دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکدهٔ مهندسی برق

درس یادگیری ماشین

پاسخ مینی پروژه۱

نام و نام خانوادگی	مهدی خدابنده‌لو
شمارهٔ دانشجویی	۴۰۱۰۵۳۱۴
تاریخ	بهار ۱۴۰۳

فهرست مطالب

فهرست تصاویر

فهرست جداول

فهرست برنامه‌ها

Notebook Colab
github

۱ سوال اول

۱.۱ فرآیند آموزش و ارزیابی یک مدل طبقه بند خطی را به صورت دیاگرامی بلوکی نمایش دهید و در مورد اجزای مختلف این دیاگرام بلوکی توضیحاتی بنویسید. تغییر نوع طبقه بندی از حالت دوکلاسه به چندکلاسه در کدام قسمت از این دیاگرام بلوکی تغییراتی ایجاد می کند؟ توضیح دهید.

تصویر ۱ دیاگرام کلی آموزش را برای طبقه‌بندی دو کلاسه نمایش می‌دهد. طبق این دیاگرام نیاز است که پس از بارگیری دیتاست قبل از هر کاری داده‌های null را حذف کنیم. در صورتی که این دیتاست دارای چندین مشخصه باشد و فرض کنیم این مشخصه‌ها به صورت ستونی در کنار هم قرار گرفته باشند کل سطری که در آن یک داده‌ی null قرار دارد حذف می‌شود.

در مرحله‌ی بعد باید داده‌ها به خوبی مخلوط شوند. برخی از دلایلی که داده‌ها را مخلوط می‌کنیم به شرح زیر است:

-جلوگیری از بایاس: در صورتی که مخلوط کردن داده انجام نشود و داده‌ها دارای الگوی خاصی باشند، ممکن است این الگو نیز توسط مدل یاد گرفته شده باشد برای جلوگیری از یادگیری الگوی داده‌ها باید آن ها را مخلوط کنیم.

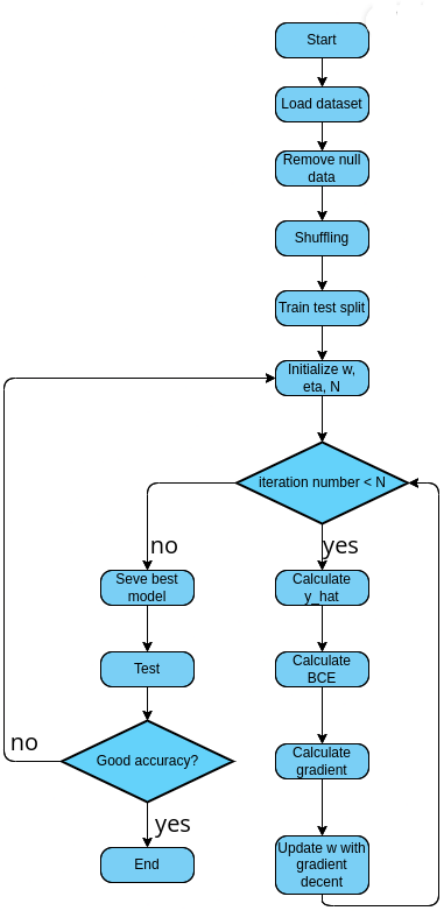
-بهبود تعمیم پذیری: مخلوط کردن داده‌ها کمک می‌کند تا اطمینان حاصل شود که مدل به خوبی به داده‌های دیده نشده تعمیم می‌یابد.

پس از مخلوط کردن داده‌ها نیاز است که داده‌های تست و آموزش را با نسبت معقولی از هم جدا کنیم. اگر تعداد داده‌های آموزش کم باشده مدل به خوبی آموزش نمی‌بیند، همچنین در صورتی که داده‌های تست کم باشند نتیجه‌ی حاصل از تست مدل معتبر نخواهد بود. به طور کلی بهتر است داده‌های تست ۱۵ الی ۲۰ درصد کل داده‌ها باشند.

پس از تقسیم داده‌ها نوبت به مقداردهی اولیه‌ی وزن‌ها، نرخ آموزش و تعداد تکرار های آموزش می‌رسد. مقدار اولیه‌ي وزن ها را می‌توان به صورت تصادفی انتخاب کرد. در صورتی که نرخ یادگیری کم باشد سرعت رسیدن به پاسخ کم خواهد بود از طرفی در صورتی که مقدار این پارامتر زیاد باشد دقت مدل پایین خواهد بود. پارامتر دیگری که باید تنظیم شود تعداد تکرار برای آموزش است. در صورتی که این پارامتر کم باشد مدل به خوبی آموزش نمی‌بیند، از طرفی اگر این پارامتر زیاد باشد هم زمان آموزش طولانی می‌شودد و هم ممکن است overfitting رخ بدهد.

پس از تنظیم مقادیر اولیه پارامترها وارد یک حلقه می‌شویم. در این حلقه هر بار پس از محاسبه‌ی خروجی خطا را محاسبه می‌کنیم و وزن‌ها را با استفاده از گرادیان نزولی آپدیت می‌کنیم. این حلقه N بار تکرار می‌شود. سپس آخرین وزن‌ها ذخیره می‌شوند و با استفاده از این وزن‌ها تست انجام می‌شود. در صورتی که نتایج قابل قبول نبود مقادیر نرخ یادگیری و تعداد تکرار را تغییر می‌دهیم و دوباره مدل را آموزش می‌دهیم.

درصورتی که بخواهیم دسته‌بندی چند کلاسه انجام دهیم به جای این که w های یک خط را تنظیم کنیم نیاز داریم تا w های چندین خط را تنظیم کنیم. -



شکل ۱: دیاگرام آموزش و تست برای طبقه بندی دو کلاسه

۲.۱ با استفاده از**sklearn.datasets**، یک دیتاست با ۱۰۰۰ نمونه، ۴ کلاس و ۳ ویژگی تولید کنید و آن را به صورتی مناسب نمایش دهید. آیا دیتاستی که تولید کردید چالش برانگیز است؟ چرا؟ به چه طریقی می توانید دیتاست تولیدشده خود را چالش برانگیزتر و سخت تر کنید؟

همان طور که در کد ۱ قابل مشاهده است با استفاده از دستور make_classification نقطه را تولید کرده‌ایم. و سپس آن ها را در یک نمودار سه بعدی نمایش داده‌ایم. شکل۱ این نقاط را نمایش می‌دهد.

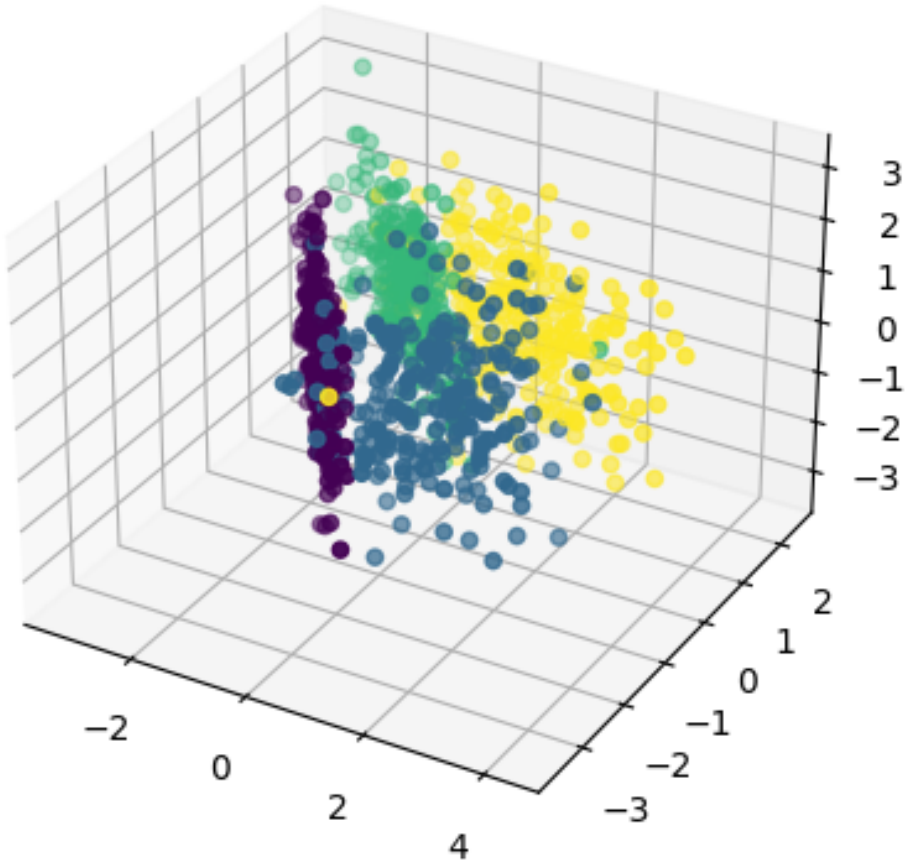
```
X,y = make_classification(n_samples=1000, n_features=3, n_classes=4,
        n_redundant=0, random_state=14,
        n_clusters_per_class=1, class_sep=1)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
```

```
ax.scatter(X[:,0], X[:,1], X[:,2], c=y)
```

Code 1: Generate data (Python)

