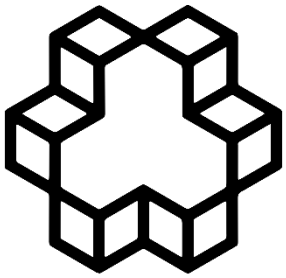


تمرین سری اول درس یادگیری ماشین

استاد:

دکتر مهدی علیاری شوره‌دلی

دانشجو:

نازنین بنداریان

۴۰۲۱۳۸۷۶

بهار ۱۴۰۳

فهرست

لینک گیت هاب:

https://github.com/nazaninbondarian/MachineLearning2024

۱– سوال اول

۱-۱-بخش الف

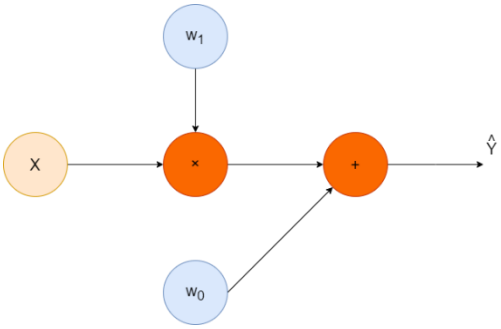
دسته‌بندی دو کلاسه در واقع ما با دو کلاس سر و کار داریم و به آن کلاس‌بندی باینری نیز گفته می‌شود. در واقع target های ما به صورت صفر و یک می‌باشد. اگر با بیش از دو کلاس سروکار داشته باشیم، کلاس‌بندی یا طبقه‌بندی چند کلاس می‌گوییم.

بیان دیگر برای تفکیک کلاس‌بندی به صورت خطی و غیرخطی می‌باشد. در طبقه‌بندی خطی از یک خط راست برای جداسازی و کلاس‌بندی داده‌ها استفاده می‌شود. در کلاس‌بندی غیرخطی مرزهایی که داده‌ها را از هم جدا می‌کنند به صورت دایروی، بیضی و یا سایر اشکال غیرخطی باشد. هرچقدر به مرز نزدیک می‌شویم میزان اطمینان از اینکه متعلق به کدام کلاس است کمتر می‌شود.

در حالت طبقه‌بندی خطی به صورت دو کلاسه یادگیر ماشین در واقع یک معادله خط را می‌یابد که داده‌ها را از یکدیگر جدا می‌کنند. در واقع ما به دنبال یافتن معادله خط به صورت زیر می‌باشیم.

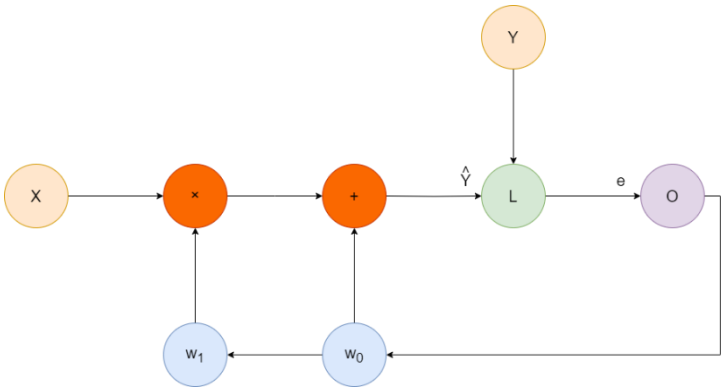
$$\hat{y} = w_1 x + w_0$$

که در آن $\hat{y}$ خروجی پیش‌بینی مدل، x ورودی، w_1 و w_0 پارامترهای مدل می‌باشند.



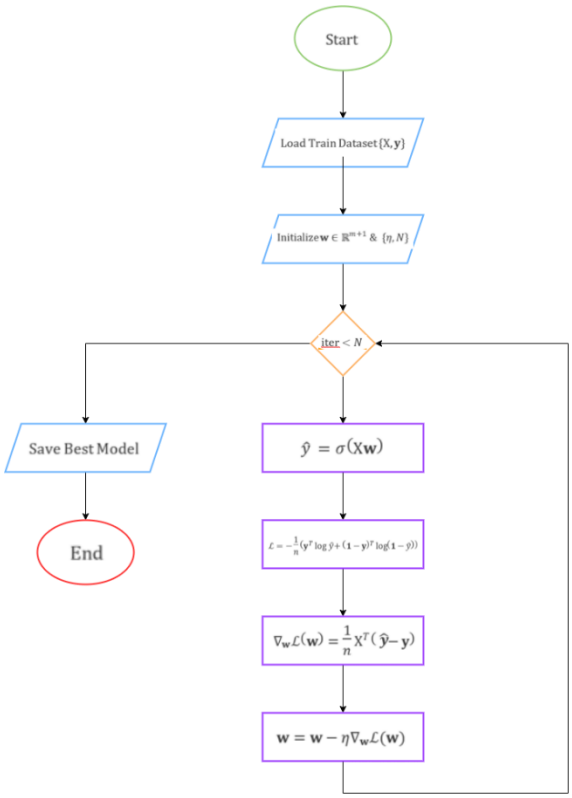
شکل ۱ ساختار بلوک دیاگرامی مربوط به معادله‌ی خط

در شکل زیر بلوک دیاگرام یادگیری مربوط به روش کلاس‌بندی خطی دو کلاسه نمایش داده شده است. اما نیازمند این هستیم که متوجه بشویم چقدر از هدف مطلوب دور می‌باشیم. L(Lost Function) تابع اتلاف می‌باشد که اختلاف بین خروجی تولید شده و خروجی مطلوب را می‌سنجد. سپس با یک تابع بهینه‌ساز O مانند گرادیان نزولی مقدار پارامترها را بروزرسانی می‌کنیم. مقادیر پارامترهای مدل تا زمانی که خطا به سمت صفر برود، بهینه سازی می‌شود.



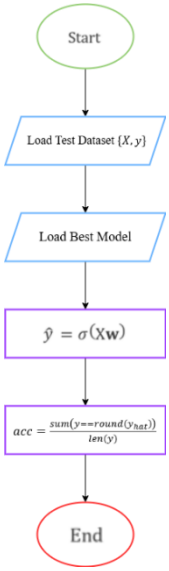
شکل ۲ ساختار بلوک دیاگرام یادگیری

دیاگرام کلی مراحل آموزش در شکل زیر نمایش داده شده است. در گام اول داده‌های آموزش را بارگذاری می‌کنیم. در گام دوم وزن‌ها را مقداردهی اولیه می‌کنیم. فرا پارامترهای میزان نرخ یادگیری η و تعداد دوره‌های آموزش را مقداردهی می‌کنیم. تا زمانی که از تعداد تکرار کمتر بودیم، وارد حلقه می‌شود و چهار گام را تکرار می‌کند؛ گام اول محاسبه خروجی مدل می‌باشد، گام دوم محاسبه‌ی تابع هزینه از مقایسه $\hat{y}$ و y، محاسبه‌ی گرادیان و بروزرسانی پارامترهای مدل. هنگامی که برابر تعداد تکرار شود یک مدل را ذخیره می‌کند، که بهترین مدل است و الگوریتم آموزش پایان می‌یابد.



شکل ۳ الگوریتم آموزش

در فرایند ارزیابی پس از بارگذاری داده‌های تست و مدل ذخیره شده در مرحله‌ی آموزش، مقداری خروجی $\hat{y}$ را محاسبه می‌کنیم. میزان دقت فرآیند را با محاسبه‌ی رابطه ذکر شده در گام دو بررسی می‌نماییم.



شکل ۴ الگوریتم ارزیابی

در صورتی که به صورت چند کلاسه بخواهیم دسته‌بندی کنیم، مرز تصمیم‌گیری ما به صورت یک ابر صفحه خواهد بود. اگر بخواهیم چندین کلاس را توسط چندین خط جدا کنیم، می‌توانیم از دو رویکرد متفاوت به صورت زیر پیش برویم.

رویکرد یکی در برابر بقیه (One vs Rest) می‌باشد. برای هر کلاس i یک تابع f_i را در نظر می‌گیرد که داده‌های کلاس i را از تمام کلاس‌های دیگر تفکیک کند، که در این حالت مسئله به یک طبقه‌بندی دو کلاس تبدیل شده است. به تعداد همه کلاس‌ها این رویکرد را بکار می‌بریم و همچنین طبقه‌بندی را طراحی می‌کنیم. در این حالت هر کدام از طبقه‌بندها هر کدام از کلاس‌های مشخص را از سایر کلاس‌ها جدا می‌کند.

رویکر دیگر رویکرد یکی در برابر دیگری (One vs One) می‌باشد. برای داده‌های هر کلاس باید به تعداد کل کلاس‌ها منهای یک طبقه‌بند خطی طراحی کنیم که در این حالت داده‌های کلاس به خصوص i را از تمام کلاس‌هی دیگر جدا کنیم. یعنی برای جدا کردن داده‌های کلاس i از کلاس j باید یک طبقه‌بند داشته باشیم ($f_{ij} = x, w_{ij}$). تعداد کل طبقه‌بندها در این رویکرد $\binom{C}{2}$ می‌باشد که بیشتر از حالت قبل می‌باشد (C تعداد کل کلاس‌ها می‌باشد).

