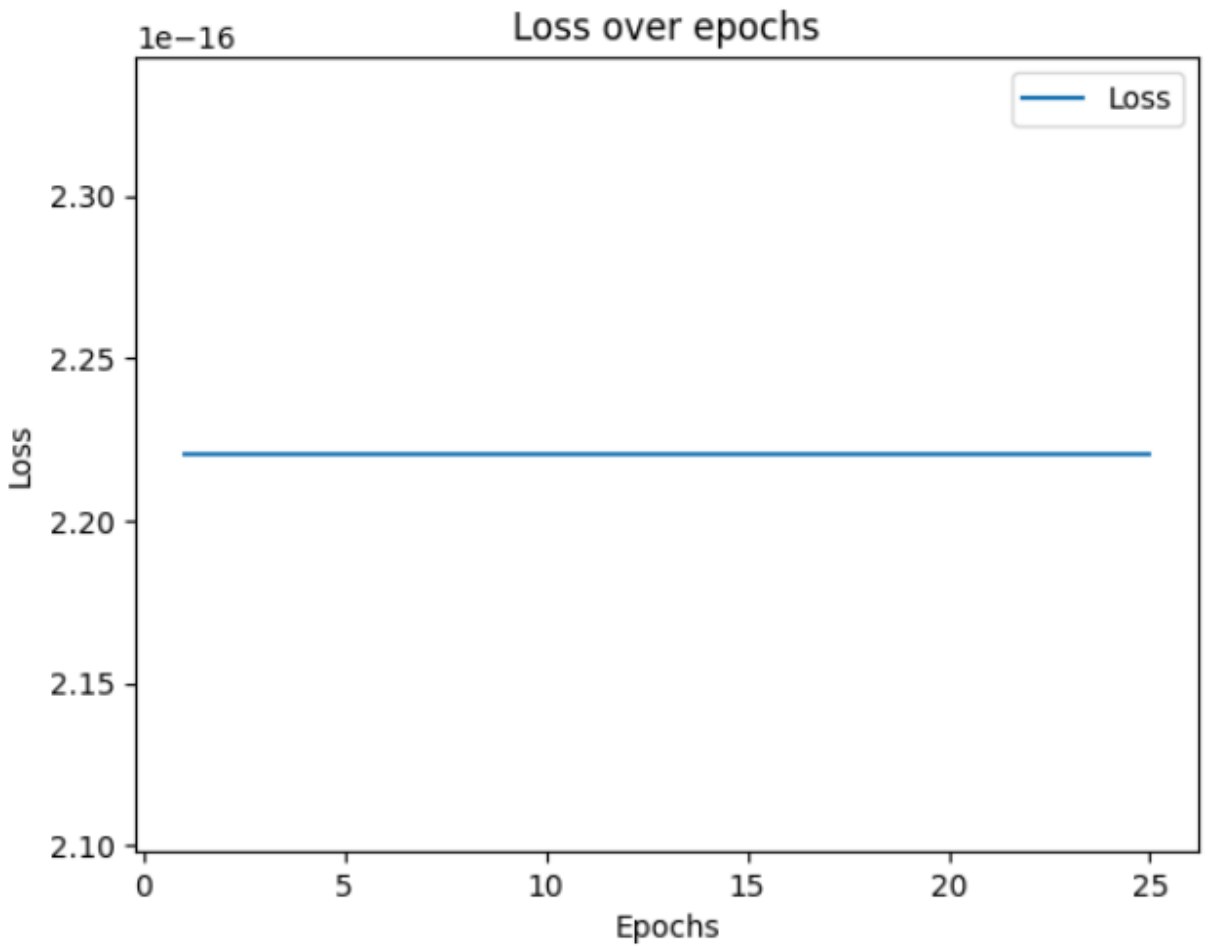
```
f2 = f1_score(y_test , model_1.predict(X_te_normal))
f2
```

```
1.0
```