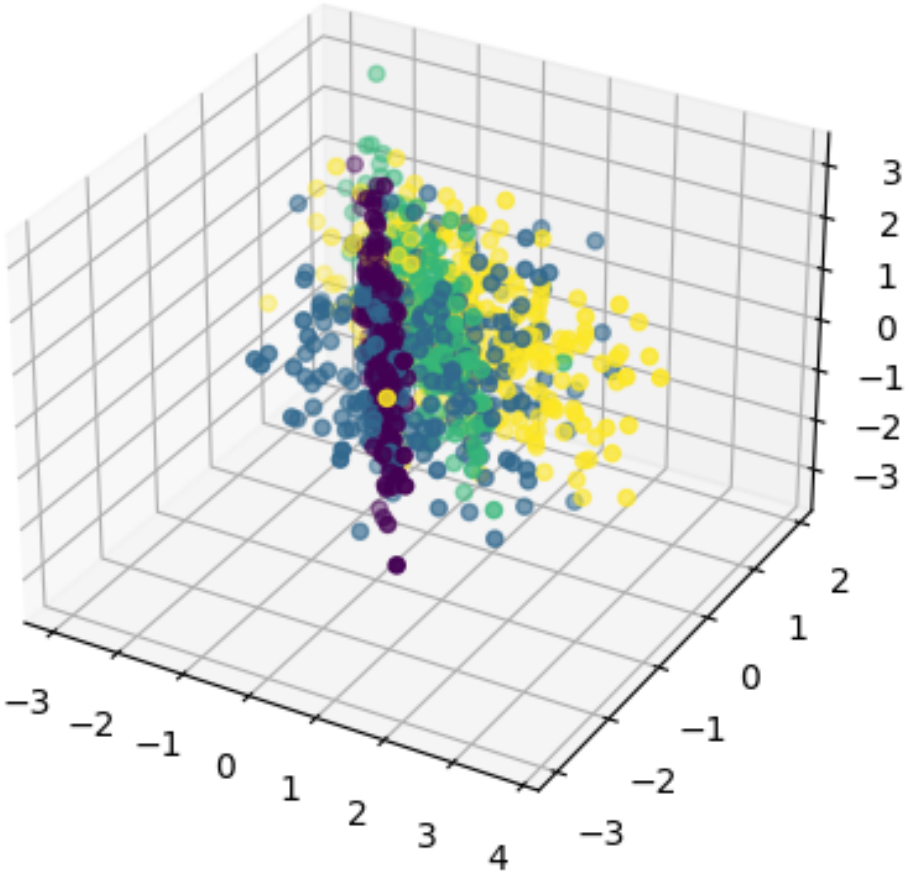
همان طور که از شکل ۲ پیداست دیتاستی که تولید شده زیاد چالش برانگیز نمی‌باشد و می‌توان داده ها را از هم تفکیک کرد. برای این که داده های تولید شده را چالش بر انگیز تر کنیم می‌توانیم کارهای مختلفی انجام دهیم.



شکل ۲: داده های تولید شده با class_sep=۱

یکی از این کارها این است که فراپارامتر class_sep را کاهش دهیم. شکل ۳ داده‌ها را در حالتی که این پارامتر 0.4 است نمایش می‌دهد. همان طور که مشخص است داده‌های کلاس‌های مختلف به هم نزدیک تر شه اند وتفکیک آن‌ها مشکل تر شده است.

راه دیگری مه برای سخت تر کردن داده‌ها وجود دارد این است که با جای این که داده‌ها را به صورت خوشه‌ای پراکنده کنیم آن ها را به شکل‌های دیگر مثل دایروی پراکنده کنیم.



شکل ۳: داده های تولید شده با class_sep=۰.۵

۳.۱ با استفاده از حداقل دو طبقه‌بند خطی آماده‌ی پایتون و در نظر گرفتن فراپارامترهای مناسب چهار کلاس موجود در دیتاست قبلی را از هم تفکیک کنید. ضمن روند توضیح فراپارامترها نتیجه دقت آموزش و ارزیابی را نشان دهید. برای بهبود نتایج از چه روش‌هایی استفاده کردید؟

جدا کردن داده‌های آموزش و تست:

در ۲ داده‌های آموزش و تست را به نسبت ۸۰ به ۲۰ تقسیم کرده‌ایم.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=14)
```

Code 2: Train and test split (Python)

Logistic_regression

در کد ۳ ابتدا تعداد تکرار برای آموزش را ۵۰ در نظر گرفته‌ایم و پس از آموزش، با استفاده از داده‌های تست دقت مدل را اندازه گیری کرده‌ایم. با مقادیری که برای هر یک از وارامتر ها در نظر گرفته شده دقت برابر با ۸۹ درصد است.

```
logistic_regression_model = LogisticRegression(max_iter=50, random_state=14)
logistic_regression_model.fit(X=X_train, y=y_train)

y_hat = logistic_regression_model.predict(X_test)
y_hat_prob = logistic_regression_model.predict_proba(X_test)
accuracy = logistic_regression_model.score(X_test, y_test)
print('accuracy :', accuracy*100, '%')
```

Code 3: Train and test for LogisticRegression(Python)

SGDClassifier

در کد ۴ ابتدا تعداد تکرار برای آموزش را ۵۰ در نظر گرفته‌ایم و پس از آموزش، با استفاده از داده‌های تست دقت مدل را اندازه گیری کرده‌ایم. با مقادیری که برای هر یک از وارامتر ها در نظر گرفته شده دقت برابر با ۵.۸۷ درصد است.

```
SGDClassifier_model = SGDClassifier(loss='log_loss', max_iter=50, random_state
    =14)
SGDClassifier_model.fit(X_train, y_train)
y_hat = SGDClassifier_model.predict(X_test)
y_hat_prob = SGDClassifier_model.predict_proba(X_test)
accuracy = SGDClassifier_model.score(X_test, y_test)
print('accuracy :', accuracy*100, '%')
```

Code 4: Train and test for SGDClassifier(Python)

برای بهبود نتایج باید تعداد تکرار برای آموزش را مناسب در نظر بگیریم. در صورتی که این پارامتر زیاد در نظر گرفته شده باشد باعث overfitting می‌شود، این پدیده باعث می‌شود که مدل روی داده‌های آموزش عملکرد خوبی داشته باشد ولی وقتی داده‌هایی غیر از داده‌های آموزش به مدل داده شود عملکرد آن خراب می‌شود. این پدیده مانند این است که یک دانش آموز به جای یاد گرفتن درس آن را حفظ کرده باشد. مسئله‌ی دیگری که در بهبود نتایج موثر است تعداد کافی داده‌های آموزش است، در صورتی مه تعداد داده‌های آموزش کم باشد مدل به خوبی آموزش نمی‌بیند.

۴.۱ مرز و نواحی تصمیم گیری برآمده از مدل آموزش دیدهٔ خود را به همراه نمونه ها در یک نمودار نشان دهید. اگر می توانیدنمونه هایی که اشتباه طبقه بندی شده اند را با شکل و رنگ متفاوت نمایش دهید

کد ۵ مربوط به نمایش نواحی تصمیم گیری در Logistic Regression است.شکل ۴ خروجی این کد است. کد ۶ نیز مربوط به نمایش نواحی تصمیم گیری ونقاطی که اشتباه دسته‌بندی شده‌اند است. برای نمایش بهتر تنها نقاطی که اشتباه دسته‌بندی شده اند نمایش داده شده اند و نقاطی که به درستی دسته بندی شده اند نمایش داده نشده اند. کد ۵ خروجی این کد است.

```
x1_min, x2_min, x3_min = np.min(X, axis=0)
x1_max, x2_max, x3_max = np.max(X, axis=0)

n_of_points = 50
x1r = np.linspace(x1_min, x1_max, n_of_points)
x2r = np.linspace(x2_min, x2_max, n_of_points)
x3r = np.linspace(x3_min, x3_max, n_of_points)

x1m, x2m, x3m = np.meshgrid(x1r.flatten(), x2r.flatten(), x3r.flatten())

xm = np.stack((x1m.flatten(), x2m.flatten(), x3m.flatten()), axis=1)
ym = logistic_regression_model.decision_function(xm)
q = xm.shape
c = np.argmax(ym, axis=1)

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.scatter(xm[:,0], xm[:,1], xm[:,2], c=c)
```

Code 5: Decision region for LogisticRegression(Python)

```
x1_min, x2_min, x3_min = np.min(X, axis=0)
x1_max, x2_max, x3_max = np.max(X, axis=0)

n_of_points = 50
x1r = np.linspace(x1_min, x1_max, n_of_points)
x2r = np.linspace(x2_min, x2_max, n_of_points)
x3r = np.linspace(x3_min, x3_max, n_of_points)

```

```
x1m, x2m, x3m = np.meshgrid(x1r.flatten(), x2r.flatten(), x3r.flatten())

xm = np.stack((x1m.flatten(), x2m.flatten(), x3m.flatten()), axis=1)
ym = logistic_regression_model.decision_function(xm)
q = xm.shape
c = np.argmax(ym, axis=1)

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.scatter(xm[:,0], xm[:,1], xm[:,2], c=c, alpha=0.01)

non_eq_idx = np.where(y_test!=y_hat)[0]
ax.scatter(X[non_eq_idx,0], X[non_eq_idx,1], X[non_eq_idx,2], color='red')
```

Code 6: Decision region with wrong points for LogisticRegression(Python)

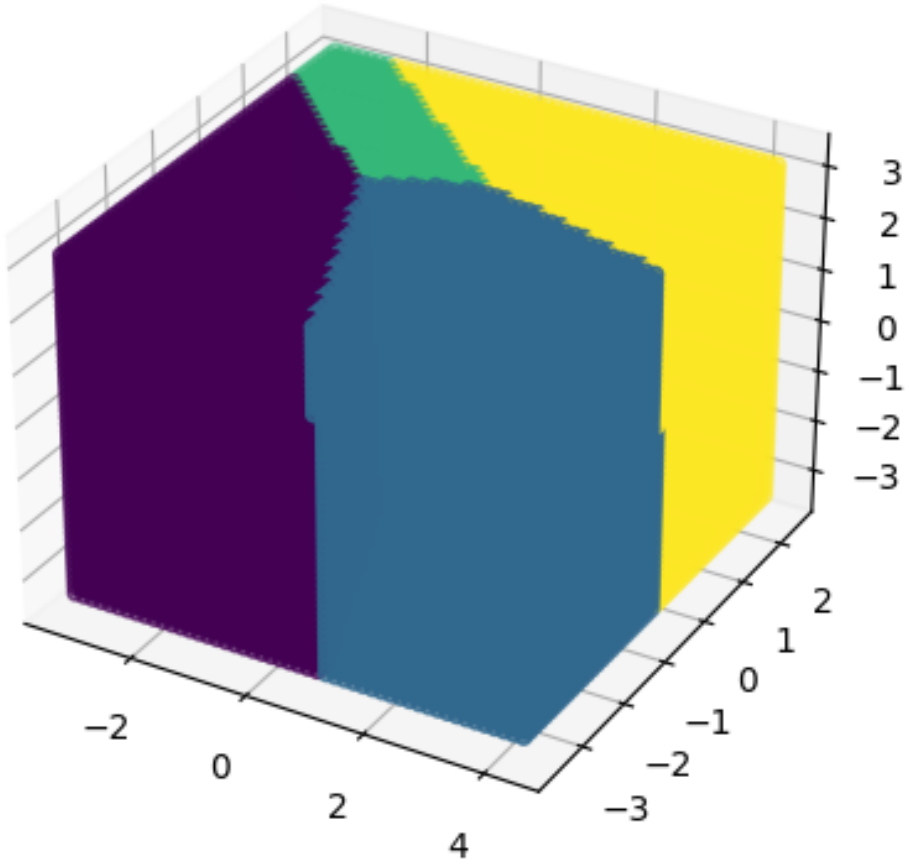
شکل ۶ و شکل ۷ مربوط به نواحی تصمیم گیری و نقاط اشتباه در SGDClassifier هستند.

۵.۱ فرآیندی مشابه قسمت ۲ را با تعداد کلاس و ویژگی دلخواه؛ اما با استفاده از ابزار drawdata تکرار کنید.قسمت های ۳ و ۴ را برای این داده های جدید تکرار و نتایج را به صورتی مناسب نشان دهید

تفاوتی که این قسمت با قسمت‌های قبل دارد تنها در تولید و آماده سازی داده و نمایش نواحی تصمیم گیری است. شکل ۸ داده‌های تولید شده را نمایش می‌دهد.

در کد ۷ ابتدا به هر یک از برچسب‌ها یک عدد نسبت داده‌ایم(a:۱ b:۲ c:۳ d:۴) سپس آن ها را مخلوط کرده‌ایم و در مرحله‌ی آخر داده‌ها را به نسبت ۸۰ به ۲۰ به داده‌های آموزش و تست تقسیم کرده‌ایم.

