به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

گزارش تمرین شماره ۱

شیما سادات ناصری

۴۰۱۱۲۸۱۴

دکتر مهدی علیاری شوره دلی

فروردین ۱۴۰۳

فهرست مطالب

عنوان شماره صفحه

چکیده

تمامی کدها به همراه همین گزارش در برنچ زیر موجود است:

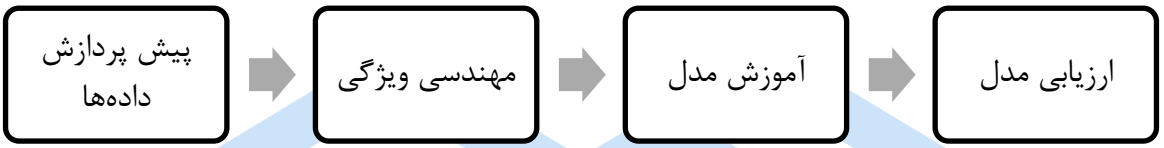
https://github.com/shimanaseri/ML-coarse/tree/Mini-Project-1

ریپو به صورت پرایوت در آمده است.

سوال ۱

۱

بلوک دیاگرام فرایند به صورت زیر می‌تواند باشد:



هر بخش از این بلوک دیاگرام به صورت زیر است:

۱. پیش پردازش داده ها: این مرحله شامل تمیز کردن و آماده سازی داده های خام برای آموزش است. ممکن است شامل کارهایی مانند مدیریت مقادیر گمشده، مقیاس‌بندی ویژگی‌ها، رمزگذاری متغیرهای طبقه‌بندی، و تقسیم داده‌ها به مجموعه‌های آموزشی و آزمایشی باشد.

۲. مهندسی ویژگی: مهندسی ویژگی شامل تبدیل ویژگی های خام به قالبی است که مدل بتواند بهتر آن را درک کند. این بخش می‌تواند شامل مقیاس‌بندی داده‌ها باشد تا همه آن‌ها در یک مقیاس باشند، ویژگی‌های موجود با هم ترکیب شود تا ویژگی‌های جدید ایجاد شده، یا تعداد ویژگی‌ها را کاهش داده تا روی مهم‌ترین آنها تمرکز شود.

۳. آموزش مدل: در این مرحله مدل طبقه بندی خطی بر روی داده های از پیش پردازش شده آموزش داده می شود. آموزش شامل یادگیری پارامترهای مدلی است که به بهترین وجه با داده های آموزشی مطابقت دارد. برای طبقه‌بندی خطی، این معمولاً به معنای یافتن بهترین خط یا مرز برای جداسازی کلاس‌های مختلف است.

۴. ارزیابی مدل: هنگامی که مدل آموزش داده شد، با استفاده از مجموعه داده جداگانه ای که در طول آموزش استفاده نشده است (مجموعه آزمون) ارزیابی می شود. معیارهای ارزیابی مانند دقت، loss و امتیاز F1 برای ارزیابی عملکرد مدل بر روی داده‌های دیده نشده محاسبه می‌شوند.

تغییر از طبقه بندی دو کلاسه به چند کلاسه:

تغییر از طبقه بندی دو کلاسه به چند طبقه عمدتاً بر بخش های آموزش و ارزیابی تأثیر می گذارد:

آموزش: در طبقه بندی دو کلاسه، مدل یاد می گیرد که بین دو کلاس تمایز قائل شود. در چند کلاس، یاد می گیرد که بین بیش از دو کلاس تمایز قائل شود. بنابراین، ساختار مدل و روشی که از داده‌ها یاد می‌گیرد باید برای مدیریت چندین کلاس تنظیم شود.

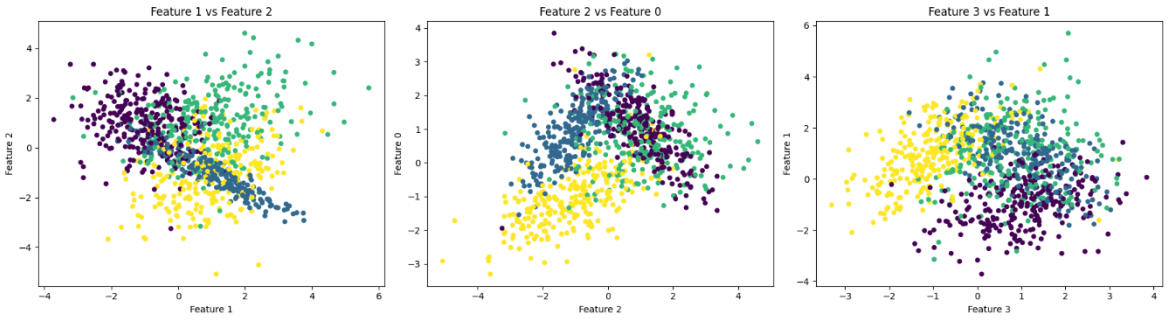
ارزیابی: هنگامی که ما یک مدل را در حالت دو کلاسه ارزیابی می کنیم، عمدتاً به این علاقه مندیم که چقدر خوب آن دو کلاس را پیش بینی می کند. در حالت چند کلاسه، باید ببینیم که چگونه هر یک از کلاس های چندگانه را پیش بینی می کند. این ممکن است به معنای استفاده از روش‌ها و معیارهای ارزیابی مختلف باشد که برای سناریوهای چند طبقه طراحی شده‌اند.