برای این مدل f1 score هم برابر ۱ است.

```
def app_loss(model , loss_list):
  model.fit(X_tr_normal , y_train)
  y_pred = model.predict(X_te_normal)
  loss = log_loss(y_test , y_pred)
  loss_list.append(loss)

models = []
for j in range(25):
  models.append(SGDClassifier(max_iter = j+1 , alpha = 0.01 , random_state =
74))
losses = []
for i in range(25):
  app_loss(model = models[i] , loss_list = losses)


print("Final Losses:", losses)
plt.plot(range(1, i+2), losses, label='Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

برای رسم نمودار loss برای مدل‌های آماده به تعداد دوره آموزش classifier موردنظر را تعریف کردیم و تعداد دوره آموزش آن ها را به ترتیب از ۱ تا تعداد دوره آموزش مدل اصلی قرار دادیم و سپس برای هر یک loss را حساب کردیم و همه را در یک لیست ذخیره کردیم.سپس آن را رسم کردیم.نتیجه به صورت زیر است:



به این دلیل loss تغییری نکرده که SGDClassifier از همان ابتدا accuracy برابر ۱ را می‌دهد و با بالا رفتن max_iter تفاوتی در loss به وجود نمی‌آید.احتمالا این موضوع به دلیل سادگی دیتاست می‌باشد.

سوال۳)

۳ سوال سوم

یک دیتاست در زمینهٔ آب و هوا با نام Weather in Szeged 2006-2016 را در نظر بگیرید. در این دیتاست هدف آن است که ارتباط بین Humidity با Temperature و هم‌چنین ارتباط بین Humidity و Apparent Temperature پیدا شده و با کمک داده‌های Humidity و Temperature تخمین انجام شود.

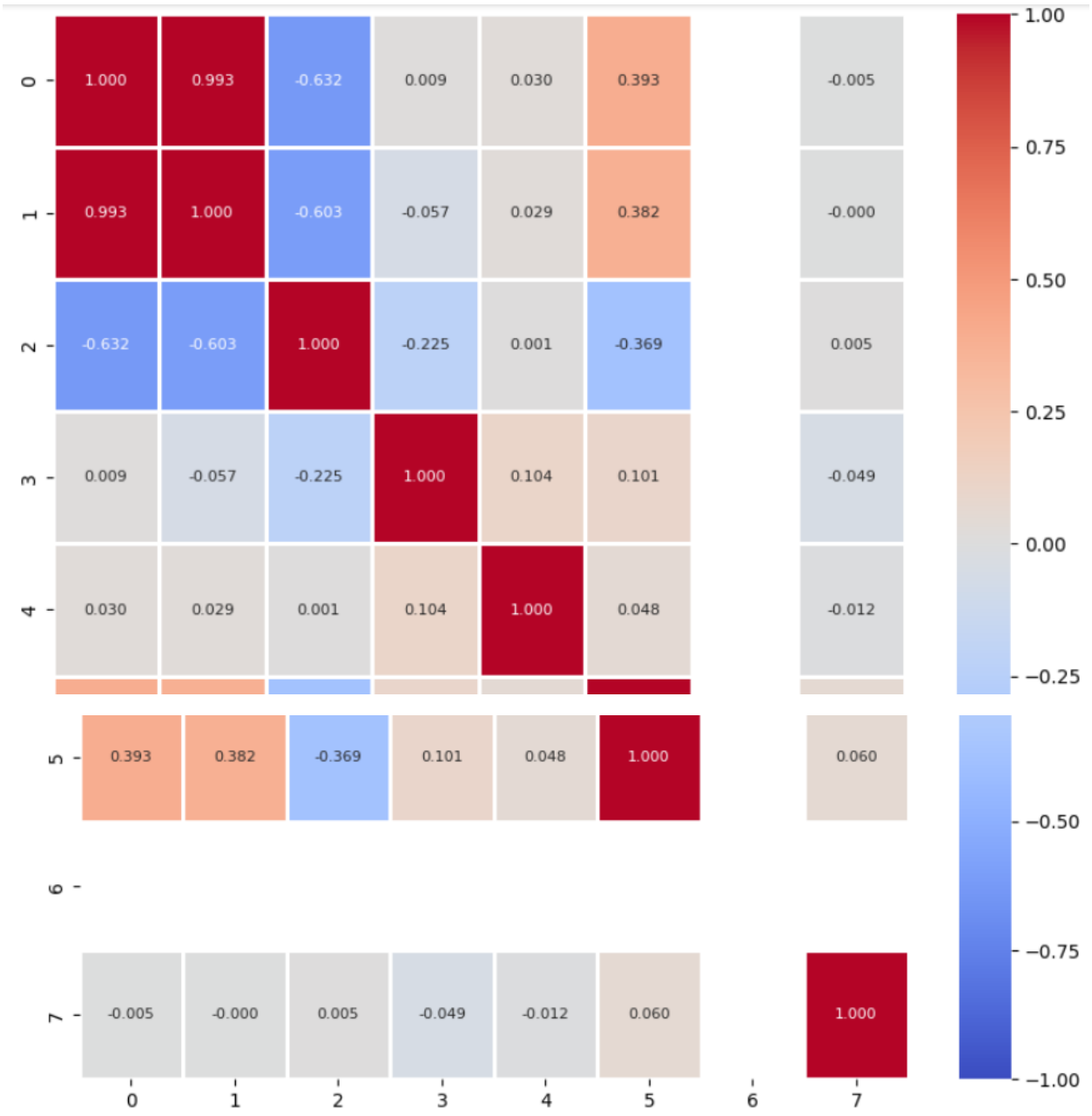
۱. ابتدا هیت‌مپ ماتریس همبستگی و هیستوگرام پراکندگی ویژگی‌ها را رسم و تحلیل کنید.