```
data = widget.data_as_pandas
X = data[['x','y']].values
y = data[['label']].values.flatten()
mapping = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
y = np.array([mapping[element] for element in y])
np.random.seed(14)
perm = np.random.permutation(len(y))
X = X[perm,:]
y = y[perm]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```



شکل ۴: نواحی تصمیم گیری برای LogisticRegression

```
    random_state=14)
```

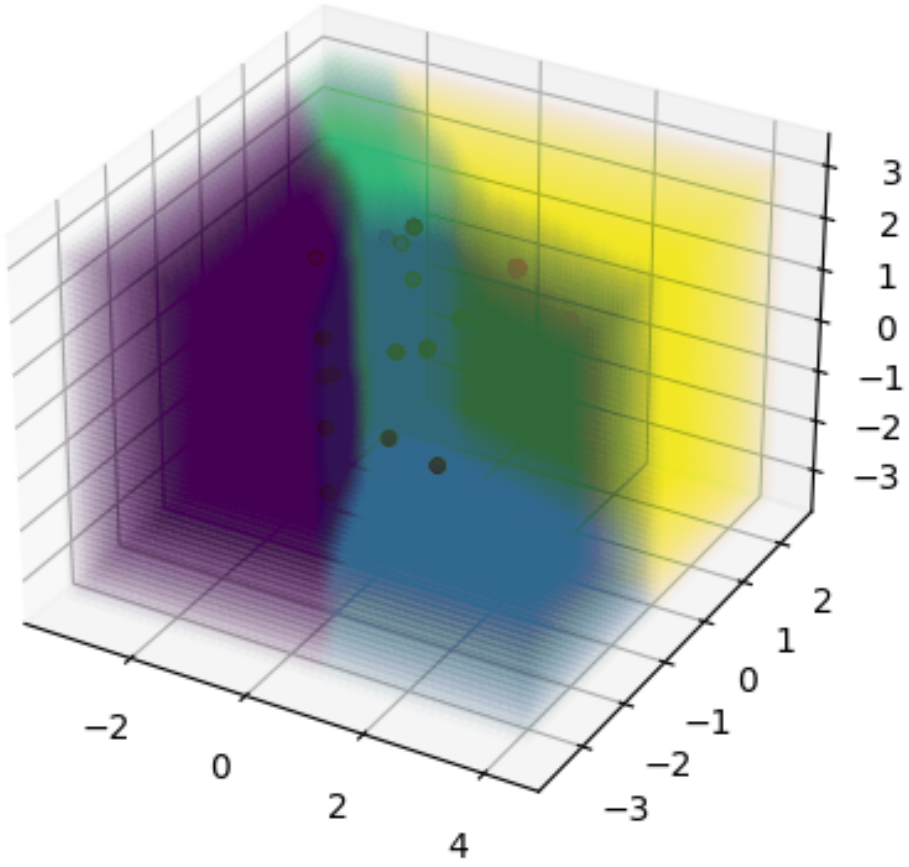
Code 7: Decision region with wrong points for LogisticRegression(Python)

پس از آماده سازی داده‌ها نوبت به آموزش و تست می‌رسد. برای Regression Logistic ابتدا تعداد تکرار آموزش مانند قسمت قبل ۵۰ در نظر گرفته شد ولی دقت ۳۳.۷۷ درصد شد. علت این امر این است که داده‌هایی که با روش datadraw تولید گردید توزیع چالش برانگیزتری دارند چون داده‌های کلاس‌های مختلف به هم نزدیک تر هستند. با تغییر افزایش تعداد تکرار به ۳۰۰ دقت به ۲۲.۸۵ رسید.

کد ۸ مربوط به نمایش نواحی تصمیم گیری است و شکل ۹ خروجی این کد است. همان طور که قابل مشاهده است نقاطی که اشتباه دسته‌بندی شده اند به رنگ قرمز در آمده اند.

```
x1_min, x2_min = np.min(X, axis=0)
x1_max, x2_max = np.max(X, axis=0)

```



شکل ۵: نواحی تصمیم گیری به همراه نقاط اشتباه برای LogisticReggression

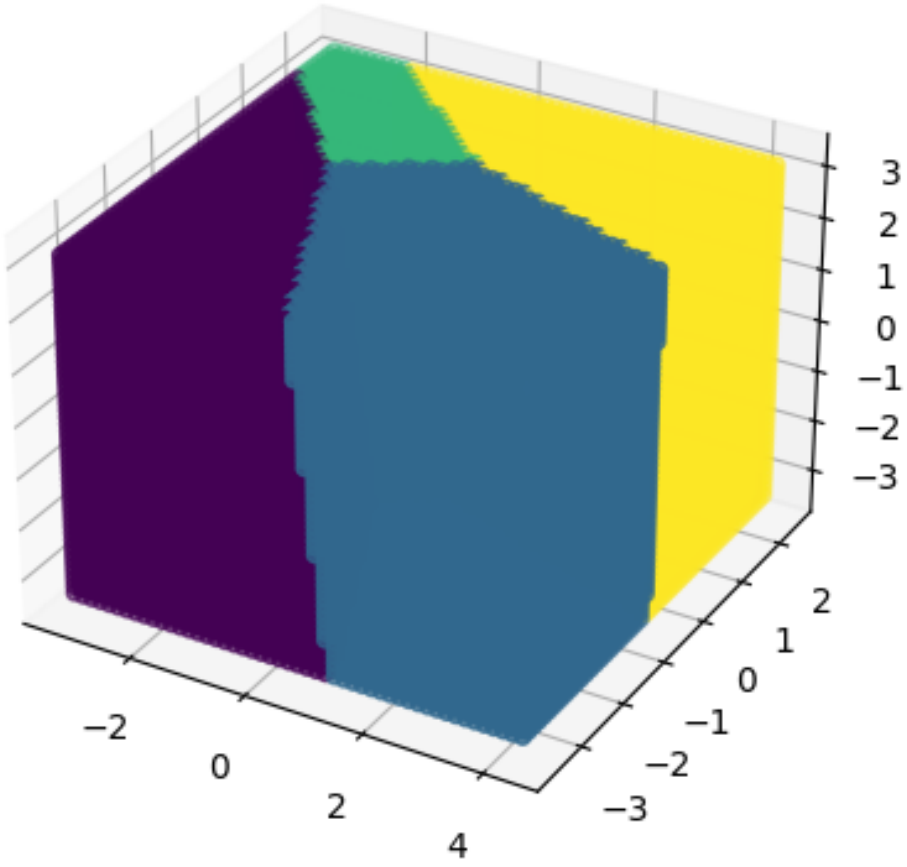
```
n_of_points = 500
x1r = np.linspace(x1_min, x1_max, n_of_points)
x2r = np.linspace(x2_min, x2_max, n_of_points)

x1m, x2m = np.meshgrid(x1r, x2r)

xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
ym = logistic_regression_model.decision_function(xm)

ym = np.argmax(ym, axis=1)


```



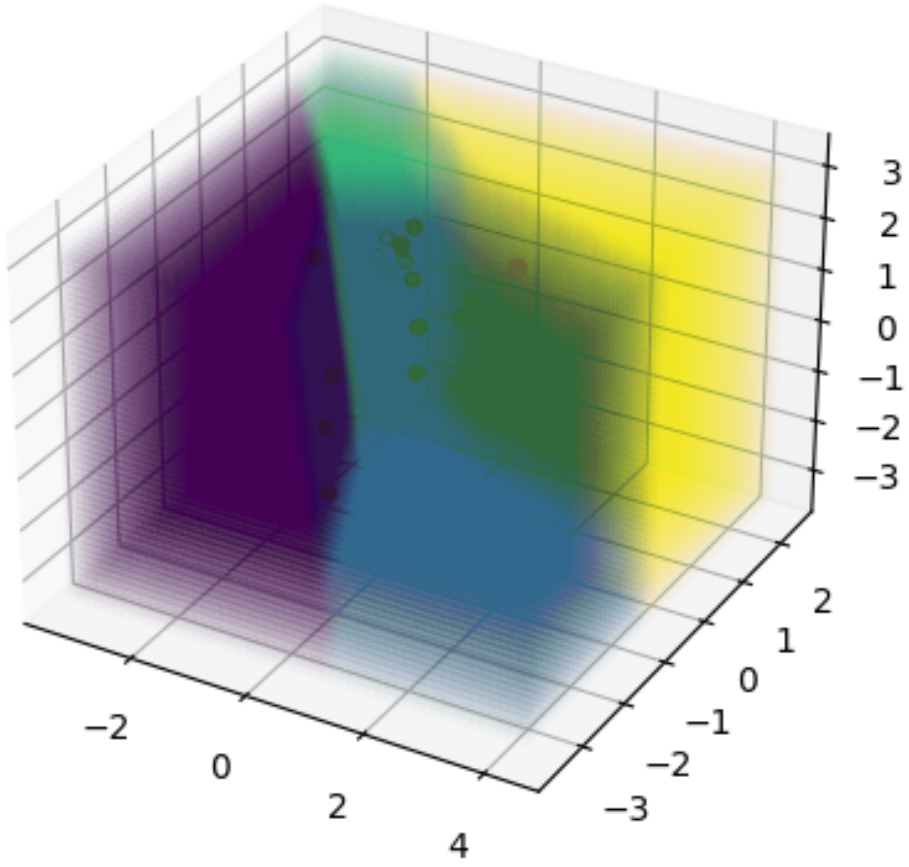
شکل ۶: نواحی تصمیم گیری برای SGDClassifier

```
x1d = np.concatenate((xm[:,0], X_test[:,0]), axis=0)
x2d = np.concatenate((xm[:,1], X_test[:,1]), axis=0)
yd = np.concatenate((ym, y_test+4), axis=0)
plt.scatter(x1d, x2d, c=yd)
non_eq_idx = np.where(y_test!=y_hat)[0]

plt.scatter(X_test[non_eq_idx,0], X_test[non_eq_idx,1], color='red')
```

Code 8: Decision region with wrong points for LogisticRegression(Python)

برای SGDClassifier ابتدا تعداد تکرار برای آموزش برابر با ۵۰ در نظر گرفته شد که در این حالت دقت تنها ۶.۲۶ درصد بود. با افزایش تعداد تکرار به ۱۰۰۰ دقت تنها به ۹.۴۲ درصد رسید که دقت بسیار کمی است. شکل ۱۰ نواحی تصمیم گیری را برای این مدل نمایش می‌دهد. همان طور که مشخص است تعداد زیادی از نقاط اشتباه دسته‌بندی شده اند.