در رویکرد اول این ابهام وجود دارد که ممکن است که یک بخش از فضا در طرف مثبت هیچ خطی قرار نگیرد (همزمان به هیچ کلاسی تعلق ندارند) و یا یک بخش از فضا در طرف مثبت بیش از دو خط قرار گیرند

(همزمان به دوکلاس تعلق داشته باشند). در این حالت به جای اینکه نگاه کنیم که طرف مثبت کدام خط قرار دارد، می‌توانیم فاصله تا خط‌ها را بیابیم. در این حالت کلاسی را به عنوان تخمینی از برچسب انتخاب می‌کنیم که مقدار f_i آن بیشتر شده است. در واقع f_i میزان امتیاز تعلق به آن کلاس می‌باشد. این یک روش ابتکاری است و بحث ریاضی قوی در پشت آن نیست.

$$\hat{y}^q = i^* = \arg\max_i f_i(x, w_i)$$

در رویکرد دوم نیز یکسری ابهامات وجود دارد. در ابتدا نحوه‌ی عملکرد این رویکرد را که چگونه طبقه‌بند چند کلاس را با طبقه‌بند باینری که ایجاد کرده‌ایم به دست بیاوریم، بررسی می‌کنیم. در این حالت برای یک کلاس خاص i به تعداد $C-1$ طبقه‌بند داریم که داده‌های کلاس i را از سایر کلاس‌ها تمیز می‌دهد. در ابتدا مقدار f_{ij} مربوط به هر یک از داده‌ها را محاسبه می‌کنیم. سپس بشماریم برای طبقه‌بندهای مربوط به هر یک از کلاس‌ها i مشخص، چه تعدادی f_{ij} مثبت می‌باشند. سپس برای iهای مختلف تکرار می‌کنیم. سپس کلاس i که تعداد بیشتری نتایج مثبت می‌دهد را به عنوان کلاس درست انتخاب کنیم. در این حالت طبقه‌بندهای بیشتری که در آن کلاس قرار دارند، معتقدند که این داده به این کلاس مشخص تعلق دارند.

در این رویکرد نیز این ابهام وجود دارد که ممکن است یک داده همزمان به بیش از یک کلاس متعلق باشد. در این حالت نیز برای رفع این ابهام می‌توانیم مقدار f_{ij} مربوط به هر یک از iهای مشخص را جمع کنیم و i که مقدار آن بیشتر است را بازگردانیم. که در این حالت یک امتیاز است یا یک تناسبی با فاصله‌ی نقطه از طبقه‌بند خطی ما (ابرصفحه) دارد. این روش نیز ابتکاری است و بحث ریاضی قوی در پشت آن نیست.

$$\hat{y}^q = i^* = \arg\max_i \sum_j f_{ij}(x, w_i)$$

در حالت کلی ما یک مشکل دیگری که داریم این می‌باشد که طبقه‌بندهای ما به صورت جداگانه آموزش می‌بینند و به دست می‌آیند که ما همه‌ی آن‌ها را بر روی هم قرار می‌دهیم و از آنها برای یک مسئله جدید استفاده می‌کنیم. ما باید سعی کنیم که همه‌ی آن‌ها همزمان آموزش ببینند.

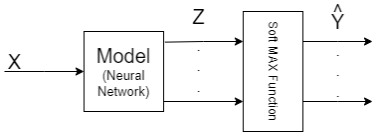
می‌توانیم مبحث Logistic Regression را با تابع هزینه Cross Entropy Loss به حالت چند کلاس تعمیم بدهیم. در این حالت داده وارد به مدل یادگیری ماشین می‌شود. به عنوان مثال مدل یادگیری ماشین را به صورت شبکه عصبی در نظر می‌گیریم. خروجی مدل (خروجی شبکه عصبی) ییک بردار به ابعاد تعداد کلاس‌ها می‌باشد. خروجی مدل به یک Soft Max Function داده می‌شود، که به صورت معادله‌ی زیر عمل می‌کند.

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}}$$

درایه‌های خروجی به صورت یک عدد مثبت در بازه‌ی [1 0] می‌باشند، که مجموع آن‌ها برابر یک می‌باشد. بردار خروجی $\hat{Y}$ به صورت یک بردار از توزیع احتمال است که احتمال تعلق به هر یک از کلاس‌ها را نمایش می‌دهد. برای آموزش درست پارامترهای مدل می‌توانیم از تابع هزینه Cross Entropy زیر استفاده کنیم.

$$J = -\sum_{q=1}^{Q}\sum_{i=1}^{C} y_i \log \hat{y}_i$$

در این رویکرد مشکل اینکه هر طبقه‌بند به صورت جداگانه آموزش می‌بیند را برطرف می‌کند و تمام پارامترهای مدل که خروجی‌های متفاوت تولید می‌کنند، به صورت همزمان آموزش می‌بینند. در این حالت خواص خوب Logistic Regression را دارد، یعنی کمینه کردن تابع هزینه هم ارز با بیشینه کردن تابع Likelihood می‌باشد که توجیه احتمالاتی قوی دارد.



شکل ۵ بلوک دیاگرام مربوط به طبقه‌بندی چند کلاس

۱-۲-بخش ب

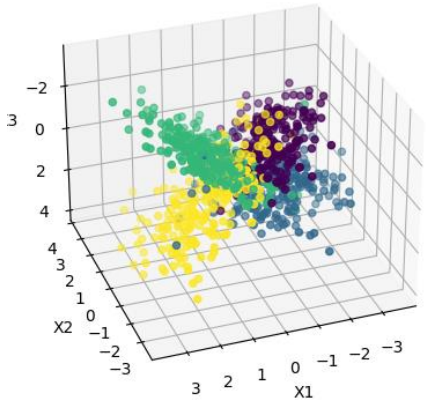
در این بخش یک دیتاست ۳ بعدی تولید می‌کنیم؛ بعد براساس تعداد ویژگی‌ها تعیین می‌شود. برای تولید داده از دستور زیر استفاده می‌کنیم.

```
from sklearn.datasets import  make_classification
X, y = make_classification(n_samples=1000, n_features=3,
                           n_redundant=0, n_clusters_per_class=1,
                           class_sep=1, n_classes=4, random_state=76)
```

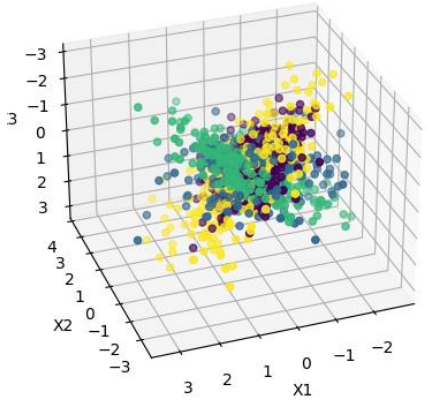
در ابتدا تابع make_classification را از کتابخانه sklearn.datasets فراخوانی می‌کنیم. آرگومان‌های مربوط به آن به شرح زیر است.

n_samples تعداد داده‌ها، n_features تعداد ویژگی‌ها و n_classes تعداد کلاس‌ها را مشخص می‌کنند. n_redundant تعداد داده‌های بی اهمیت را مشخص می‌کند. از آنجایی که بسیار از کارهایی که انجام می‌دهیم به صورت تصادفی می‌باشند، آرگومان random_state برای این منظور است که قابلیت تولید مجدد داشته باشیم، استفاده می‌شود. در واقع از درون یک جعبه تصادفی از پیش تعیین شده تعدادی داده برمی‌داریم. n_clusters_per_class به این منظور است که مشخص کنیم داده‌های ما چه تعداد نوع توزیع داشته باشد.

با استفاده از آرگومان class_sep می‌توانیم میزان چالش براگیزی و سختی داده‌های تولید شده را تغییر بدهیم. این آرگومان در واقع میزان تفکیک‌پذیری داده‌ها مشخص می‌کند. هرچقدر این عدد بزرگ باشد داده‌ها به راحتی قابل طبقه‌بندی می‌باشند؛ و هرچقدر این مقدار را کوچک کنیم، طبقه‌بندی داده‌ها دشوارتر می‌شود و مدل یادگیری ماشین ما دشوارتر می‌شود.



شکل ۶ داده‌ها با سختی کمتر



شکل ۷ داده با سختی بیشتر

۱-۳-بخش ج

در ابتدا داده‌ها را با استفاده از دستور زیر به دو بخش داده‌های یادگیری و ارزیابی تقسیم می‌کنیم (٪۲۰ از کل داده‌ها را به عنوان داده ارزیابی در نظر می‌گیریم).

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

سپس برای مدل یک شی ایجاد می‌کنیم و مراحل آموزش را ایجاد می‌کنیم.

```
model=LogisticRegression(multi_class='multinomial',solver='sag',max_iter=200,random_state=76)

model.fit(x_train, y_train)
```

در حالت چند کلاسه، الگوریتم آموزشی در صورتی که گزینه multi_class روی ovr تنظیم شده باشد از طرح one-vs-rest (ovr) استفاده می‌کند و اگر گزینه multi_class روی multinomial تنظیم شده باشد از تلفات آنتروپی متقابل استفاده می کند (در حال حاضر گزینه multinomial فقط توسط حل کننده‌های lbfgs، sag، saga و newton-cg پشتیبانی می‌شود).

می‌توانیم به صورت احتمالی و لگاریتمی نیز میزان تعلق به کلاس‌ها را نیز مشخص کنیم.

```
model.predict_proba(x_test)

model.predict_log_proba
```

با استفاده از دستور زیر میزان دقت را بررسی می‌کنیم.

```
model1.score(x_train, y_train)

model.score(x_test, y_test)
```

جدول ۱ ارزیابی روش LogisticRegression

	Train	Test
multinomial	0.835	0.835
ovr	0.83375	0.825

در اینجا می‌خواهیم با SGDClassifier کار کنیم. پس از ایجاد شی برای آموزش از دستور زیر استفاده می‌کنیم. تابع اتلاف به صورت پیش فرض روی hing است و ما آن را به log_loss تغییر می‌دهیم.

```
model2 = SGDClassifier(loss='log_loss', random_state=76)

model2.fit(x_train, y_train)
```

میزان امتیاز را نیز مشابه دستور قبل محاسبه می‌کنیم.