```
import seaborn as sns
import pandas as pd
!gdown 1xFKHtOOhwaxz686lVuPPXdzuEjheiL2z
air_data = pd.read_csv('/content/weatherHistory.csv')
air_data_np = np.array(air_data)
pd_data = pd.DataFrame(air_data_np[: , 3:11])
```

ابتدا دیتا از یک فایل csv از گوگل درایو فراخوانی شده و سپس ستون ۳ تا ۱۰ که دارای دیتای عددی است استخراج می‌شود.

```
import matplotlib.pyplot as plt
cor_matrix = pd_data.corr()
plt.figure(figsize=(10, 10))
sns.heatmap(cor_matrix, annot=True, cmap='coolwarm', linewidths=1,
annot_kws={"size": 8}, fmt='.3f',
            yticklabels=cor_matrix.columns, vmin=-1, vmax=1)
```

با استفاده از کد بالا correlation matrix رسم می‌شود که به صورت زیر است:

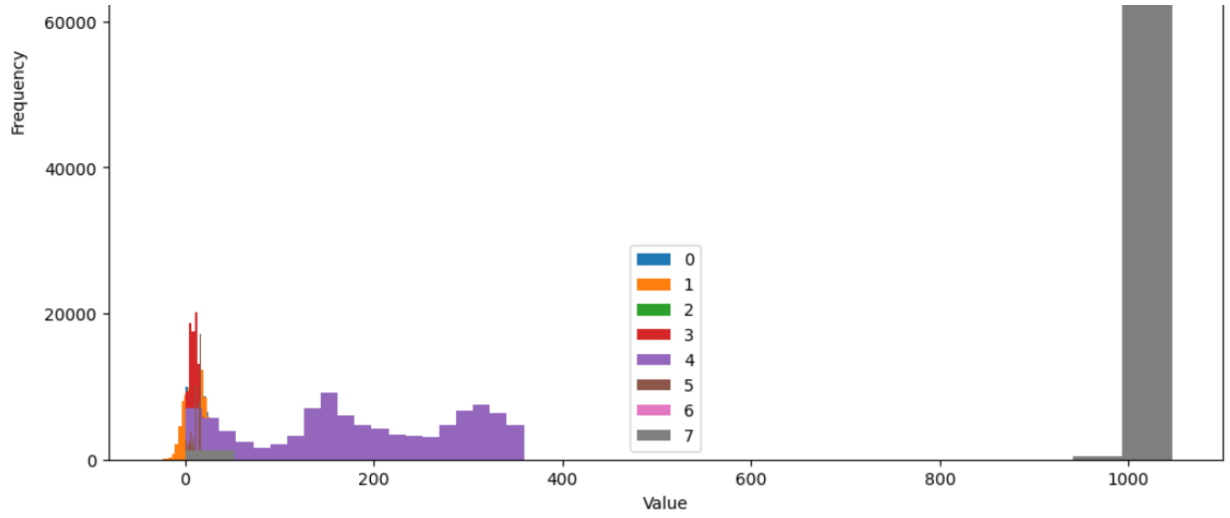


همان طور که در این ماتریس مشاهده می‌شود،از این ۸ ستون دیتا،ستون۱ و ۲ وابستگی زیادی دارند.هر چه این عدد بزرگتر و نزدیک به ۱ باشد،وابستگی دو دیتا زیادتر است.همچنین اگر این عدد منفی باشد،وابستگی معکوس بین دو دیتا به اندازه آن عدد وجود دارد.