شکل ۷: نواحی تصمیم گیری به همراه نقاط اشتباه برای SGDClassifier

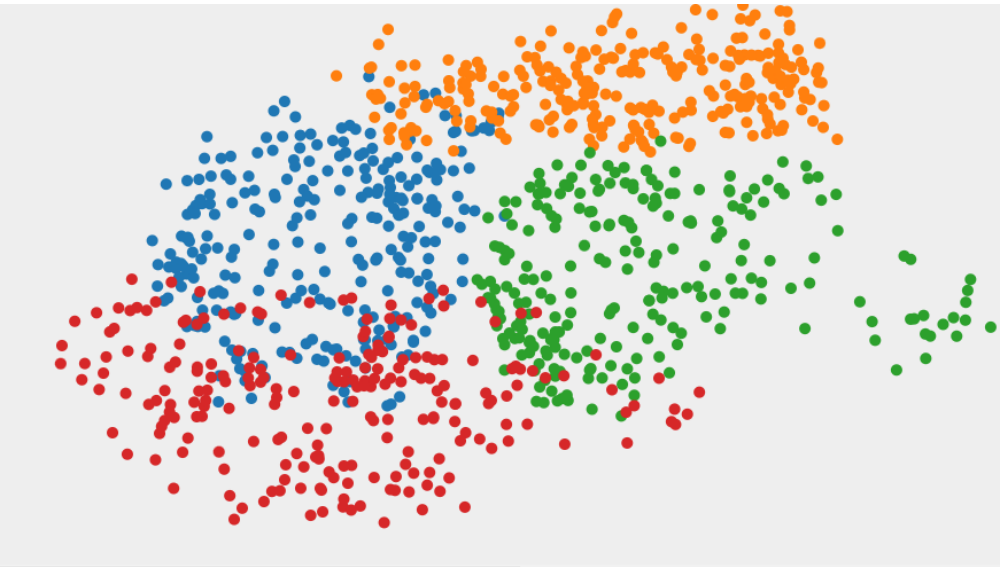
۲ سوال دوم

۱.۲ با مراجعه به صفحه‌ی دیتاست Bearing CWRU با یک دیتاست مربوط به حوزه‌ی تشخیص عیب آشنا شوید. با جستجوی آن در اینترنت و مقالاتُ توضیحاتی از اهداف، ویژگی‌ها و حالت های مختلف این دیتاست توضیحاتی ارایه کند.

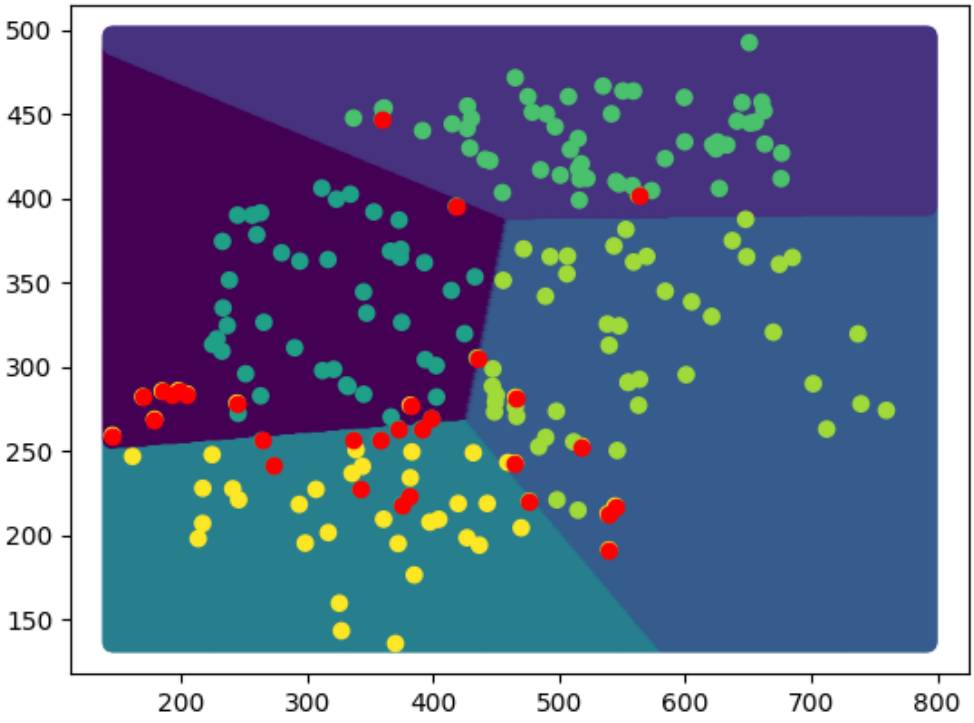
برای تشخیص عیب از دو ویژگی drive end accelerometer data (که در کد x1 نامیده شده) و fan end accelerometer data (که در کد x2 نامیده شده) استفاده شده است.

۲.۲ برای تشکیل دیتاست مراحل زیر را انجام دهید:

الف: از هر کلاس M نمونه به طول N جدا کنید(M حد اقل ۱۰۰ و N حداقل ۲۰۰ باشد).یک ماتریس از داده های هر دو کلاس به همراه برچسب آن‌ها تشکیل دهید. می‌توانید پنجره‌ای به طول N در نظر بگیرید و در نهایت یک ماتریس MxN از داده‌های هر کلاس استخراج



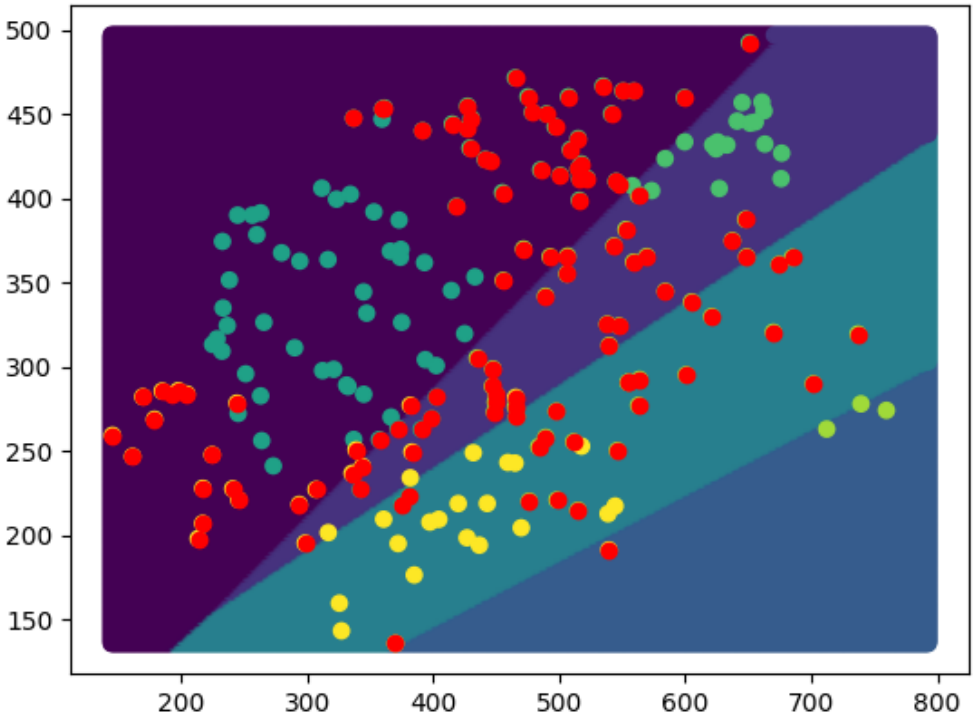
شکل ۸: داده‌های تولید شده با drawdata



شکل ۹: نواحی تصمیم گیری برای Regression Logistic

کنید.

ما در این جا M را برابر با ۵۰۰ و N را ۲۰۰ در نظر گرفته‌ایم. همان طور که از کد ۹ مشخص است هر یک از ویژگی‌های کلاس‌های معیوب و سالم را با استفاده از دستور reshape به صورت مربعی درآورده ایم.



شکل ۱۰: نواحی تصمیم گیری برای SGDCLassifier

```
M = 500
N = 200
x1_normal = np.reshape(bearing_normal_dataset['X098_DE_time'][0:M*N], (M,N))
x2_normal = np.reshape(bearing_normal_dataset['X098_FE_time'][0:M*N], (M,N))
x1_fault = np.reshape(bearing_fault_dataset['X108_DE_time'][0:M*N], (M,N))
x2_fault = np.reshape(bearing_fault_dataset['X108_FE_time'][0:M*N], (M,N))
```

Code 9: Convert data to a MxN window(Python)

ب: در مورد اهمیت استخراج ویژگی در یادگیری ماشین توضیحاتی بنویسید.سپس، با استفاده از حداقل هشت عدد از روش های ذکر شده در جدول۱، ویژگی‌های دیتاست قسمت «۲-آ» را استخراج کنید و یک دیتاست جدید تشکیل دهید.