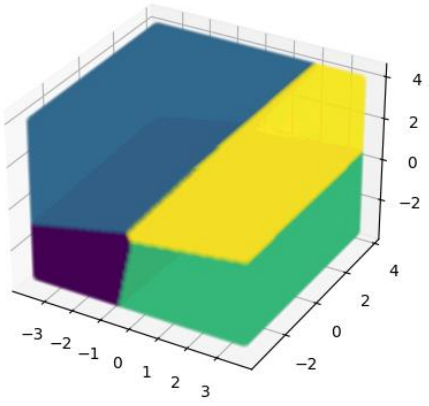
جدول ۲ ارزیابی روش SGDClassifier

	Train	Test
SGDClassifier	0.82	0.835

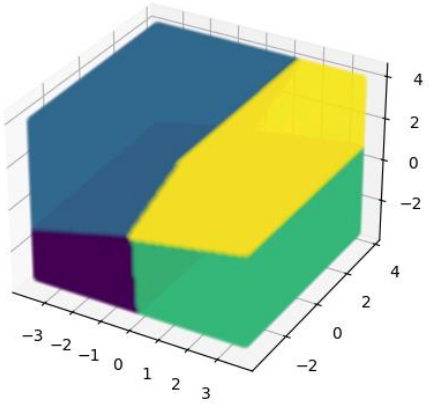
۱-۴-بخش د

با استفاده از دستورات زیر صفحه تصمیم‌گیری برای سه حالت را نصب می‌کنیم.

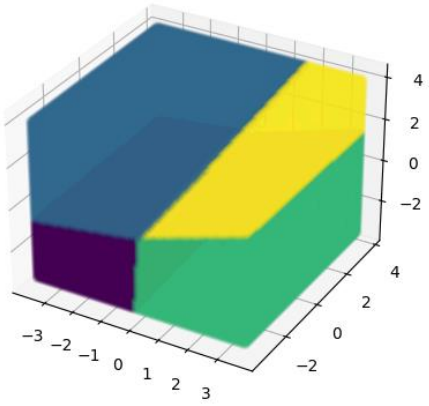
```
# Generate mesh grid
x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
z_min, z_max = X[:, 2].min() - 0.1, X[:, 2].max() + 0.1
xx, yy, zz = np.meshgrid(np.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100),
                         np.linspace(z_min, z_max, 100))
# Make predictions
mesh_input = np.c_[xx.ravel(), yy.ravel(), zz.ravel()]
predictions = model.predict(mesh_input)
predictions = np.round(predictions).astype(int)
# Plot decision regions
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap='viridis')  # Original data points
ax.scatter(mesh_input[:, 0], mesh_input[:, 1],mesh_input[:, 2],c=predictions, cmap='viridis', alpha=0.1)  # Decision regions
plt.show()
```



شکل ۸ صفحه تصمیم‌گیری حالت اول



شکل ۹ مرز تصمیم‌گیری حالت دوم



شکل ۱۰ مرز تصمیم‌گیری حالت سوم

۱-۵-بخش ه

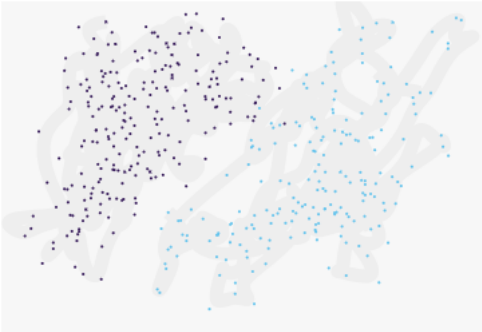
در ابتدا ماژول مورد نظر را نصب می‌کنیم.

```
!pip install --upgrade drawdata
```

سپس با دستورات زیر صفحه‌ای بر روی آن باز می‌شود که می‌توانیم با ماووس داده‌ها را بکشیم. سپس فایل csv. مربوط به داده‌ها را ذخیره می‌کنیم و با استفاده از روش سوم آپلود فایل به کمک gdown فایل را آپلود می‌کنیم (در ابتدا یک پوشه ایجاد می‌کنیم). در بخش آدرس share مربوط به گوگل درایو قرار می‌گیرد.

```
from drawdata import draw_scatter
scatter_plot = draw_scatter()
scatter_plot
!pip install --upgrade --no-cache-dir gdown
```

```
!gdown 1vM7KhyVHEAzzij1fFBf9dyjJHkp9UScB
```



شکل ۱۱ شکل مصور داده رسم شده

توسط دستور زیر فایل را می‌خوانیم و سپس بررسی می‌کنیم که داده null موجود نباشد.

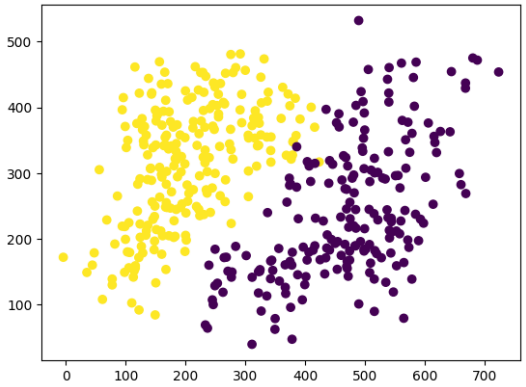
```
df = pd.read_csv('/content/Data/data.csv')
null_counts = df.isnull().sum()
```

داده‌های مربوط به کلاس a را به برچسب ۱، و داده‌های کلاس b را به برچسب صفر تغییر می‌دهیم.

```
df['z'] = df['z'].replace('a', '1')
df['z'] = df['z'].replace('b', '0')
```

سپس داده ورودی و خروجی را انتخاب می‌کنیم که به صورت زیر قابل نمایش می‌باشند.

```
X1 = df[['x']]
X2 = df[['y']]
y = df[['z']]
```



شکل ۱۲ نمایش دیتاها

سپس با استفاده از دستور زیر داده‌های آموزش و آزمایش را ایجاد می‌کنیم.

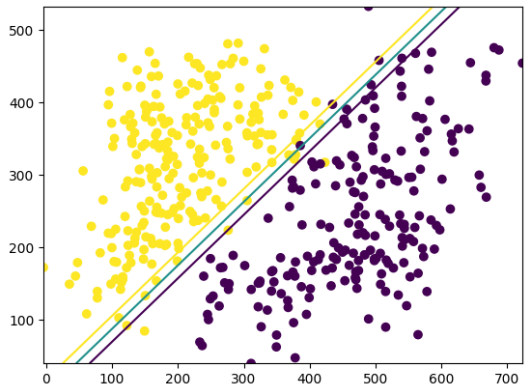
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

در اینجا این ایراد وارد است که ما تعداد دقیق داده‌های مربوط به هر دسته را نمی‌دانیم. با فرض برابری این دستور را استفاده نموده‌ایم. سپس برای دو مدل زیر مراحل قبل را پیش می‌بریم و خط تصمیم‌گیری را رسم می‌نماییم.

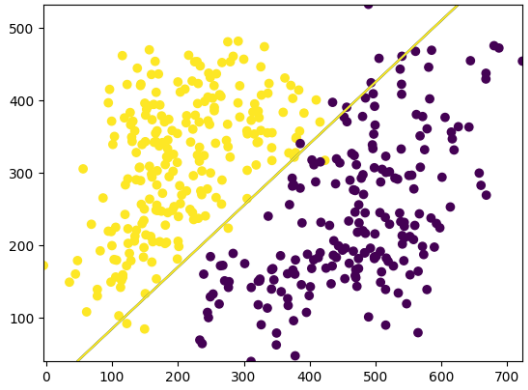
```
model = LogisticRegression(solver='sag', max_iter=200, random_state=76)
model1 = LogisticRegression (loss='log_loss', random_state=76)
```

جدول ۳ مقادیر امتیاز دو روش

	Train	Test
LogisticRegression	0.9739884393063584	0.9770114942528736
LogisticRegression	0.9710982658959537	0.9885057471264368



شکل ۱۳ مرز تصمیم‌گیری مدل اول



شکل ۱۴ مرز تصمیم‌گیری مدل دوم

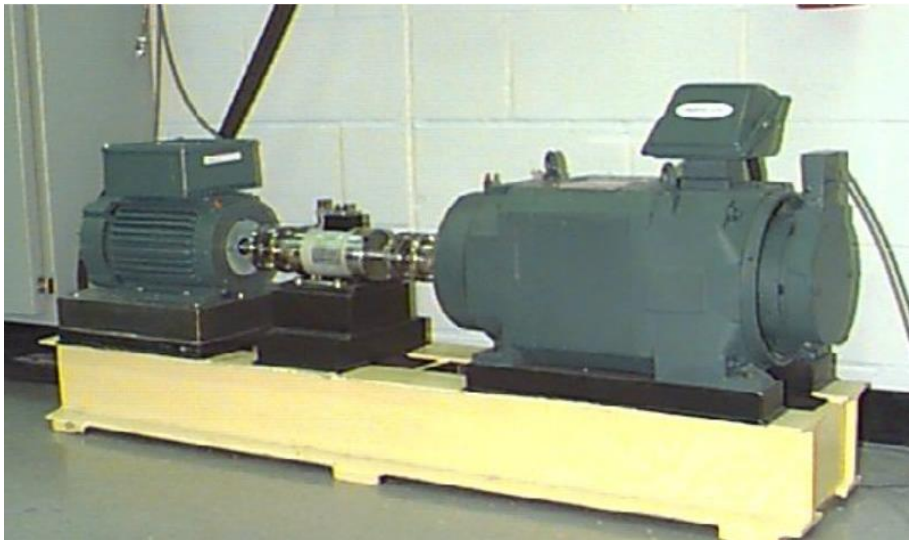
۲–سوال دوم

۲-۱-بخش الف

این داده‌ها شامل داده‌های تست بلبرینگ را برای بلبرینگ‌های معمولی و معیوب فراهم می کند. آزمایش‌ها با استفاده از یک موتور Reliance Electric با قدرت ۲ اسب بخاری انجام شده است و داده‌های شتاب در مکان‌های نزدیک و دور از بلبرینگ‌های موتور اندازه‌گیری شد. در جمع‌آوری این داده‌ها شرایط آزمایش واقعی موتور و همچنین وضعیت خطای بلبرینگ برای هر آزمایش به دقت مستند شده است.