```
plt.figure(figsize=(12, 8))  # Adjust figure size as needed
for column in pd_data.columns:
    plt.hist(pd_data[column], bins=20, alpha=1, label=column)  # Adjust
number of bins as needed
plt.xlabel('Value')
plt.ylabel('Frequency')
```

```
plt.title('Histograms of Each Column')
plt.legend()
plt.show()
```

کد بالا هیستوگرام مربوط به دیتاهای ما را رسم می‌کند.نتیجه به صورت زیر است:



همان طور که مشاهده بیشتر پراکندگی دیتاها اطراف ۰ تا ۳۰ است.یکی از دیتاها که با رنگ بنفش نشان داده شده از صفر تا حدود ۳۵۰ است.یکی از دیتاها هم که با رنگ خاکستری نشان داده شده است در حدود ۱۰۰۰ است.

۲. روی این دیتاست، تخمین LS و RLS را با تنظیم پارامترهای مناسب اِعمال کنید. نتایج به‌دست‌آمده را با محاسبهٔ خطاها و رسم نمودارهای مناسب برای هر دو مدل با هم مقایسه و تحلیل کنید.

LS

رگرسیون حداقل مربعات یک مدل قطعی است، به این معنی که بر خلاف سایر مدل‌های تصادفی، خروجی یا وزن‌های محاسبه‌شده به حالت الگوریتم بستگی ندارد. در عوض، آنها فقط به داده های ورودی بستگی دارند. و از این رو نیازی به تکرار نیست.هدف از حداقل مربعات تلاش برای یافتن

خطی است که بهترین مطابقت با مجموعه داده را دارد، نوعی خط که وقتی بر مجموعه نقاط داده داده شده به عنوان ورودی اعمال می شود، کمترین خطای ممکن را خواهد داشت.

در روش حداقل مربعات معمولی، سعی می کنیم با به حداقل رساندن مجذور اختلاف بین مقدار پیش بینی شده و مقدار مشاهده شده یک متغیر وابسته معین، یک خط مستقیم را بر روی نقاط داده قرار دهیم.

کد مربوط به LS به صورت زیر است:

```
import pandas as pd
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression

X = air_data['Humidity'].values.reshape(-1, 1)
y = air_data['Temperature (C)'].values

X = np.column_stack([np.ones_like(X), X])

coefficients = np.linalg.inv(X.T @ X) @ X.T @ y

intercept, slope = coefficients

y_pred = intercept + slope * X[:, 1]
mse = np.mean((y - y_pred)**2)
print("Mean Squared Error:", mse)