کد ۱۰ تابعی که به منظور استخراج ویژگی نوشته شده را نشان می‌دهد. ورودی این تابع ویژگی‌های دیتاست خام می‌باشد و خروجی آن یک dictionary است که شامل ویژگی های استخراج شده از ویژگی های خام و خود ویژگی‌های خام و خود ویژگی‌های خام است. این تابع تمامی ویژگی‌های ذکر شده در جدول را محاسبه می‌کند. نکته‌ی مهمی که در این تابع وجود دارد این است که به منظور جلوگیری از تقسیم شدن بر صفر، زمانی که مقسوم علیه صفر شود، به جای صفر مقدار اپسیلون قرار می‌گیرد.

```
def feature_extract(x1,x2,ep):
```

```python
    x1_n = np.expand_dims(x1, axis=2)
    x2_n = np.expand_dims(x2, axis=2)
    xc = np.concatenate((x1_n,x2_n), axis=2)
    xc_mean = np.mean(xc, axis=2)
    m = np.expand_dims(xc_mean, axis=2)
    m = np.concatenate((m,m), axis=2)

    feature = {'x1': x1,'x2': x2}
    feature['std']  = np.std(xc, axis=2)
    feature['p']    = np.amax(np.abs(xc), axis=2)
    temp = np.std(xc, axis=2)**3
    temp[temp == 0] = ep
    feature['ske']  = np.mean((xc-m)**3, axis=2) / temp
    temp = np.std(xc, axis=2)**4
    temp[temp == 0] = ep
    feature['kur']  = np.mean((xc-m)**4, axis=2) / temp
    feature['rms']  = np.sqrt(np.mean(xc**2, axis=2))
    temp = np.sqrt(np.mean(xc**2, axis=2))
    temp[temp == 0] = ep
    feature['cf']   = np.amax(np.abs(xc), axis=2) / temp
    feature['smr']  = np.mean(np.sqrt(np.abs(xc)), axis=2) ** 2
    temp = np.mean(np.sqrt(np.abs(xc)), axis=2) ** 2
    temp[temp == 0] = ep
    feature['clf']  = np.amax(np.abs(xc), axis=2) / temp
    temp = np.mean(np.abs(xc), axis=2)
    temp[temp == 0] = ep
    feature['sf']   = np.sqrt(np.mean(xc**2, axis=2)) / temp
    temp = np.mean(np.abs(xc), axis=2)
    temp[temp == 0] = ep
    feature['if1']  = np.amax(np.abs(xc), axis=2) / temp
    feature['if2']  = np.max(np.abs(xc), axis=2) / temp
    return feature
```

Code 10: Function for feature extraction(Python)

همان طور که از کد ۱۱ مشخص است مقدار اپسیلون را 10^{-10} در نظر گرفته‌ایم. پس از این مرحله داده‌های کلاس normal و fault

را با هم ترکیب می‌کنیم. البته نیاز است که قبل از استفاده از داده‌ها آن‌ها را به خوبی بر بزنیم.

```python
x_normal = feature_extract(x1_normal, x2_normal,1e-10)
x_fault = feature_extract(x1_fault, x2_fault,1e-10)

X = {'x1': np.concatenate((x_normal['x1'], x_fault['x1']), axis=0),
     'x2': np.concatenate((x_normal['x2'], x_fault['x2']), axis=0),
     'std': np.concatenate((x_normal['std'], x_fault['std']), axis=0),
     'p': np.concatenate((x_normal['p'], x_fault['p']), axis=0),
     'ske': np.concatenate((x_normal['ske'], x_fault['ske']), axis=0),
     'kur': np.concatenate((x_normal['kur'], x_fault['kur']), axis=0),
     'rms': np.concatenate((x_normal['rms'], x_fault['rms']), axis=0),
     'cf': np.concatenate((x_normal['cf'], x_fault['cf']), axis=0),
     'smr': np.concatenate((x_normal['smr'], x_fault['smr']), axis=0),
     'clf': np.concatenate((x_normal['clf'], x_fault['clf']), axis=0),
     'sf': np.concatenate((x_normal['sf'], x_fault['sf']), axis=0),
     'if1': np.concatenate((x_normal['if1'], x_fault['if1']), axis=0),
     'if2': np.concatenate((x_normal['if2'], x_fault['if2']), axis=0)}
y0 = np.zeros((M,N))#normal
y1 = np.ones((M,N))#fault
y = np.concatenate((y0,y1), axis=0)
```

Code 11: (Python)

ج: ضمن توضیح اهمیت فرآیند بر زدن، داده‌ها را در صورت امکان مخلوط کرده و با نسبت تقسیم دلخواه و معقول به دو بخش آموزش و ارزیابی تقسیم کنید.

در صورتی که داده‌های کلاس‌های مختلف به خوبی مخلوط نشوند فرایند یادگیری به درستی انجام نمی‌شود. برخی از دلایلی که داده‌ها را مخلوط می‌کنیم به شرح زیر است:

-جلوگیری از بایاس: در صورتی که مخلوط کردن داده انجام نشود و داده‌ها دارای الگوی خاصی باشند، ممکن است این الگو نیز توسط مدل یاد گرفته شده باشد برای جلوگیری از یادگیری الگوی داده‌ها باید آن ها را مخلوط کنیم.

-بهبود تعمیم پذیری: مخلوط کردن داده‌ها کمک می‌کند تا اطمینان حاصل شود که مدل به خوبی به داده‌های دیده نشده تعمیم می‌یابد.در کد ۱۲ ابتدا داده‌ها را به شکل یکسان مخلوط کرده‌ایم. پس از مخلوط کردن داده‌ها را به نسبت ۸۰ به ۲۰ به داده‌های آموزش و تست تقسیم کرده‌ایم.

```python
data_shape = y.shape
for key,value in X.items():
```

```
  X[key] = X[key].flatten()
y = y.flatten()
np.random.seed(14)
perm = np.random.permutation(len(y))
for key,value in X.items():
  X[key] = X[key][perm]
y = y[perm]
for key,value in X.items():
  X[key] = X[key].reshape(data_shape)
y = y.reshape(data_shape)
#train and test split
train_size = 0.8
train_a = round(train_size*M*2)

X_train = {}
X_test = {}
for key, value in X.items():
  X_train[key] = X[key][0:train_a,:]
  X_test[key] = X[key][train_a:,:]
y_train = y[0:train_a,:]
y_test = y[train_a:,:]
```

Code 12: Data shuffling and train and test split(Python)

د: حداقل دو روش برای نرمال سازی داده ها را با ذکر اهمیت این فرآیند توضیح دهید و با استفاده از یکی از این روش ها، داده ها را نرمال کنید. آیا از اطلاعات بخش ارزیابی در فرآیند نرمال سازی استفاده کردید؟ چرا؟

از آن جایی که رنج تغییرات ویژگی‌های مختلف متفاوت است ممکن است برخی از آنها که دامنه‌ی بزرگتری دارند اثر ویژگی‌هایی که دامنه‌ی کم تری دارند را خنثی کنند، از این رو بهتر است برای جلوگیری از این پدیده داده‌ها را نرمالیزه کنیم. یکی از روش‌های نرمالیزه کردن scaling to a range است که فرمول آن به شکل زیر است:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \times (max - min) + min \quad (۱)$$

روش دیگری که می‌توان نام برد روش log scaling است که در این روش کافی است از داده‌ها فقط لگاریتم بگیریم. روش دیگر zscore نام دارد که رابطه‌ی آن به شرح زیر است:

$$x_{scaled} = \frac{x - \mu}{\sigma} \quad (۲)$$

در رابطه‌ی بالا μ میانگین و σ انحراف معیار است.

کد ۱۳ مربوط به نرمالیزه کردن داده‌های تست و آموزش به روش zscore است. نکته‌ای که وجود دارد این است از اطلاعات بخش تست برای نرمالیزه کردن استفاده نمی‌شود، زیرا داده‌های تست حکم داده‌های دنیای واقی را دارند که قرار است در آینده به مدل داده شوند. نکته‌ای که وجود دارد این است که برای نرمالیزه کردن داده‌های تست هم از میانگین و انحراف معیار داده‌های آموزش استفاده می‌شود.

```python
for key,value in X.items():
  x_mean = np.mean(X_train[key])
  x_std = np.std(X_train[key])
  X_train[key] = (X_train[key]- x_mean)/x_std
  X_test[key] = (X_test[key]- x_mean)/x_std
```

Code 13: Data normalization(Python)

۳.۲ بدون استفاده از کتابخانه های آمادهٔ پایتون، مدل طبقه بند، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کدنویسی کنیدتا دو کلاس موجود در دیتاست به خوبی از یکدیگر تفکیک شوند. نمودار تابع اتلاف را رسم کنید و نتیجهٔ ارزیابی رویداده های تست را با حداقل ۲شاخصه محاسبه کنید. نمودار تابع اتلاف را تحلیل کنید. آیا می توان از روی نمودارتابع اتلاف و قبل از مرحلهٔ ارزیابی با قطعیت در مورد عمل کرد مدل نظر داد؟ چرا و اگر نمی توان، راه حل چیست؟

همان طور که در کد ۱۴ قابل مشاهده است به ترتیب توابع سیگموید، Logistic_regression، LOSS گرادیان، گرادیان نزولی و دقت پیاده‌سازی شده‌اند.

```python
def sigmoid(x):
  return 1 / (1 + np.exp(-x))

def logestic_regression(x,w):
  u = 0
  for key,value in x.items():
    u += x[key] * w[key]
  h_hat = sigmoid(u)
  return h_hat

def bce(y, y_hat):
  ep = 1e-10
  loss = -(np.mean(y*np.log(y_hat + ep) + (1-y)*np.log(1-y_hat + ep)))
```

```
  return loss

def gradient(x, y, y_hat):
  grd = {}
  for key, value in x.items():
    grd[key] = np.sum(np.multiply(x[key],(y_hat-y)))
  return grd

def gradient_decent(w, eta, grads):
  for key, value in w.items():
    w[key] -= eta * grads[key]
  return w

def accuracy(y, y_hat):
  acc = np.sum(y == np.round(y_hat)) / y.size
  return acc
```

Code 14: Sigmoid, Loss, Logistic Regression, Gradient, Gradient Decent(Python)

در کد ۱۵ مقادیر اولیه‌ی وزن‌ها، تعداد تکرار برای آموزش و گام آموزش تعیین شده است.

```
X_train['bias'] = np.ones(X_train['x1'].shape)
X_test['bias'] = np.ones(X_test['x1'].shape)
np.random.seed(14)
w_initial = np.random.rand(len(X)+1,1)
w = {'bias': w_initial[0,0], 'x1': w_initial[1,0], 'x2': w_initial[2,0],
     'std': w_initial[3,0], 'p': w_initial[4,0], 'ske': w_initial[5,0],
     'kur': w_initial[6,0], 'rms': w_initial[7,0], 'cf': w_initial[8,0],
     'smr': w_initial[9,0], 'clf': w_initial[10,0], 'sf': w_initial[11,0],
     'if1': w_initial[12,0], 'if2': w_initial[13,0]}
initial_eta = 0.00001
eta = initial_eta
n_epochs = 100
```