بلبرینگ‌های موتور با استفاده از ماشین‌کاری تخلیه الکتریکی (EDM) با عیوب کاشته شدند. گسل‌هایی با قطر ۰.۰۰۷ اینچ تا قطر ۰.۰۴۰ اینچ به طور جداگانه در مسیر ورودی داخلی، عنصر نورد (یعنی توپ) و مسیر بیرونی معرفی شدند. بلبرینگ‌های معیوب دوباره در موتور آزمایشی نصب شدند و داده‌های ارتعاش برای بارهای موتور ۰ تا ۳ اسب بخار (سرعت موتور ۱۷۲۰ تا ۱۷۹۷ RPM) ثبت شد.

آزمایش‌ها اغلب به منظور تأیید فناوری‌ها، نظریه‌ها و تکنیک‌های جدید مورد نیاز است. این آزمایش‌های بلبرینگ موتور به منظور مشخص کردن عملکرد IQ PreAlert، یک سیستم ارزیابی وضعیت یاتاقان موتور توسعه‌یافته در Rockwell آغاز شد. از زمان این انگیزه اولیه، برنامه آزمایشی برای ارائه یک پایگاه داده عملکرد موتور گسترش یافته است که می‌تواند برای اعتبارسنجی و/یا بهبود مجموعه‌ای از تکنیک‌های ارزیابی وضعیت حرکتی مورد استفاده قرار گیرد. برخی از پروژه‌هایی که اخیراً یا در حال حاضر از این پایگاه داده استفاده می‌کنند عبارتند از: فناوری ارزیابی وضعیت Winsnode، تکنیک‌های تشخیصی مبتنی بر مدل، و الگوریتم‌های تعیین سرعت موتور.



شکل ۱۵ سیستم مورد آزمایش

همانطور که در شکل بالا نشان داده شده است، پایه تست از یک موتور ۲ اسب بخاری (سمت چپ)، یک مبدل/رمزگذار گشتاور (مرکز)، یک دینامومتر (سمت راست) و الکترونیک کنترل (نمایش داده نشده) تشکیل شده است. بلبرینگ‌های تست محور موتور را پشتیبانی می‌کنند. خطاهای تک نقطه‌ای با استفاده از ماشینکاری تخلیه الکتریکی با قطر خطای ۷ میل، ۱۴ میل، ۲۱ میل، ۲۸ میل و ۴۰ میل ($1mil = 0.001inch$) به بلبرینگ‌های آزمایشی معرفی شدند. یاتاقان‌های SKF برای گسل‌های با قطر ۷، ۱۴ و ۲۱ میل و یاتاقان‌های معادل NTN برای گسل‌های ۲۸ میل و ۴۰ میل استفاده شد.

داده‌های ارتعاش با استفاده از شتاب‌سنج‌هایی که با پایه‌های مغناطیسی به محفظه متصل شده بودند، جمع‌آوری شد. شتاب سنج‌ها در موقعیت ساعت ۱۲ هم در انتهای محرک و هم در انتهای فن محفظه موتور قرار گرفتند. در طی برخی آزمایش‌ها، یک شتاب‌سنج نیز به صفحه پایه نگهدارنده موتور متصل شد. سیگنال‌های ارتعاشی با استفاده از یک ضبط کننده ۱۶ کانالی DAT جمع‌آوری و در محیط Matlab پردازش شدند. همه فایل‌های داده در قالب Matlab (*.mat) هستند. داده‌های دیجیتال با ۱۲۰۰۰ نمونه در ثانیه و همچنین داده‌ها با ۴۸۰۰۰ نمونه در ثانیه برای خطاهای بلبرینگ انتهای درایو جمع آوری شد. داده‌های سرعت و اسب بخار با استفاده از مبدل/رمزگر گشتاور جمع‌آوری شد و با دست ثبت شد.

خطاهای راهرو بیرونی، خطاهای ثابت هستند، بنابراین قرارگیری عیب نسبت به ناحیه بار یاتاقان تأثیر مستقیمی بر پاسخ لرزشی سیستم موتور/بیرینگ دارد. به منظور تعیین کمیت این اثر، آزمایش‌هایی برای یاتاقان‌های انتهایی فن و درایو با گسل‌های بیرونی راه آهن واقع در ساعت ۳ (مستقیماً در ناحیه بار)، در ساعت ۶ (متعامد با ناحیه بار) و در ساعت ۱۲ انجام شد.

برای همه فایل ها، مورد زیر در نام متغیر نشان می دهد:

DE - داده های شتاب سنج انتهای درایو

FE - داده های شتاب سنج انتهای فن

BA - داده های شتاب سنج پایه

Time - داده های سری زمانی

RPM - دور در دقیقه در طول تست

۲-۲-بخش ب

۲-۲-۱-ایجاد ماتریس $M \times N$ از داده‌های هر کلاس

در ابتدا یک فایل برای ذخیره سازی داده‌ها ایجاد می‌کنیم. سپس فایل mat. مربوط به داده‌ست را توسط دستورات زیر دریافت می‌کنیم.

```
!wget -q https://engineering.case.edu/sites/default/files/97.mat
!wget -q https://engineering.case.edu/sites/default/files/105.mat
```

فایل‌های دریافت شده را توسط دستور زیر آپلود می‌کنیم.

```
from scipy.io import loadmat
NN = loadmat("/content/Data/97.mat")
FF = loadmat("/content/Data/105.mat")
```

سپس با عنوان سرستون‌های به دست آمده داده‌ها را به صورت فایل csv. تبدیل کرده‌ایم و ذخیره می‌کنیم.

```
Normal=pd.DataFrame({'X097_DE_time': NN1.flatten(), 'X105_BA_time': NN2.flatten()})
Normal.to_csv("/content/Data/Normal.csv",index=False)
```

سپس بررسی صورت می‌گیر آیا داده null وجود دارد یا خیر. از آنجایی که یافت نشد نیاز به حذف ستونی نمی‌باشد. سپس با توجه به درخواست صورت سوال ماتریس داده‌های با ابعاد 200×100 ایجاد می‌کنیم.

```
# Extract samples from class Normal
for i in range(num_samples):
    start_idx = np.random.randint(0, len(Normal_data) - sample_length + 1)
    sample = Normal_data[start_idx:start_idx + sample_length]
    Normal_samples.append(sample)
```

سپس دو ماتریس ایجاد شده را به هم متصل می‌کنیم.

```
data_matrix = np.vstack((Normal_samples, Fault_samples))
```

برای داده‌های Normal برچسب یک و داده‌های Fault برچسب صفر اطلاق می‌شود.

```
Normal_labels = np.ones((num_samples, 1))
Fault_labels = np.zeros((num_samples, 1))
```

یک ماتریس از برچسب‌ها با استفاده از دستور زیر ایجاد می‌کنیم و در نهایت ماتریس برچسب و داده‌ها را بهم متصل می‌کنیم.

```
labels = np.vstack((Normal_labels, Fault_labels))
main_matrix = np.hstack((data_matrix, labels))
```

۲-۲-۲-استخراج ویژگی

به صورت خلاصه می‌توان دلایل استخراج ویژگی را در زیر بیان نمود.

۱. **ابزار پردازش داده:** ماتریس داده‌ها معمولاً شامل اطلاعات بزرگی است که به سادگی قابل فهم نیستند. با استخراج ویژگی‌ها، می‌توان این داده‌ها را به فرمت‌های ساده‌تر و قابل فهم‌تر تبدیل کرد.

۲. **کاهش بعدی:** معمولاً ماتریس‌های داده دارای ابعاد بزرگی هستند که ممکن است به مشکلات محاسباتی و ذخیره‌سازی منجر شوند. با استخراج ویژگی‌ها، می‌توان بعد ماتریس را کاهش داد و بهینه‌سازی فضای حافظه و محاسباتی را انجام داد.

۳. **بهبود عملکرد مدل:** ویژگی‌های مناسب و معنی‌دار می‌توانند عملکرد مدل را بهبود بخشیده و به دقت و قدرت پیش‌بینی آن کمک کنند.

۴. **رفع مشکلات عدم تعادل:** در مواردی که داده‌ها ناهموارتر و یا عدم تعادل دارند، استخراج ویژگی‌ها می‌تواند به توازن و بهبود کیفیت مدل کمک کند.

۵. **تسهیل در تفسیر نتایج:** با استخراج ویژگی‌های مهم، نتایج مدل قابل فهم‌تر می‌شوند و این امکان را فراهم می‌کند که دلایل تصمیم‌گیری مدل را بیشتر درک کنیم.

۶. **سازگاری با الگوریتم‌های مختلف:** استخراج ویژگی‌ها معمولاً مستقل از نوع الگوریتم است و می‌تواند با هر الگوریتمی سازگار باشد.

در این بخش ما ۹ ویژگی را از ماتریس ساخته شده در بخش قبل استخراج می‌کنیم.

```
std_dev_values = np.std(data_matrix, axis=1)
peak_values = np.max(np.abs(data_matrix),axis=1)
crest_factor_values=np.max(np.abs(data_matrix),axis=1)/np.sqrt(np.mean(data_matrix**2, axis=1))
peak_to_peak_values = np.max(data_matrix, axis=1)-np.min(data_matrix, axis=1)
```

```
shape_factor_values=np.sqrt(np.mean(data_matrix**2,axis=1))/np.mean(np.abs(data_matrix), axis=1)

mean_values = np.mean(data_matrix, axis=1)

rms_values = np.sqrt(np.mean(data_matrix**2, axis=1))

absolute_mean_values = np.mean(np.abs(data_matrix), axis=1)

impulse_factor_values=np.abs(np.max(data_matrix,axis=1))/np.mean(np.abs(data_matrix), axis=1)
```

سپس ماتریس ویژگی را به کمک دستور زیر ایجاد می‌کنیم.

```
features=np.column_stack((std_dev_values,peak_values,crest_factor_values,peak_to_peak_values,shape_factor_values,mean_values,absolute_mean_values,rms_values,impulse_factor_values))
```

۲-۲-۳-برش زدن

دلایل برش زدن را مب‌توان به صورت موردی زیر بیان کنیم.

۱. **تنظیمات ضعیف:** ممکن است برش زدن دیتاست به دلیل تنظیمات نادرست در فرآیند جمع‌آوری داده‌ها اتفاق بیفتد، مانند ناهمگنی در جمع‌آوری اطلاعات یا عدم کیفیت داده‌ها.