plt.scatter(X[:, 1], y, label='Data')
plt.plot(X[:, 1], y_pred, color='red', label='Regression Line')
plt.xlabel('Humidity')
plt.ylabel('Temperature')
plt.title('Temperature vs Humidity')
plt.legend()
plt.show()
```

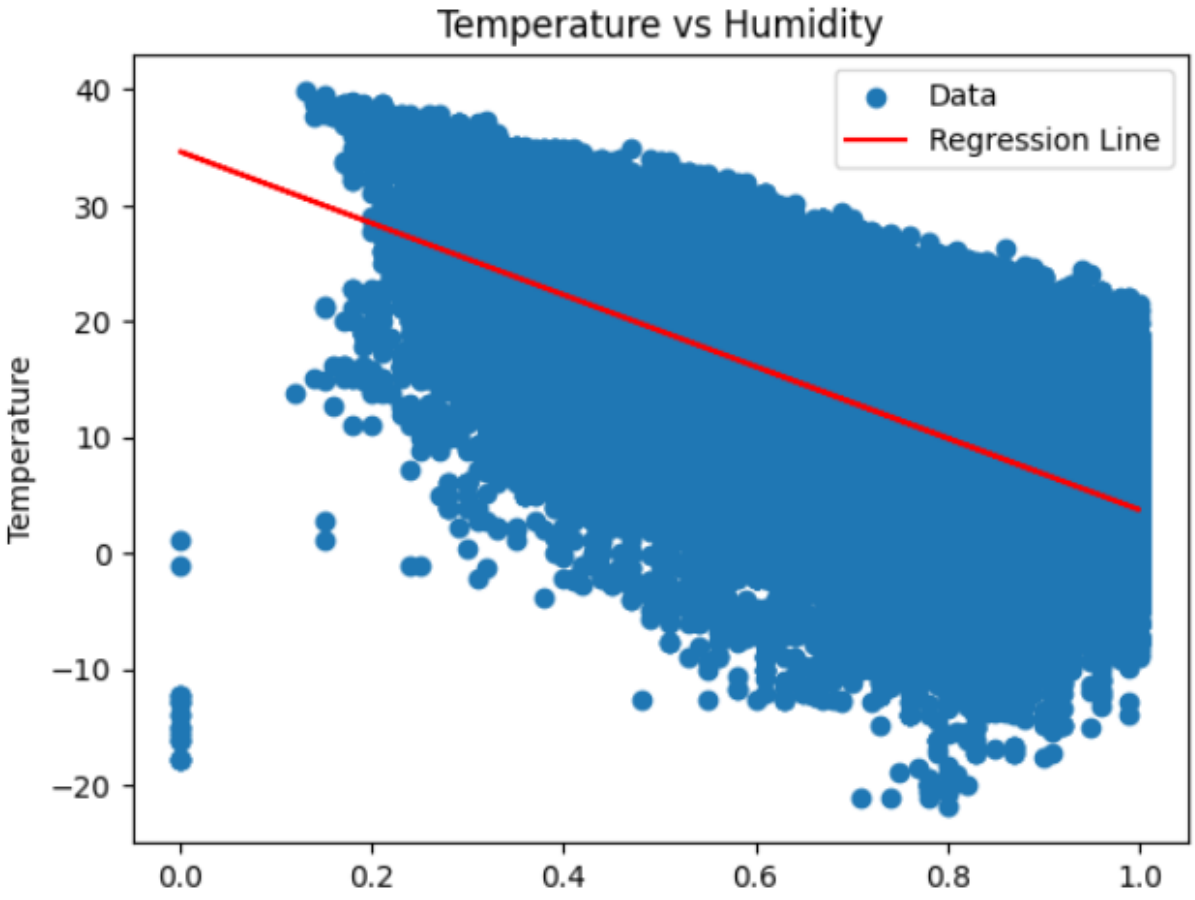
در این کد از همان رابطه‌ی مربوط به LS استفاده شده است.

$$\underline{\hat{\theta}} = \left(\underline{X}^T \underline{Q}\, \underline{X}\right)^{-1} \underline{X}^T \underline{Q}\, \underline{y}.$$

همچنین برای خطا از میانگین مربعات خطا استفاده شده است.

نتیجه به صورت زیر است:

```
Mean Squared Error: 54.761829807719856
```




```
Intercept: 34.636929126889186
Slope: -30.89438375805085
```

برای این مدل،مقدار خطا برابر حدود ۵۴.۷۶ است.با توجه به این که دیتاهای زیادی داریم و خیلی از هم پراکنده هستند،این مقدار از خطا طبیعی است.همچنین خط در نظر گرفته شده با توجه به قرارگیری داده ها تقریبا خوب است.

دو مقدار آخر هم عرض از مبدا و شیب خط را نشان می‌دهند.

RLS

یک الگوریتم فیلتر تطبیقی است که برای تخمین آنلاین پارامترها در یک مدل خطی استفاده می شود. این الگوریتم به ویژه زمانی مفید است که داده‌های جریانی دارید و نیاز دارید مدل خود را به

طور مداوم بر اساس مشاهدات جدید به‌روزرسانی کنیم. RLS گسترش روش حداقل مربعات معمولی است، اما با در دسترس قرار گرفتن داده های جدید، پارامترهای مدل را به طور مکرر به روز می کند.

ثابت forgetting factor ثابتی است که تاثیر مشاهدات گذشته را بر برآورد فعلی کنترل می‌کند.این ثابت عددی بین صفر و یک است.هر چه این عدد بزرگتر باشد،مشاهدات اخیر وزن بیشتری دارند و مشاهدات قدیمی تر سریعتر فراموش می‌شوند و هر چه کوچکتر باشد،کندتر فراموش می‌شوند.تنظیم این ثابت کاملا بستگی به کاربرد مدنظر ما دارد.

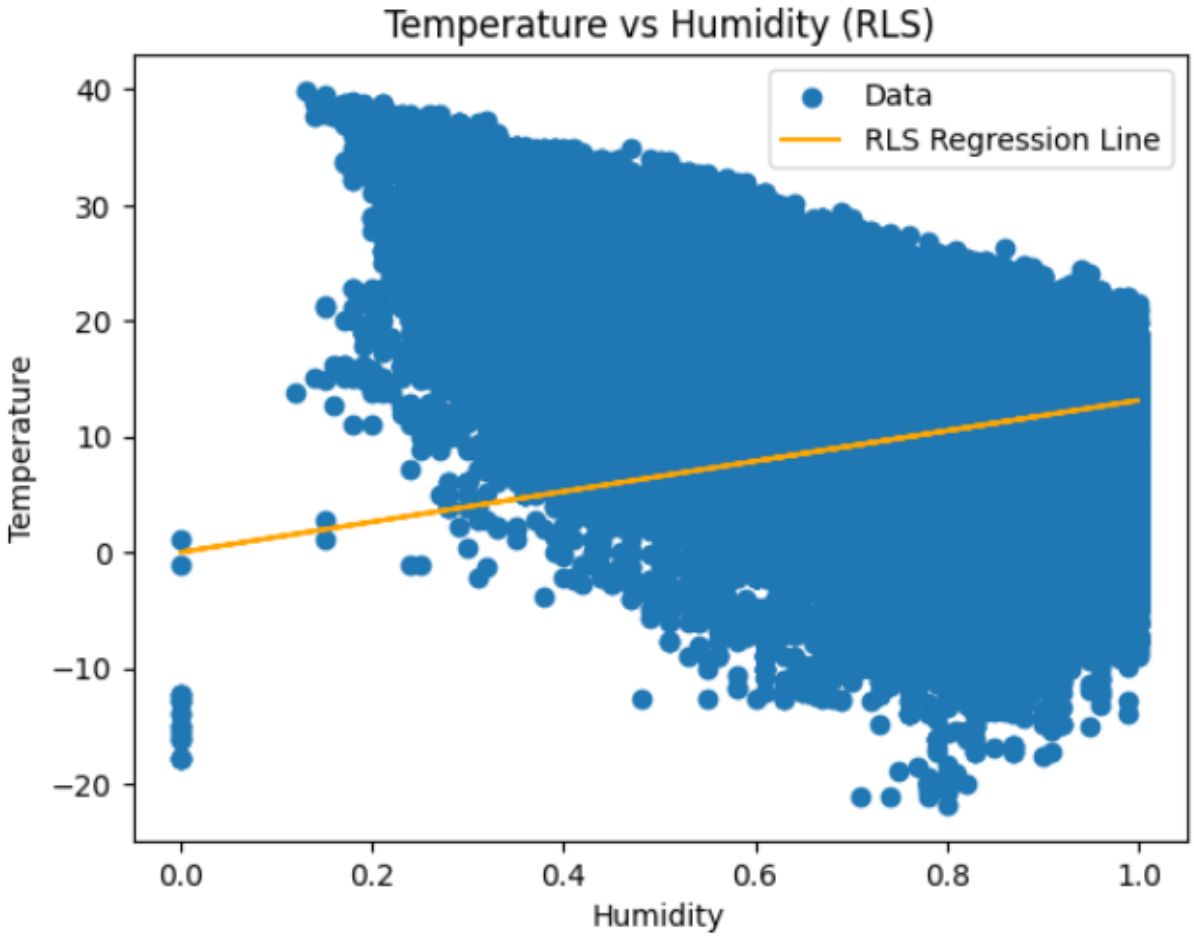
کد به صورت زیر است:

```
class RecursiveLeastSquares:
    def __init__(self, n_features, forgetting_factor=1.0):
        self.n_features = n_features
        self.forgetting_factor = forgetting_factor
        self.theta = np.zeros((n_features, 1))
        self.P = np.eye(n_features)

    def update(self, x, y):
        x = np.atleast_2d(x)
        y = np.atleast_1d(y)


        err = y - np.dot(x, self.theta)


        K = np.dot(self.P, x.T) / (self.forgetting_factor + np.dot(np.dot(x,
self.P), x.T))


        self.theta += np.dot(K, err)
        self.P = (1 / self.forgetting_factor) * (self.P - np.dot(K, np.dot(x,
self.P)))

    def error(self , x , y):
        x = np.atleast_2d(x)
        y = np.atleast_1d(y)


        err = y - np.dot(x, self.theta)
        return err
    def predict(self, x):
        x = np.atleast_2d(x)
        return np.dot(x, self.theta)

import pandas as pd
import matplotlib.pyplot as plt
```

```
X = air_data['Humidity'].values.reshape(-1, 1)
y = air_data['Temperature (C)'].values

model = RecursiveLeastSquares(n_features=1)
err_list = []

for i in range(len(X)):
    model.update(X[i], y[i])
    err_list.append(model.error(X[i] , y[i]))



y_pred = [model.predict(x)[0, 0] for x in X]


mse = np.mean(np.array(err_list)**2)
print("Mean Squared Error:", mse)
plt.scatter(X, y, label='Data')
plt.plot(X, y_pred, color='orange', label='RLS Regression Line')
plt.xlabel('Humidity')
plt.ylabel('Temperature')
plt.title('Temperature vs Humidity (RLS)')
plt.legend()
plt.show()
```

و نتیجه به صورت زیر است:

```
Mean Squared Error: 134.02601813760666
```



مقدار خطا در این حالت برابر حدود ۱۳۴ است.همان طور که مشاهده می‌شود،مقدار خطای این روش بیشتر از LS است.همچنین خط کشیده شده در این حالت باز هم نسبتا تخمین خوبی است.

۳. در مورد Weighted Least Square توضیح دهید و آن را روی دیتاست دادشده اِعمال کنید.

WLS

در واقع تفاوت اصلی Weighted Least Square با LS این است که به مشاهداتی که داریم بر اساس اهمیت آن برای ما وزن هایی اختصاص داده می‌شود تا با تاثیر وزن وارد مدل ما بشوند.

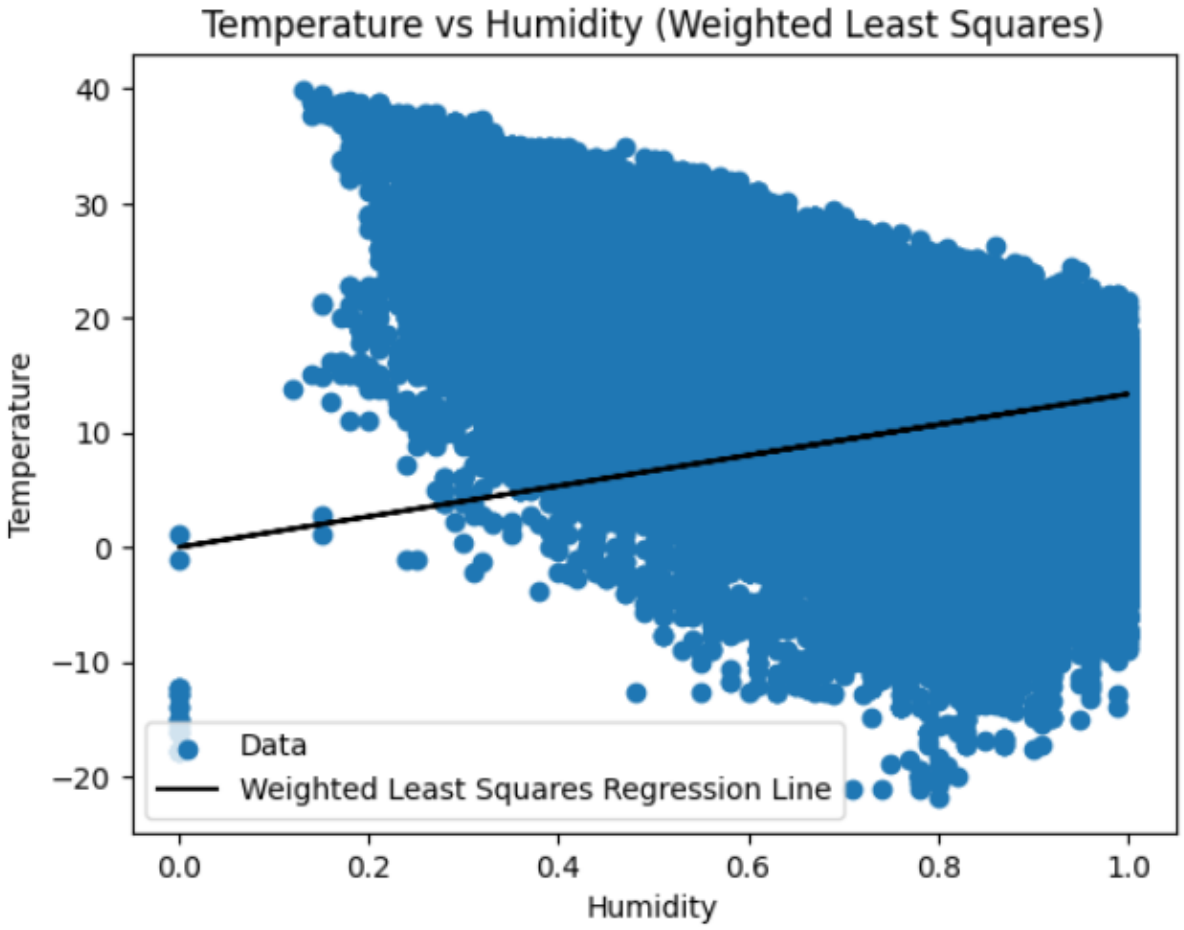
کد به صورت زیر است:

```
X = air_data['Humidity'].values.reshape(-1, 1)
y = air_data['Temperature (C)'].values

weights = np.linspace(0.1, 1, len(y))


W = np.diag(weights)
X_weighted = np.dot(W, X)
y_weighted = np.dot(W, y)
coefficients =
np.linalg.inv(X_weighted.T.dot(X_weighted)).dot(X_weighted.T).dot(y_weighted)


y_pred = X.dot(coefficients)
mse = np.mean((y - y_pred)**2)
print("Mean Squared Error:", mse)

plt.scatter(X, y, label='Data')
plt.plot(X, y_pred, color='k', label='Weighted Least Squares Regression
Line')
plt.xlabel('Humidity')
plt.ylabel('Temperature')
plt.title('Temperature vs Humidity (Weighted Least Squares)')
plt.legend()
plt.show()
```

همان طور که مشاهده می‌شود،تنها تفاوت این الگوریتم با Least Square این است که X و y وزن دار شده اند و یک w در آن ضرب شده است.

نتیجه به صورت زیر است:

```
Mean Squared Error: 134.06284853018997
```



همان طور که مشاهده می‌شود،مقدار خطا در این جا هم حدود ۱۳۴ است که تقریبا با حالت RLS برابر است.خطی که برای تخمین کشیده شده هم تقریبا شبیه حالت RLS است.

به عنوان نتیجه گیری کلی در بین سه الگوریتم LS و RLS و WLS خطای الگوریتم LS برای این دیتاست از بقیه کمتر است.همچنین خط کشیده شده در حالت LS مقداری دقیق تر از دو حالت دیگر می‌باشد.

پایان