Code 15: Initial values(Python)

پس از مراحل بالا نوبت به آموزش مدل می‌رسد. کد ۱۶ مربوط به آموزش مدل است که در هر اجرا وزن‌ها در آن آپدیت می‌شوند.

```python
error_hist = []
for epoch in range(n_epochs):
    # predictions
    y_hat = logestic_regression(X_train, w)
    # loss
    e = bce(y_train, y_hat)
    error_hist.append(e)

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

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

    print(f'Epoch={epoch}, \t E={e}')
```

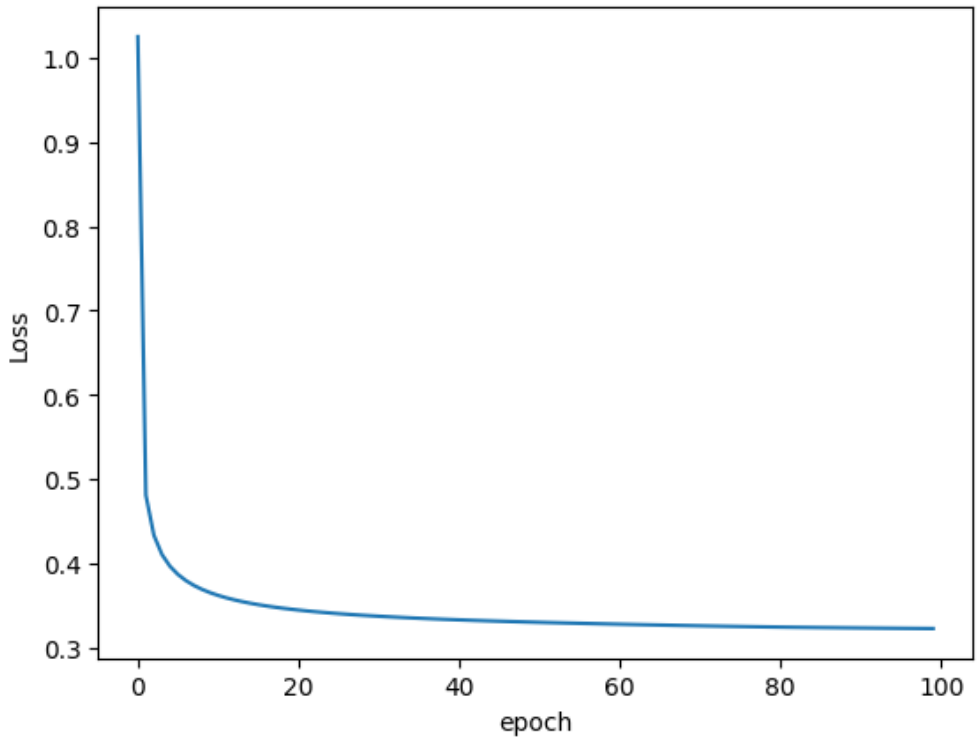
Code 16: Train model(Python)

شکل ۱۱ خروجی تابع Loss را در هر اجرا نمایش می‌دهد.

شکل ۱۲ قدر مطلق وزن‌ها را نمایش می‌دهد. دقت مدل ۸۷ درصد و شاخصه‌ی MSE حدودا برابر با 0.1 می‌باشد. مسیله‌ای که وجود دارد این است که به دلایلی نمی‌توان قبل از ارزیابی سیستم با داده‌های تست در مورد عملکرد آن نظر داد. یکی از این دلایل این است که ممکن است سیستم overfit شده باشد، این یعنی مدل داده‌های آموزش را بیش از حد یادگرفته و به همین دلیل روی داده‌های دیکر عملکرد خوبی نخواهد داشت، برای جلوگیری از این مشکل باید تعداد تکرار برای یادگیری زیاد از حد نباشد. یکی دیگر از دلایلی که وجود دارد این است ممکن است که تعداد داده‌های آموزش کم باشد و یا داده‌ها به خوبی مخلوط نشده باشند یا به بیان دیگر این داده‌ها به خوبی نماینده‌ی کل داده‌ها نباشند، برای رفع این مشکل باید داده‌های آموزش به گونه‌ای باشند که داده‌های انتخاب شده به خوبی نماینده‌ی کل داده‌ها باشند.

۴.۲ فرآیند آموزش و ارزیابی را با استفاده از یک طبقه بند خطی آماده‌ی پایتون انجام داده و نتایج را مقایسه کنید. در حالت استفاده از دستورات آماده‌ی سایکیتلرن، آیا راهی برای نمایش نمودار تابع اتلاف وجود دارد؟ پیاده سازی کنید.

کد ۱۷ مربوط به آماده سازی داده‌ها می‌باشد. از آنجایی که برای آموزش بهتر مدل لازم است تعداد داده‌های هر دو کلاس حدودا برابر باشد از هر دو کلاس ۱۰۰۰۰۰ داده انتخاب کرده‌ایم. سپس این داده‌ها را در کنار هم قرار داده‌ایم و آنها را مخلوط کرده‌ایم. پس از مخلوط مردن داده‌ها آن ها را با نسبت ۸۰ به ۲۰ به داده‌های آموزش و تست تقسیم کرده‌ایم.

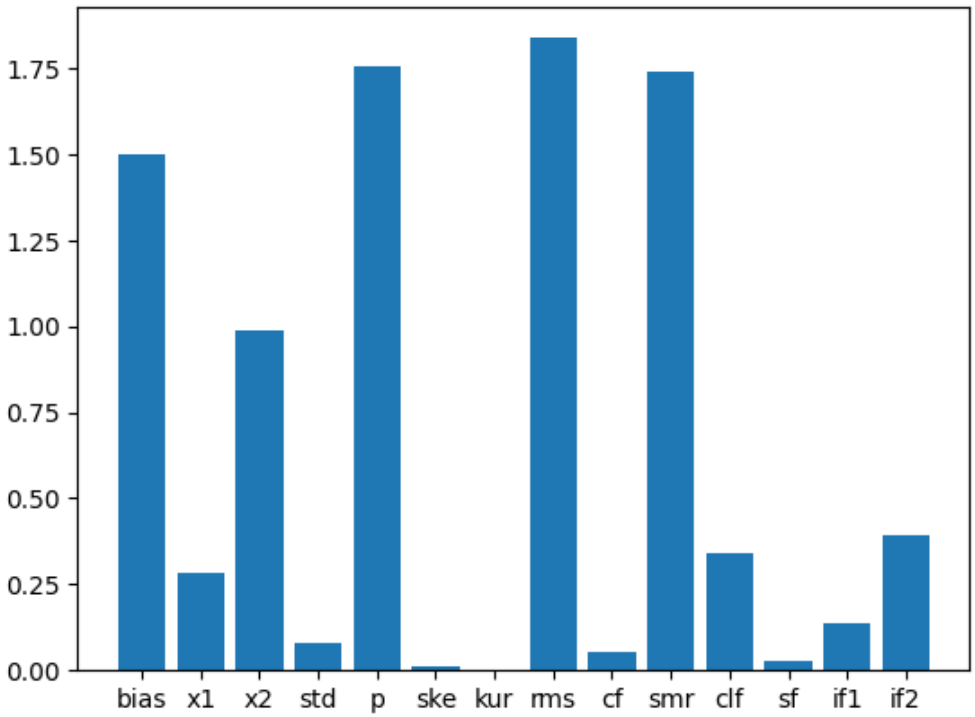


شکل ۱۱: خروجی تابع Loss در هر مرحله

```
x1_normal = bearing_normal_dataset['X099_DE_time'][0:10000,:]
x2_normal = bearing_normal_dataset['X099_FE_time'][0:10000,:]
x1_fault = bearing_fault_dataset['X108_DE_time'][0:10000,:]
x2_fault = bearing_fault_dataset['X108_FE_time'][0:10000,:]
x1 = np.concatenate([x1_normal, x1_fault], axis=0)
x2 = np.concatenate([x2_normal, x2_fault], axis=0)
y_normal = np.zeros(x1_normal.shape)
y_fault = np.ones(x1_fault.shape)
X = np.concatenate([x1, x2], axis=1)
y = np.concatenate([y_normal, y_fault], axis=0)

# Shuffling
X_shuffled, y_shuffled = shuffle(X, y, random_state=14)

# train and test split
X_train, X_test, y_train, y_test = train_test_split(X_shuffled, y_shuffled,
                                                    test_size=0.2,
```



شکل ۱۲: قدر مطلق وزن ها

```
                                                    random_state=14)
```

Code 17: Data preparing(Python)

کد ۱۸ مربوط به آموزش و تست مدل آماده می‌باشد. دقت این مدل تنها 56.2 درصد است که نسبت به مدلی که به صورت دستی زدیم کم تر است. علت این پدیده می‌تواند این باشد که ما در این جا داده‌ها را بدون استخراج ویژگی به مدل دادیم در حالی که در حالت قبل ویژگی‌های زیادی از داده‌های خام استخراج کردیم و این خروجی اهمیت استخراج ویژگی را نشان می‌دهد.

```
#train
logistic_regression_model = LogisticRegression(max_iter=500, random_state=14)
logistic_regression_model.fit(X=X_train, y=y_train)
#test
y_hat = logistic_regression_model.predict(X_test)
y_hat_prob = logistic_regression_model.predict_proba(X_test)
accuracy = logistic_regression_model.score(X_test, y_test)
print('accuracy :', accuracy*100, '%')
```

Code 18: LogisticRegression train and test(Python)

۳ سوال سوم

۱.۳ ابتدا هیت مپ ماتریس همبستگی و هیستوگرام پراکندگی ویژگی ها را رسم و تحلیل کنید

همان طور که در کد ۱۹ مشاهده می‌شود پس از دانلود دیتاست سطرهایی که حاوی Null هستند حذف می‌شوند و سپس اطلاعات عددی از دیتاست استخراج می‌شود. نکته‌ی دیگری که وجود دارد این است که چون مقدار پارامتر Loud Cover ثابت و برابر با صفر است به ما کمکی نمی‌کند بنابراین این ویژگی را نیز از داده‌ها حذف کرده‌ایم.

```
!gdown 1gWTHhsD52p_0ZjqIE89UHsjfISBltKuC
weatherHistory = pd.read_csv('weatherHistory.csv')

weatherHistory = weatherHistory.dropna()
numeric_weather_df = weatherHistory.select_dtypes(include=['number'])
print(weatherHistory.keys())
numeric_weather_df = numeric_weather_df.drop(columns=['Loud Cover'])
```