۲. **تعیین مقیاس نامناسب:** در برخی موارد، برش زدن دیتاست به دلیل تعیین مقیاس نامناسب می‌باشد، به عنوان مثال، استفاده از مقیاس کوچکتر یا بزرگتر از حد مورد نیاز.

۳. **عدم کیفیت و ناهمگنی داده‌ها:** در صورت وجود داده‌های ناهمگن یا ناکارآمد، برش زدن دیتاست می‌تواند بهبودی فراهم کند و از دقت مدل‌های آموزش دیده با این داده‌ها افزایش دهد.

۴. **نیاز به تبدیل داده:** برای استفاده از داده‌ها در مدل‌های مختلف، ممکن است نیاز به تبدیل داده‌ها به فرمت‌ها و ساختارهای متفاوت باشد، که این فرآیند ممکن است شامل برش زدن دیتاست باشد.

کد np.random.seed(76) را برای مولد اعداد تصادفی NumPy روی ۷۶ تنظیم می‌کند. این تضمین می‌کند که دنباله اعداد تصادفی تولید شده توسط NumPy قابل تکرار خواهند بود اگر یک کد را چندین بار با همان seed اجرا کنید. مقدار seed می‌تواند هر عدد صحیحی باشد. استفاده از یک Seed ثابت برای اهداف اشکال زدایی و آزمایش مفید است، زیرا به شما امکان می دهد هر بار که کد خود را اجرا می‌کنید اعداد تصادفی یکسانی را بدست آورید و از نتایج ثابت اطمینان حاصل کنید. در ابتدا دو ماتریس داده و برچسب را به هم متصل می‌کنیم.

```
Labeled_Data=np.hstack((Data, labels))
```

```
np.random.seed(76)
np.random.shuffle(Labeled_Data)
```

سپس داده‌های ورودی و خروجی را تعیین می‌کنیم و با استفاده از کتابخانه sklearn داده‌ها را به دو بخش آموزش و آزمایش تقسیم می‌کنیم.

```
X = Labeled_Data[['Standard Deviation','Peak', 'Crest Factor', 'Peak to Peak','Shape Factor', 'Mean', 'Absolute Mean', 'RMS', 'Impulse Factor']].values
y = Labeled_Data[['Target']].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=76)
```

۲-۲-۴-نرمال‌سازی داده‌ها

دلایل برش زدن را مب‌توان به صورت موردی زیر بیان کنیم.

۱. **نرمال‌سازی حداقل و حداکثر:** این روش ویژگی‌ها را در یک محدوده ثابت، معمولا بین ۰ و ۱ مقیاس می‌کند و فرمول نرمال‌سازی حداقل حداکثر

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

که در آن X مقدار اصلی، X_{min} حداقل مقدار مجموعه داده و X_{max} حداکثر مقدار است. این روش توزیع اصلی داده‌ها را حفظ می‌کند، اما می‌تواند به موارد پرت حساس باشد.

۲. **نرمال‌سازی امتیاز Z (استانداردسازی):** در این روش ویژگی‌ها را به میانگین صفر و انحراف استاندارد ۱ تبدیل می‌کند. با استفاده از فرمول زیر محاسبه می‌شود

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

که در آن X مقدار اصلی، μ میانگین داده‌های مجموعه و σ انحراف استاندارد می‌باشد. این نرمالسازی نسبت به داده‌های پرت مقاوم است و برای تمام داده‌ها ارزش یکسانی دارند.

ما به کمک دستور زیر هم داده‌های آموزش و آزمایش را در نرمالایز می‌کنیم؛ زیرا شبکه در ابتدا با استفاده از داده نرمالایز آموزش دیده است و برای اعتبارسنجی از همان شبکه استفاده می‌کنیم.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train_n = scaler.fit_transform(X_train)
```

۲-۳-بخش ج

در اولین گام تابع سیگموییدد را تعریف می‌کنیم.

$\sigma(x) = \frac{1}{1+e^{-x}}$

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

در مرحله‌ی بعد تابع logestic regression را تعریف می‌کنیم که در آن مقدار $W \times x$ را از تابع سیگموییدد عبور می‌دهیم.

$\hat{y} = \sigma(\mathrm{w}^T \mathrm{x})$

```
def logistic_regression(x, w):
    y_hat = sigmoid(x @ w)
    return y_hat
```

در گام بعدی ما می‌خواهیم تابع اطلاق کراس آنتروپی باینری را تعریف کنیم.

$L = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right)$

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

در مرحله‌ی بعدی نوبت به محاسبه‌ی گرادیان می‌پردازیم.

$\nabla_w L(w) = \frac{1}{n} x^T(\hat{y} - y)$

```
def gradient(x, y, y_hat):
    grads = (x.T @ (y_hat - y)) / len(y)
    return grads
```

سپس وزن‌های سیستم را بروزرسانی می‌کنیم.

$w = w - \eta \nabla_w L(w)$

```
def gradient_descent(w, eta, grads):
    w -= eta*grads
    return w
```

ارزیابی سیستم‌های کلاس‌بندی خطی باینری معمولاً با استفاده از معیارهایی انجام می‌شود که به طور کلی شامل موارد زیر است:

۱. **دقت (Accuaracy):** نسبت تعداد داده‌هایی که به درستی تشخیص داده شده‌اند به کل داده‌های مورد ارزیابی است. این معیار می‌تواند به صورت فرمولی به صورت زیر نشان داده شود:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

۲. **دقت تشخیص مثبت (Precision):** نسبت داده‌هایی که به درستی تشخیص داده شده‌اند به تمام داده‌هایی که به عنوان مثبت تشخیص داده شده‌اند. فرمول دقت تشخیص مثبت به صورت زیر است:

$$Precision = \frac{TP}{TP+FP}$$

۳. **حساسیت (Recall):** نسبت داده‌هایی که به درستی تشخیص داده شده‌اند به کل داده‌هایی که در واقع مثبت هستند. فرمول حساسیت به صورت زیر است:

$$Recall = \frac{TP}{TP+FN}$$

۴. **اندازه‌گیری F1(F1 scoare):** میانگین هارمونیک دقت و حساسیت است و به صورت زیر محاسبه می‌شود:

$$F1 = \frac{Precesion \times Recall}{Precesion+Recall} \times 2$$

این معیارها از جمله معیارهای ارزیابی رایج برای سیستم‌های کلاس‌بندی خطی باینری هستند. در اینجا معیار اول و دوم را در نظر می‌گیریم.

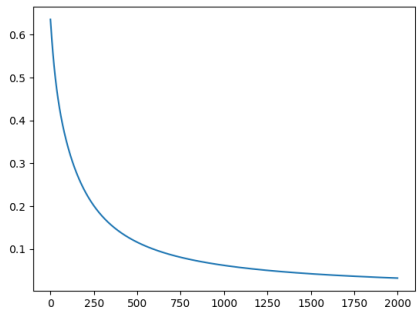
```
def accuracy(y, y_hat):
    acc = np.sum(y == np.round(y_hat)) / len(y)
    return acc
def precition(y, y_hat):
  # true positives and false positive
  TFP = []
  for i in range(len(y_hat)):
    if y_hat[i]>0.5:
      TFP.append(i)
```

```
P = []
for j in range(len(y)):
  if y[j]==1:
    P.append(j)
TP = []
for k in range(len(TFP)):
  if P[k]==TFP[k]:
    TP.append(k)
pre = len(TP)/len(TFP)
return pre
```

سپس مقادیر پارامترهای هایپر را تعیین می‌کنیم. برای ورودی بایاس که یک می‌باشد، یک ستون به بردار ورودی متصل می‌کنیم.

```
x_train = np.hstack((np.ones((len(x_train_n), 1)), x_train_n))
```

سپس قبل از حلقه یک آرایه برای ذخیره تمام خطاها برای رسم نمودار اتلاف در طول دوره ایجاد می‌کنیم. براساس نمودار اتلاف‌ها نمی‌توانیم در رابطه با بهینه بودن کامل اذعان نمود. زیرا در صورتی که برنامه را مجدد اجرا کنیم، با وزن‌های بروزرسانی شده در ۲۰۰۰ دوره‌ی قبل شروع می‌کنند و مجددا بهبود می‌دهد.



شکل ۱۶ نمودار میزان اتلاف

برای ارزیابی سیستم به داده آزمایش بایاس را می‌فزاییم و با استفاده از وزن‌های بروزرسانی شده در گام آخر آموزش داده آزمایش را ارزیابی می‌کنیم. در جدول زیر میزان دقت داده تست را نشان داده‌ایم.

```
x_test = np.hstack((np.ones((len(x_test_n), 1)), x_test_n))
y_pred = logistic_regression(x_test, w)
```

جدول ۴ ارزیابی داده آزمایش

Accuaracy	1
Precision	1

۲-۴-بخش د

در اینجا فرآیند آموزش را با استفاده از یک طبقه بندی خطی آماده پایتون در sklearn.linear-model پیاده‌سازی می‌نماییم. در ابتدا یک شی از روش آموزش خود ایجاد و مقادیر اولیه را مقداردهی می‌کنیم.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='sag', max_iter=200, random_state=14)
```

سپس از دستور زیر برای آموزش استفاده می‌کنیم.

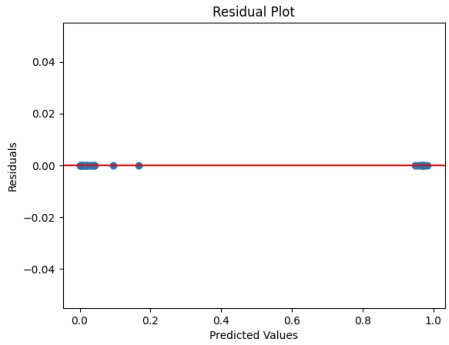
```
model.fit(x_train_n, y_train)
```

با استفاده از دستور زیر سیستم را ارزیابی می‌کنیم.

```
y_pred_s = model.predict(x_test_n)
```

میزان اتلاف را به کمک زیر می‌توانیم نشان دهیم.

```
residuals = y_test[:,0] - y_pred_s
plt.scatter(y_pred, residuals)
```



شکل ۱۷ ارزیابی سیتم با استفاده از کتابخانه آماده

۳–سوال سوم

در ابتدا فایل دیتاست مورد نظر را دانلود می‌کنیم. سپس یک فایل را برای ذخیره سازی دیتاست ایجاد می‌کنیم. از روش سوم ذکر شده در فیلم‌ها، با استفاده از دستور زیر فایل دیتاست را در گوگل کولب قرار می‌دهیم. ID مربوط به اشتراک فایل قرار گرفته در گوگل درایو را در اینجا قرار می‌دهیم.

```
!pip install --upgrade --no-cache-dir gdown
!gdown 18V-q5dHZXseCcj3QnF1BMwqjUtvhFU9u
```

سپس دیتاست آپلود شده را بارگذاری می‌کنیم، آدرس آن را با کلیک راست بر روی آن و انتخاب Copy path می‌یابیم. ابعاد آن به صورت (96453 12) می‌باشد.

```
df = pd.read_csv('/content/Data/weatherHistory.csv')
```

عنوان هر یک از ستون‌ها را با دستور زیر نمایش می‌دهیم.

```
df.columns
```

اطلاعات هر کدام از ستون‌ها را با دستور زیر نمایش می‌دهیم.

```
df.describe()
```

برای بررسی اینکه آیا در بین داده‌های قرار داده شده در یک ستون داده null وجود دارد یا خیر از دستور زیر استفاده می‌کنیم. همانطور که از نتایج مشخص است ویژگی Precip Type دارای داده null می‌باشد.

```
print(df.isnull().sum())
```

برای اینکه داده‌های null را حذف کنیم از دستور زیر استفاده می‌کنیم و ابعاد داده‌ها به صورت (95936 12) کاهش می‌یابد.

```
data_df = df.dropna(subset=["Precip Type"])
```

با استفاده از دستور زیر نیز نوع هر کدام از داده‌های ستون‌ها و تعداد داده‌های null آن را نیز می‌توانیم نمایش دهیم.

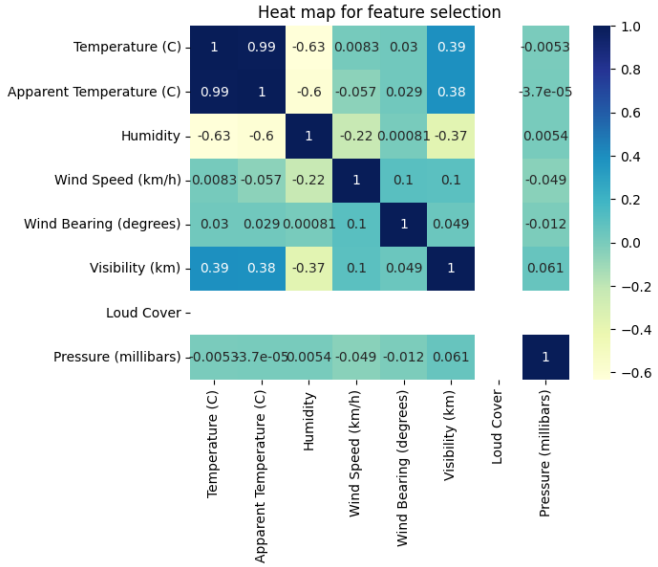
```
data_df.info()
```

۳-۱-بخش الف

در این بخش در ابتدا میزان همبستگی هر یک از ستون‌ها را محاسبه می‌کنیم. سپس میزان همبستگی محاسبه شده را به کمک دستور زیر به کمک نمودار هیت مپ ماتریس همبستگی نشان می‌دهیم. برای رسم آن ماژول seaborn را می‌افزاییم.

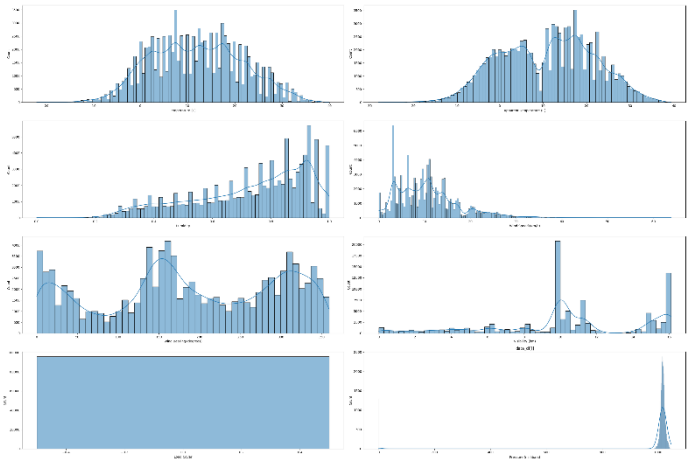
```
sns.heatmap(data_df.corr(), annot=True, cmap="YlGnBu")
```

همانطور که در نتایج مشخص شده است هر چقدر رنگ سلول پررنگتر باشد به این منظور است که میزان همبستگی ماکزیمم است. درایه‌های روی قطر اصلی چنین است زیرا میزان همبستگی هر داده با خودش زیاد است. و هرچقدر رنگ سلول کمتر باشد میزان همبستگی کمتر می‌باشد.



شکل ۱۸ نمودار هیت مپ تمام ویژگی‌ها

روش عالی برای شروع بررسی یک متغیر خاص، استفاده از هیستوگرام یا بافت‌نگار است. هیستوگرام متغیر را به دسته‌هایی تقسیم می‌کند، نقاط داده‌ای را در هر دسته می‌شمارد و دسته‌ها را روی محور x نمایش داده و تعداد نقاط را روی محور y نشان می‌دهد. در ابتدا داده‌های نوع عددی را مشخص می‌کنیم. سپس برای آن دسته از داده‌ها نمودار هیستوگرام را رسم می‌کنیم.



شکل ۱۹ نمودار هیستوگرام تمام ویژگی‌ها

```
sns.histplot(data_df[i],ax =subplot,kde = True)
```

همانطور که در نمودار هیت مپ نشان داده شده است میزان همبستگی ویژگی‌های Temperature (C) و Apparent Temperature (C) با Humidity کوچک است. در نتیجه انتظار می‌رود که با کمک این ورودی نتوان به خوبی خروجی‌ها را تخمین بزنیم.

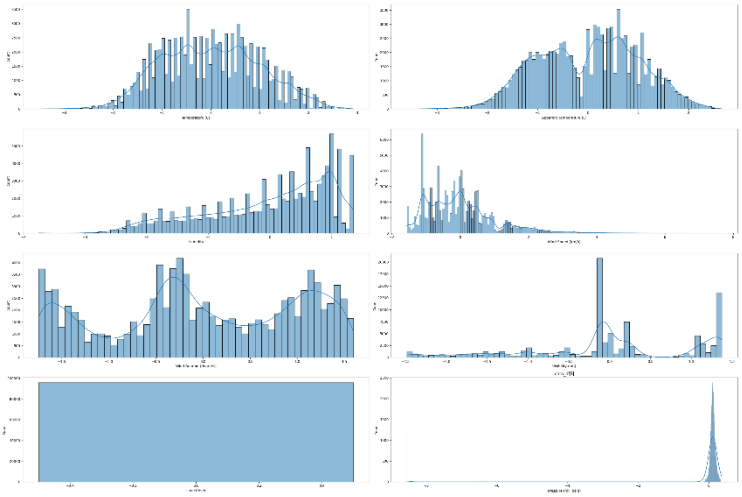
نمودار هیت مپ نشان می‌دهدکه Temperature (C) را با کمک Apparent Temperature (C) به خوبی تخمین بزنیم از آنجایی که میزان همبستگی دو ویژگی زیاد است.

۳-۲-بخش ب

۳-۲-۱- روش LS

در ابتدا تمام ویژگی‌های دیتاست را با استفاده از StandardScaler پیش پرداز انجام می‌دهیم؛ با حذف میانگین و مقیاس بندی به واریانس واحد، ویژگی ها را استاندارد می‌کنیم و مجددا نمودار هیستوگرام مربوط به ویژگی‌ها را رسم می‌کنیم.

```
scaled_data_df=pd.DataFrame(StandardScaler().fit_transform(df1),columns=df1.columns)
```



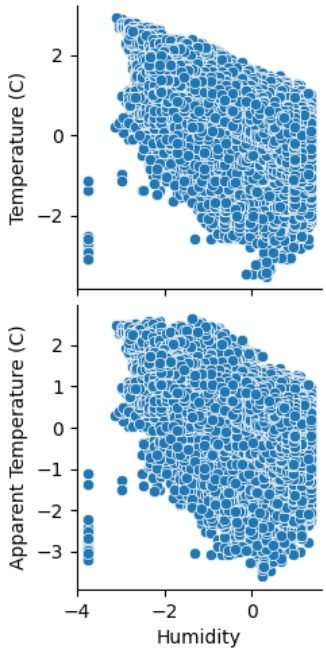
شکل ۲۰ نمودار هیستوگرام ویژگی‌های پردازش شده

با در نظر گرفتن Humidity به عنوان ورودی، Temperature (C) و Apparent Temperature (C) به عنوان خروجی مسئله را ادامه می‌دهیم.

```
X = scaled_data_df[['Humidity']]
y1 = scaled_data_df[['Temperature (C)']]
y2 = scaled_data_df[['Apparent Temperature (C)']]
```

با استفاده از دستور زیر خروجی‌های در نظر گرفته شده را براساس ورودی رسم می‌نماییم.

```
sns.pairplot( scaled_data_df, x_vars = ['Humidity'],
        y_vars = ['Temperature (C)','Apparent Temperature (C)'])
```



شکل ۲۱ رسم خروجی‌ها براساس ورودی

سپس شروع به نوشتن کد کلاس روش Leas Square(LS) می‌کنیم که یکی از روش‌های مربوط به Regression می‌باشد. در واقع ما یکسری ورودی و خروجی داریم و سعی در یافتن یک مدل برای سیستم بر اساس آن‌ها می‌باشیم.