Code 19: Data preparing(Python)

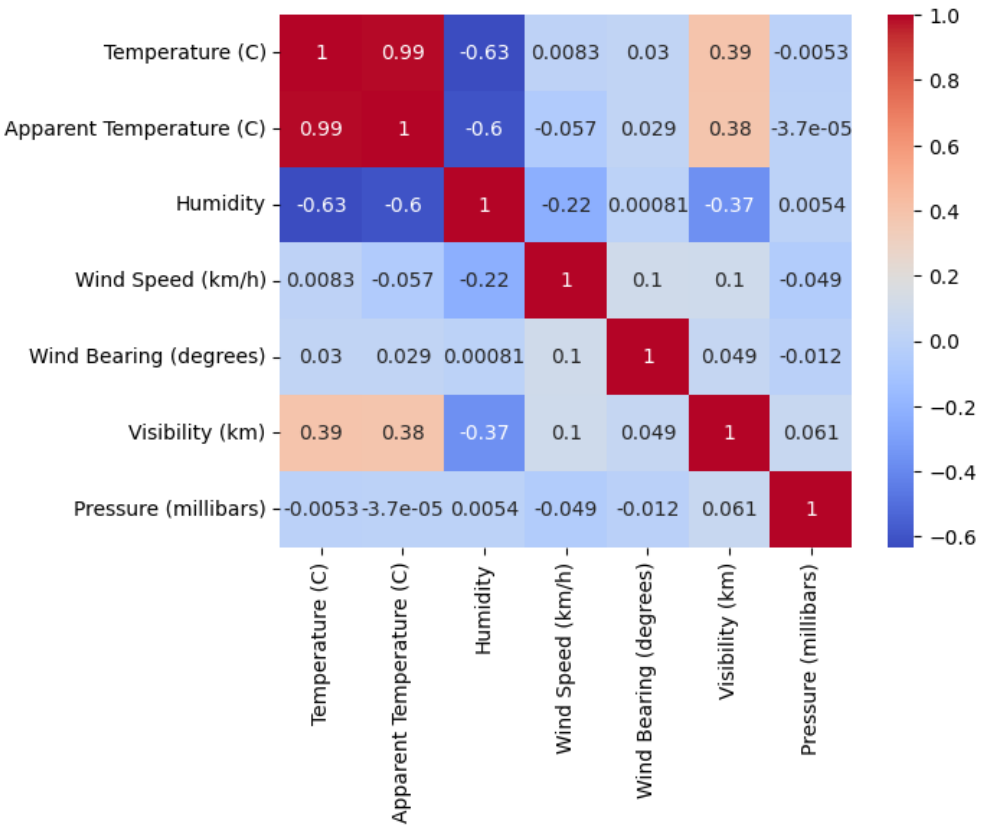
شکل ۱۳ هیت مپ و شکل ۱۴ هیستوگرام را نمایش می‌دهد. همان طور که از شکل ۱۳ مشخص است مقدار همبستگی بین رطوبت و دما زیاد است و این همبستگی منفی است. علاوه بر رطوبت، پارامتر Visibility نیز همبستگی قابل توجهی با دما دارد. شکل ۱۵ انواع همبستگی را نمایش می‌دهد.

۲.۳ روی این دیتاست، تخمین LS و RLS را با تنظیم پارامترهای مناسب اعمال کنید. نتایج به دست آمده را با محاسبه‌ی خطاها و رسم نمودارهای مناسب برای هر دو مدل با هم مقایسه و تحلیل کنید.

کد ۲۰ مربوط به پیاده‌سازی الگوریتم LS و RLS است.

```
class LSRegression():
  def __init__(self):
    self.coefficient = None

  def fit(self, X, y):
    X = np.column_stack((np.ones((len(y), 1)), X))
```



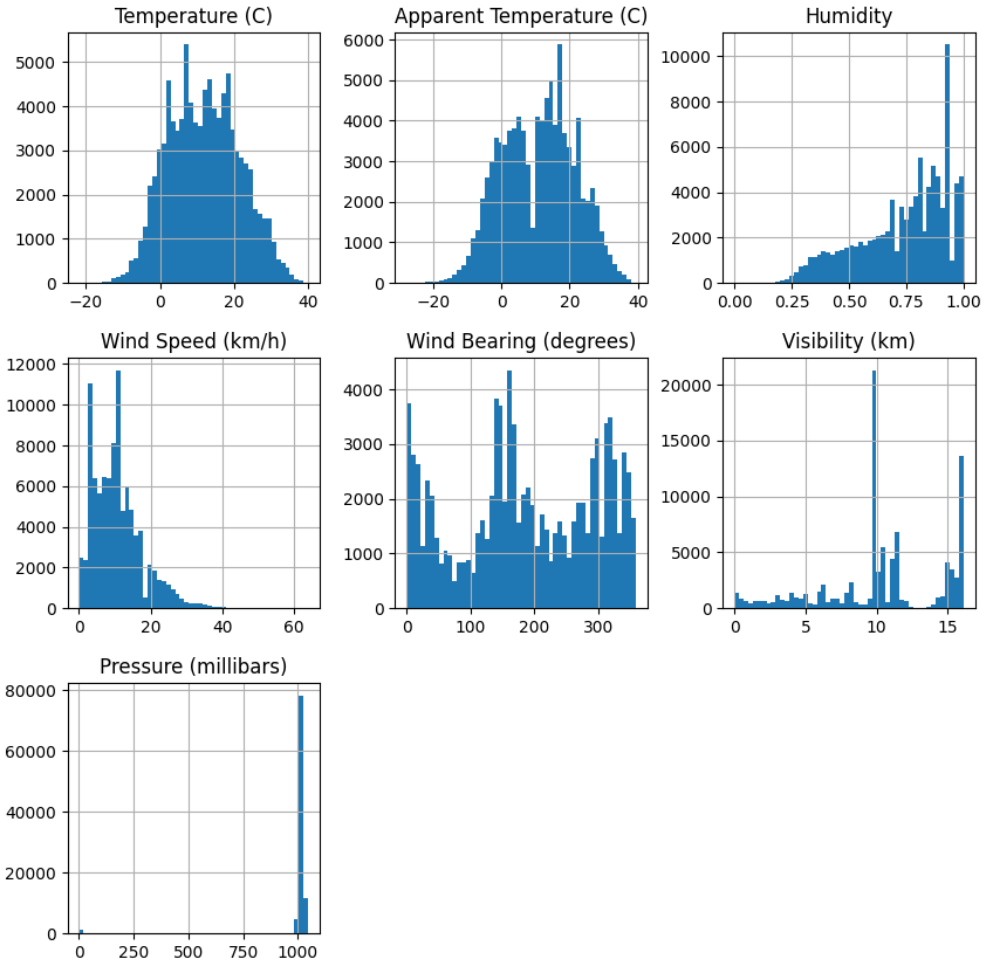
شکل ۱۳: هیت مپ

```
    self.coefficient = np.linalg.inv(X.T@X)@X.T@y

  def predict(self, X):
    X = np.column_stack((np.ones((len(X), 1)), X))
    return X @ self.coefficient

  def mse(self, y, y_hat):
    return np.mean((y-y_hat)**2)

class RLSRegression():
  def __init__(self, n_features, forgetting_factor=0.9):
    self.n_features = n_features
    self.forgetting_factor = forgetting_factor
    self.theta = np.zeros((n_features,1))
    self.P = np.eye(n_features)
```



شکل ۱۴: هیستوگرام

شکل ۱۵: انواع همبستگی

```
22
23  def fit(self, X, y):
24      errors = []
25
26      for i in range(len(X)):
```

```
        x_i = X[i].reshape(-1,1)
        y_i = y[i]

        # Predict
        y_pred = (x_i.T) @ self.theta

        # Update
        error = y_i - y_pred
        errors.append(error)
        K = (self.P @ x_i) / (self.forgetting_factor + ((x_i.T) @ self.P @ x_i))
        self.theta = self.theta + (K@error)
        self.P = (1 / self.forgetting_factor) * (self.P - (K @ (x_i.T) @ self.P)
    )

     return errors

  def predict(self, x):
        return x @ self.theta
```

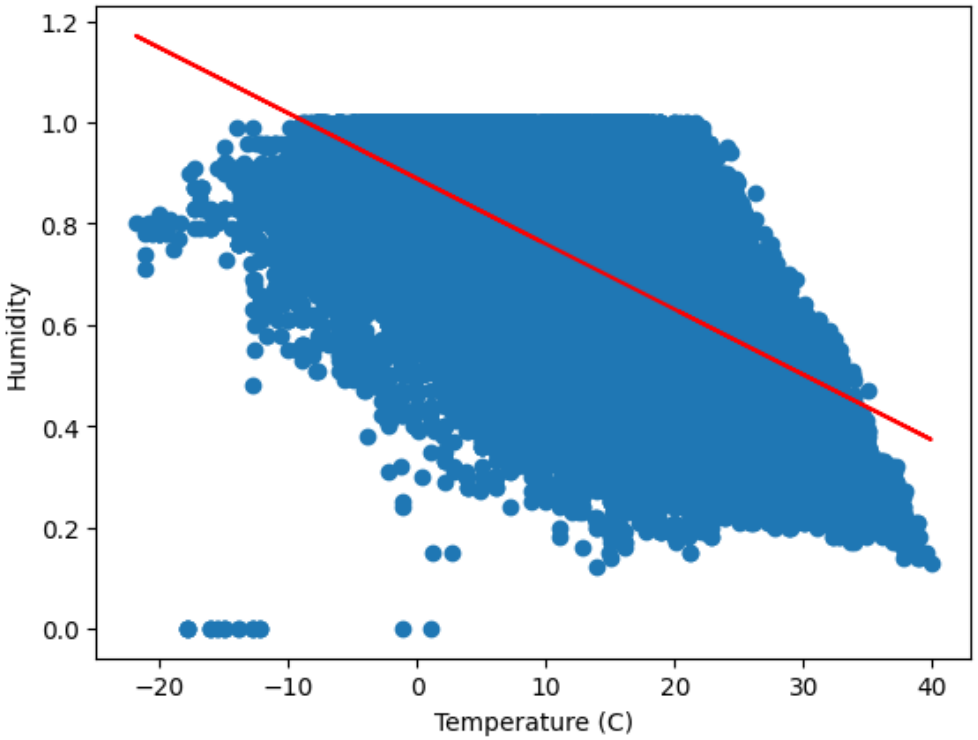
Code 20: LS and RLS implementation(Python)

کد ۲۱ مربوط به LS برای رابطه بین Humidity و Temperature است و شکل ۱۶ خروجی این کد است. در این شکل خط قرمز خطی است که برای رابطه‌ی بین Humidity و Temperature تخمین زده شده است. مقدار mse در اینجا برابر با 0.023 است. شکل ۱۷ مربوط به RLS برای رابطه بین Humidity و Temperature است و خط قرمز با آخرین شیب محاسبه شده رسم شده است به همین دلیل این خط روی نقاط قرار نگرفته. دلیل اصلی تفاوت RLS و LS این است که RLS شیب را مرتبا و به صورت زنده تغییر می‌دهد ولی در LS ابتدا کل داده‌ها را دریافت می‌کند و سپس شیب مناسب محاسبه می‌شود. mse برای RLS برابر با 0.14 است (forgetting_factor=0.75) . علت زیاد بودن خطا در RLS این است که باید شیب به صورت زنده تخمین زده شود.

```
X = numeric_weather_df['Temperature (C)'].values
y = numeric_weather_df['Humidity'].values
LSRegression_model = LSRegression()
LSRegression_model.fit(X, y)
y_hat = LSRegression_model.predict(X)
mse = LSRegression_model.mse(y, y_hat)
print('mse: ',mse)
plt.scatter(X, y)
```

```
plt.plot(X, y_hat, color='red')
plt.xlabel('Temperature (C)')
plt.ylabel('Humidity')
```

Code 21: Humidity and Temperature with LS(Python)



شکل ۱۶: رابطه‌ی بین Humidity و Temperature (LS)

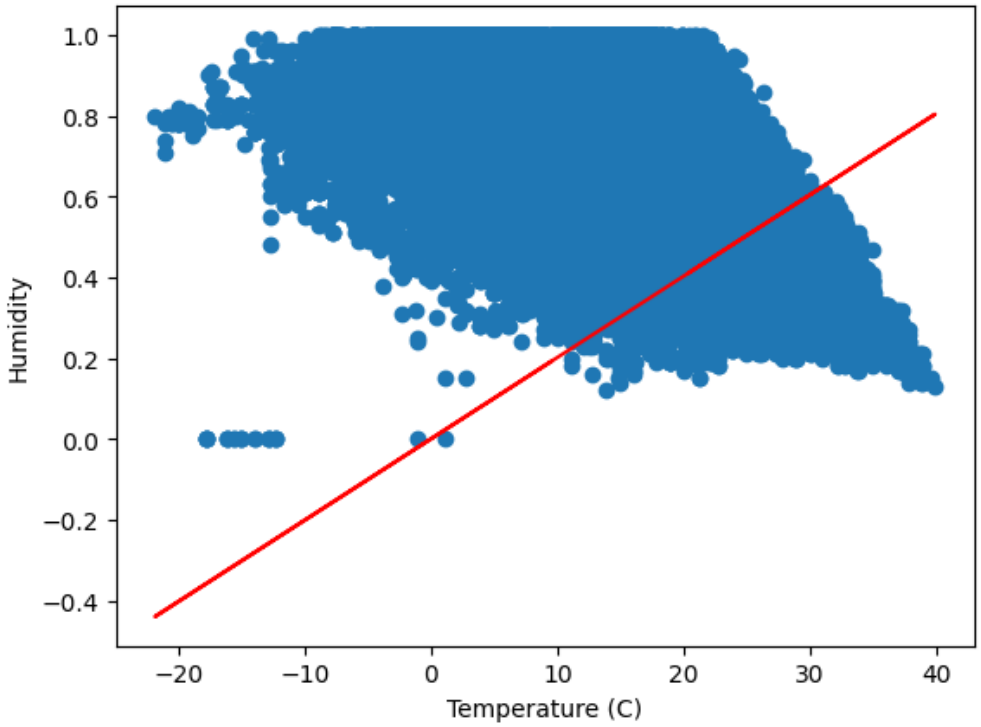
شکل ۱۸ و ۱۹ به ترتیب مربوط به LS و RLS هستند و رابطه‌ی بین Humidity و Apparent Temperature را تخمین می‌زنند. پارامتر mse برای LS برابر با 0.024 و برای RLS برابر با 0.153 است (forgetting_factor=0.6) .

۳.۳ در مورد **Weighted Least Square** توضیح دهید و آن را روی دیتاست داده شده اعمال کنید.

کد ۲۲ مربوط به پیاده‌سازی WLS و کد۲۳ مربوط به تست آن است.

```
class WLSRegression():
  def __init__(self):
    self.coefficient = None

```



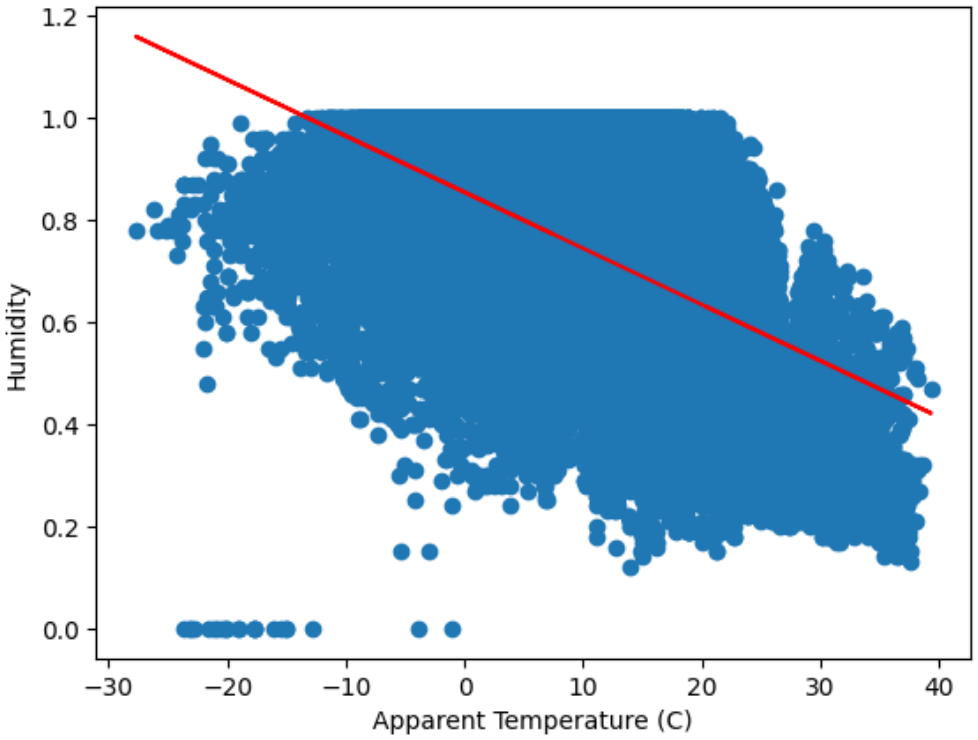
شکل ۱۷: رابطه‌ی بین Humidity و Temperature (RLS)

```
  def fit(self, x, y, w):
    X = np.column_stack((np.ones((len(y),1)),x))
    X = X.T
    W = np.diag(np.concatenate((np.array([1]),w), axis=0))
    self.coefficient = np.linalg.inv(X.T@W@X)@X.T@W@y

  def predict(self, x):
    X = np.column_stack(np.ones(len(y),1),x)
    return X @ self.coefficient
```

Code 22: WLS implement(Python)

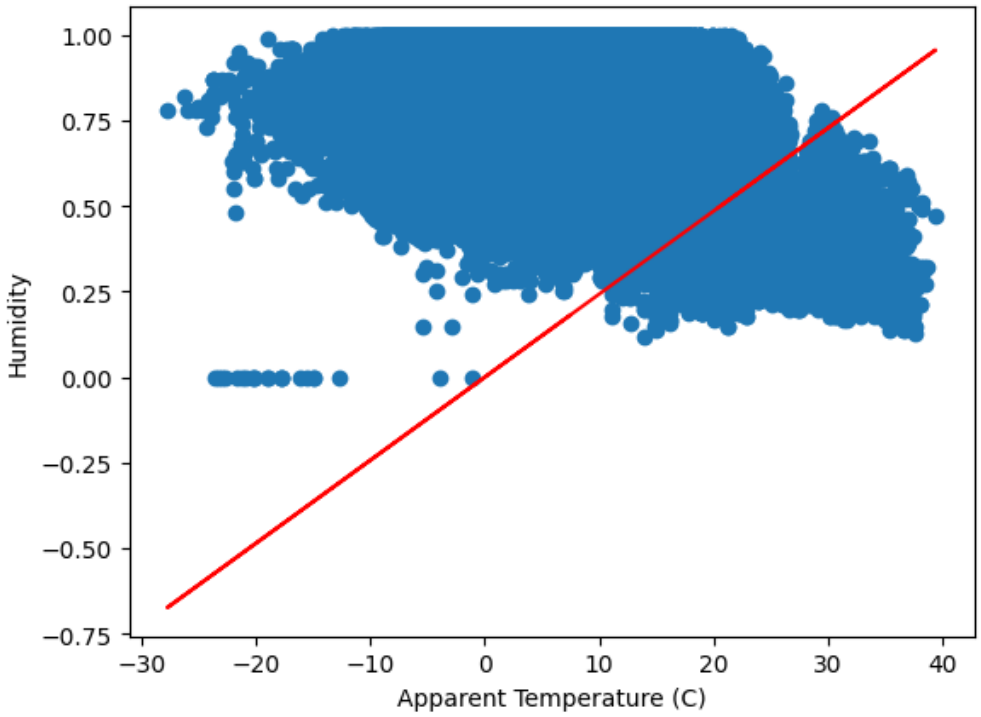
```
X = numeric_weather_df['Temperature (C)'].values
X = X[1:10000]
y = numeric_weather_df['Humidity'].values
y = y[1:10000]
WLSRegression_model = WLSRegression()
```



شکل ۱۸: رابطه‌ی بین Humidity و Temperature Apparent (LS)

```
w = 1;
w=np.array([w])
WLSRegression_model.fit(X, y, w)
y_hat = WLSRegression_model.predict(X)
mse = WLSRegression_model.mse(y, y_hat)
print('mse: ',mse)
plt.scatter(X, y)
plt.plot(X, y_hat, color='red')
plt.xlabel('Temperature (C)')
plt.ylabel('Humidity')
```

Code 23: WLS test(Python)



شکل ۱۹: رابطه‌ی بین Humidity و Temperature Apparent (RLS)