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

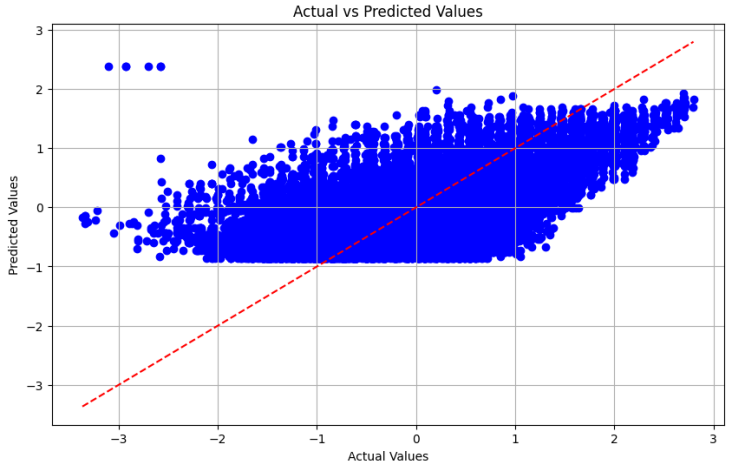
هدفمان در اینجا یافتن $\hat{\theta}$ به صورتی که به θ نزدیک باشد و مسئله overfitting نداشته باشیم. زیرا در صورتی که داده‌های ما به صورت نویزی باشد، سیستم ما نویز را نیز یادگرفته می‌باشد، که مطلوب نیست. تابع هزینه در نظر گرفته شده در اینجا به صورت MSE می‌باشد.

$$Loss = \frac{1}{n}\sum\left(yi - y^i\right)^2$$

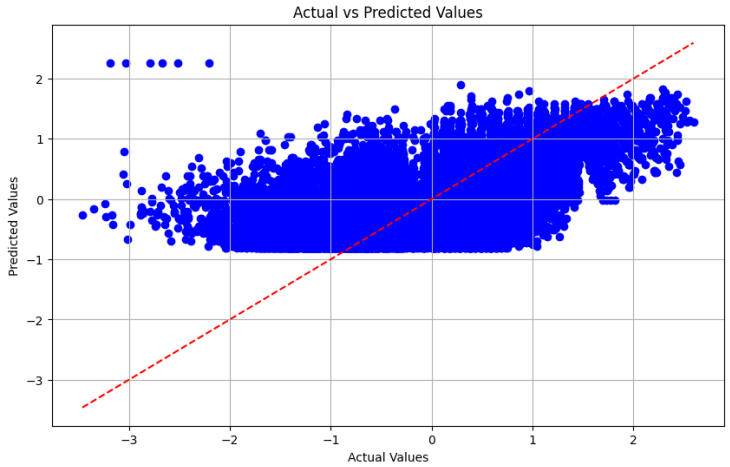
در اینجا ما از بهینه‌ساز مانند گرادیان نزولی استفاده نکرده‌ایم (در بخش LS با این بخش کاری نداریم) و سعی در تخمین مدل مناسب می‌باشیم.

داده‌های آموزش و تست با استفاده از دستور زیر ایجاد می‌کنیم. میزان آرگومان random_state را برابر دو رقم آخر شماره دانشجویی قرار می‌دهیم.

```
X_train1, X_test1, y_train1, y_test1=train_test_split(X, y1,test_size=0.2, random_state=76)
```



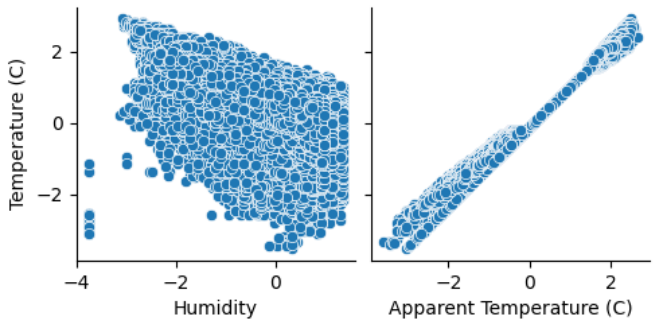
شکل ۲۲ روش Ls–خروجی اول



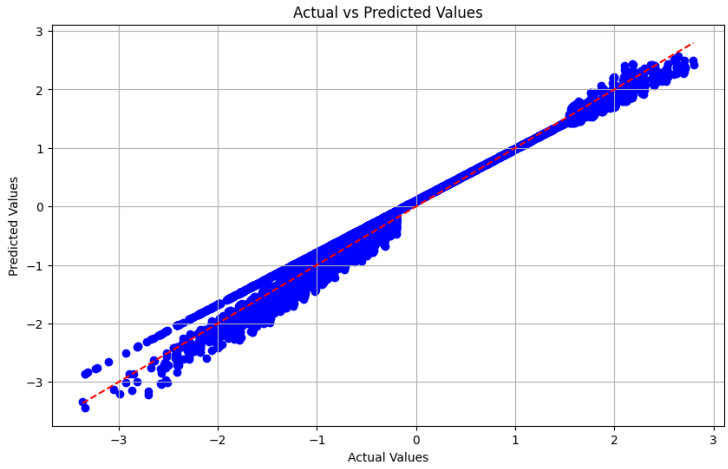
شکل ۲۳ روش Ls–خروجی دوم

براساس نمودار هیت مپ تصمیم بر این گرفتیم که ورودی را Apparent Temperature (C) و خروجی را Temperature (C) در نظر بگیریم و سیستم را تخمین بزنیم.

```
X_n = scaled_data_df[['Apparent Temperature (C)']]
y_n = scaled_data_df[['Temperature (C)']]
```



شکل ۲۴رسم خروجی براساس ورودی‌ها



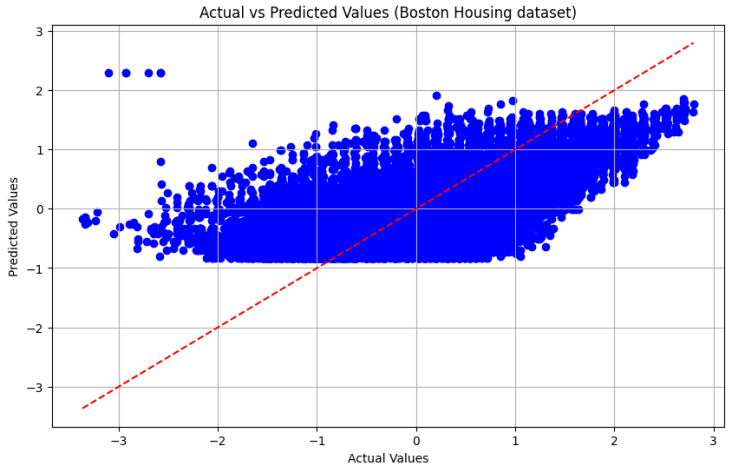
شکل ۲۵ روش Ls-تغییر ورودی

۳-۲-۲-روش RLS

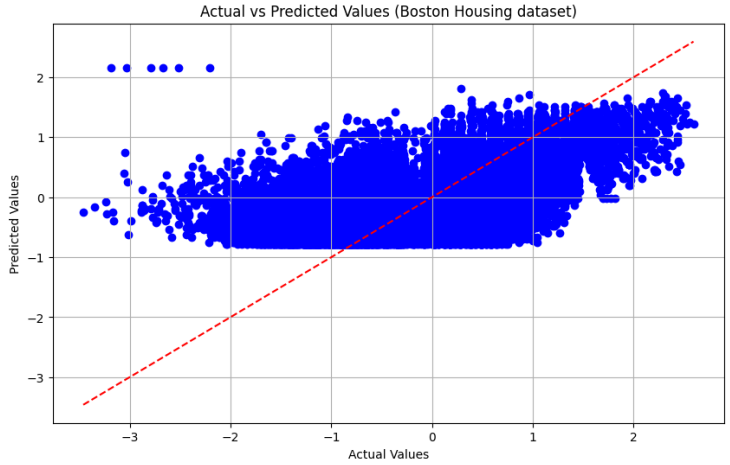
در حالت LS داده‌های ما به صورت آفلاین می‌باشد؛ ولی زمانی که شبکه به صورت آنلاین داده تولید می‌کند و ما می‌خواهیم از آن استفاده کنیم از Recursive Least Squares (RLS) استفاده می‌کنیم. مراحلی که در اینجا اتفاق می‌افتد به این صورت است که در ابتدا پارامترها مقادیر اولیه می‌گیرند، ردیابی می‌کند و سپس بروزرسانی. تفاوت این روش در این است که گام دوم و سوم برای داده‌های جدید تکرار می‌شود.

فاکتور فراموشی برای تمرکز بر روی داده‌های گذشته می‌باشد. هرچقدر بزرگتر باشد، داده‌های گذشته را زودتر فراموش می‌کند.

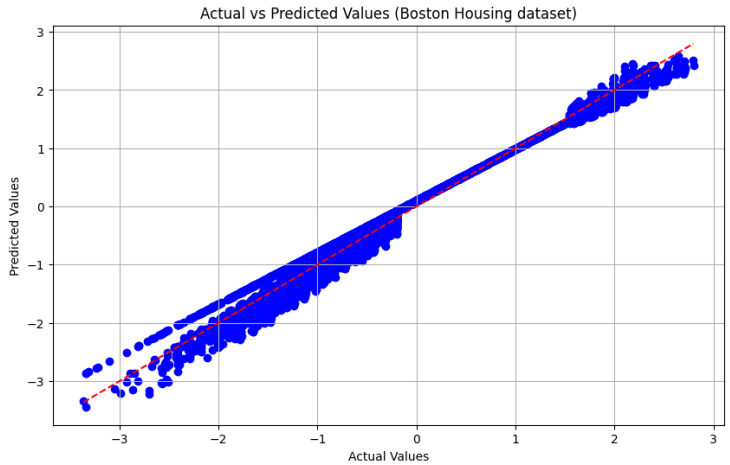
با استفاده از کتابخانه statsmodels.api نیز می‌توانیم کد بزنیم. در اینجا نتایج مربوط به کد دستی قرار دارد ولی هر دو روش کدزنی استفاده شده است.



شکل ۲۶ روش RLS–خروجی اول



شکل ۲۷ روش RLS–خروجی دوم



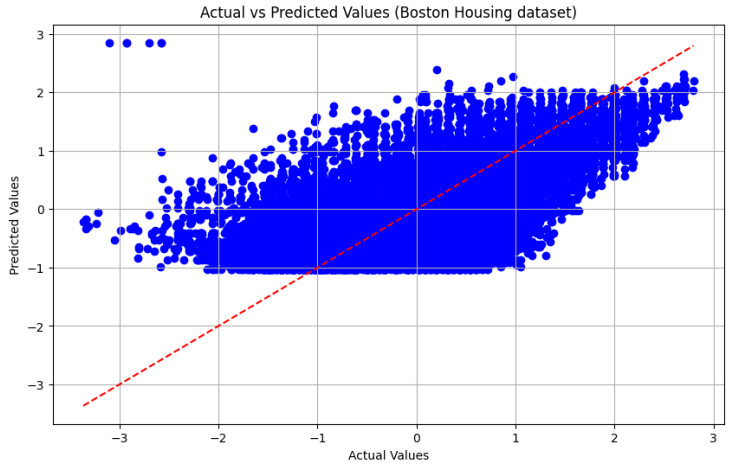
شکل ۲۸ روش RLS–تغییر ورودی

۳-۳-بخش ج

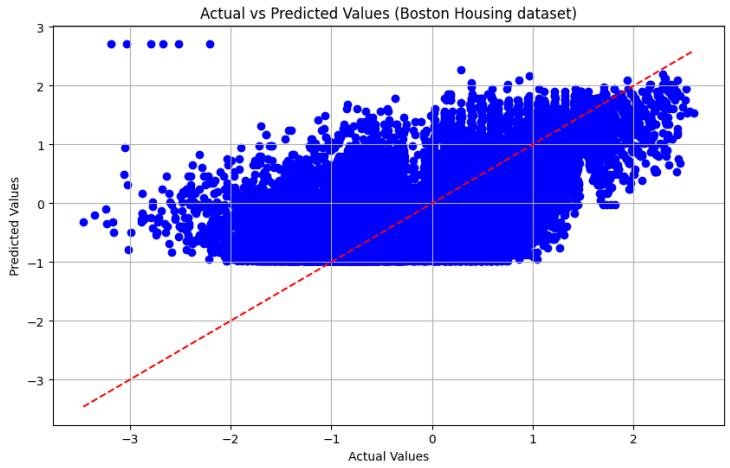
گاهی اوقات لازم است برای داده‌ها و خطاها وزن‌گذاری بکنیم (به عنوان مثال در پردازش تصویر). تفاوت Weighted Least Square با Least Square در رابطه زیر است.

$$Q = \begin{bmatrix} Q_{11} & 0 & \dots & 0 \\ 0 & Q_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & Q_{NN} \end{bmatrix}, Q = \frac{1}{var(y)}, \underline{\hat{\theta}} = \left(\underline{X}^T \underline{Q} \underline{X}\right)^{-1} \underline{X} \underline{y}$$

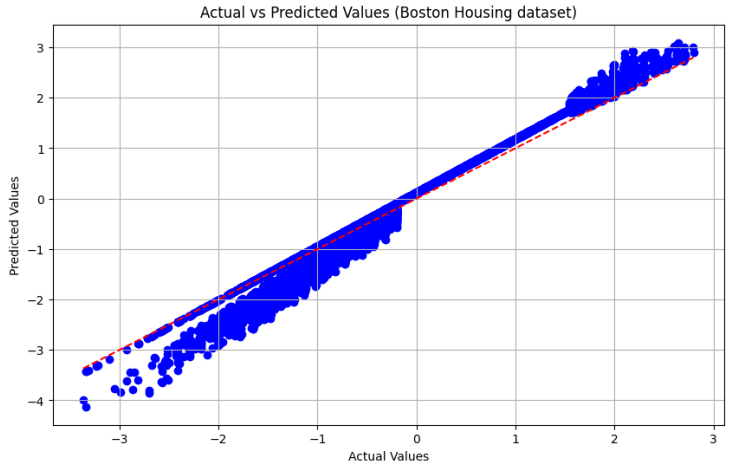
عناصر روی قطر اصلی Q می‌توانند یکسان و یا متفاوت باشند، که ما در اینجا یکسان در نظر می‌گیریم. با استفاده از کتابخانه statsmodels.api نیز می‌توانیم کد بزنیم. در اینجا نتایج مربوط به کد دستی قرار دارد ولی هر دو روش کدزنی استفاده شده است.



شکل ۲۹ روش WLS-خروجی اول



شکل ۳۰ روش WLS-خروجی دوم



شکل ۳۱ روش WLS-تغییر ورودی

در جدول زیر میزان MSE هر سه روش ذکر شده است.

جدول ۵ مقایسه مقادیر هر یک از روش‌ها

	ورودی / خروجی	Humidity	Apparent Temperature
Least Square	Temperature	0.5969746762657372	0.014550000406755059
	Apparent Temperature	0.6375466247376951	-
Recuresive Leas Square	Temperature	0.602827200423407	0.014805461127825596
	Apparent Temperature	0.6385307267271296	-
Weighted Least Square	Temperature	0.613759052252272	0.055089702256967854
	Apparent Temperature	0.652621156329484	-

۳-۴-بخش د

استفاده از تجزیه QR برای مثلثی کردن ماتریس داده ورودی منجر به یک روش جایگزین برای اجرای روش حداقل مربعات بازگشتی (RLS) می‌شود. مزایای اصلی ناشی از الگوریتم حداقل مربعات بازگشتی مبتنی بر تجزیه QR، اجرای احتمالی آن در آرایه‌های سیستولی و بهبود رفتار عددی آن با در نظر گرفتن اثرات کوانتیزاسیون است. الگوریتم‌های RLS پیشنهادی مبتنی بر تجزیه QR بر روی مثلثی‌سازی ماتریس اطلاعات تمرکز دارد تا از استفاده از وارونگی ماتریس جلوگیری شود. با این حال، نیاز محاسباتی آنها ضرب $O[N^2]$ در هر نمونه خروجی می‌باشد. در شکل زیر مراحل این الگوریتم را می‌توانیم مشاهده کنیم[۱].

Algorithm 9.1
QR-RLS Algorithm

$\mathbf{w}(-1) = [0\,0\ldots 0]^T, \quad w_0(0) = \frac{d(0)}{x(0)}$

For $k = 1$ to N (Initialization)

Do for $i = 1$ to k

$$w_i(k) = \frac{-\sum_{j=1}^{i} x(j) w_{i-j}(k) + d(i)}{x(0)} \tag{9.11}$$

End

End

$$\mathbf{U}'_0(N+1) = \lambda^{1/2}\underline{\mathbf{X}}(N) \tag{9.12}$$

$$\mathbf{d}'_{q_{2\,0}}(N+1) = [\,\lambda^{1/2}d(N)\;\lambda d(N-1)\ldots\lambda^{(N+1)/2}d(0)]^T$$

For $k \geq N+1$

Do for each k

$\gamma'_{-1} = 1$

$d'_0(k) = d(k)$

$\mathbf{x}'_0(k) = \mathbf{x}^T(k)$

Do for $i = 0$ to N

$$c_i = \sqrt{[\mathbf{U}'_i(k)]^2_{i+1,N+1-i} + x'^2_i(k-N-i)}$$

$$\cos\theta_i = \frac{[\mathbf{U}'_i(k)]_{i+1,N+1-i}}{c_i} \tag{9.41}$$

$$\sin\theta_i = \frac{x'_i(k-N-i)}{c_i} \tag{9.42}$$

$$\begin{bmatrix} \mathbf{x}'_{i+1}(k) \\ \mathbf{U}'_{i+1}(k) \end{bmatrix} = \mathbf{Q}'_{\theta_i}(k) \begin{bmatrix} \mathbf{x}'_i(k) \\ \mathbf{U}'_i(k) \end{bmatrix} \tag{9.39}$$

$$\gamma'_i = \gamma'_{i-1}\cos\theta_i \tag{9.54}$$

$$\begin{bmatrix} d'_{i+1}(k) \\ \mathbf{d}'_{q_{2i+1}}(k) \end{bmatrix} = \mathbf{Q}'_{\theta_i}(k) \begin{bmatrix} d'_i(k) \\ \mathbf{d}'_{q_{2i}}(k) \end{bmatrix} \tag{9.51}$$

End

$$\mathbf{d}'_{q_{20}}(k+1) = \lambda^{1/2}\mathbf{d}'_{q_{2N+1}}(k)$$

$$\mathbf{U}'_0(k+1) = \lambda^{1/2}\mathbf{U}'_{N+1}(k)$$

$$\gamma(k) = \gamma'_N$$

$$\varepsilon(k) = d'_{N+1}(k)\gamma(k) \tag{9.51}$$

If required compute

$$\underline{\mathbf{d}}(k) = \begin{bmatrix} d'_{N+1}(k) \\ \mathbf{d}'_{q_{2N+1}}(k) \end{bmatrix} \tag{9.51}$$

$$w_0(k) = \frac{\underline{d}_{N+2}(k)}{u_{N+1,1}(k)}$$

Do for $i = 1$ to N

$$w_i(k) = \frac{-\sum_{j=1}^{i} u_{N+1-i,i-j+1}(k) w_{i-j}(k) + \underline{d}_{N+2-i}(k)}{u_{N+1-i,i+1}(k)} \tag{9.46}$$

End

End

شکل ۳۲ الگوریتم QR-Decomposition-Based RLS

منابع

[۱] P. S. Diniz and P. S. Diniz, "QR-decomposition-based RLS filters," *Adaptive Filtering: Algorithms and Practical Implementation*, pp. 367-409, 2013.