۲

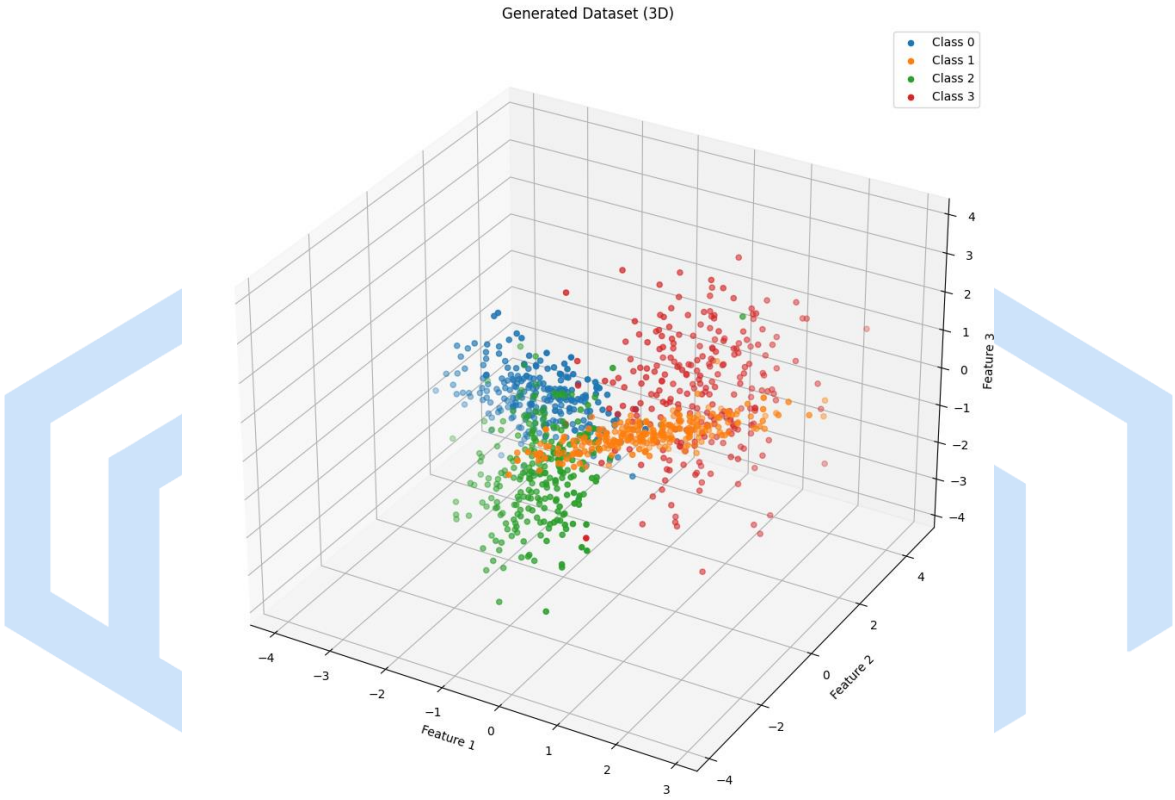
کد این قسمت از سوال به صورت زیر می‌باشد:

```python
# Generate dataset


from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
import numpy as np

# Generate a dataset with 1000 samples, 4 classes, and 3 features
X, y = make_classification(n_samples=1000, n_features=3, n_informative=3, n_redundant=0, n_classes=4, n_clusters_per_class=1, random_state=14)

"""Plotting"""

# Visualize the dataset
fig, axs = plt.subplots(1, 3, figsize=(18, 5))

for i in range(3):
    # Pick one feature as x and another as y for plotting, skipping the diagonal
    axs[i].scatter(X[:, i], X[:, (i+1)%3], c=y, cmap='viridis', s=20)
    axs[i].set_title(f'Feature {i+1} vs Feature {(i+2)%3}')
    axs[i].set_xlabel(f'Feature {i+1}')
    axs[i].set_ylabel(f'Feature {(i+2)%3}')

plt.tight_layout()
plt.show()

from mpl_toolkits.mplot3d import Axes3D

# Visualizing the dataset in 3D
fig = plt.figure(figsize=(16, 12))
ax = fig.add_subplot(111, projection='3d')

for i in range(4):
    ax.scatter(X[y == i][:, 0], X[y == i][:, 1], X[y == i][:, 2], label=f'Class {i}')

ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.set_title('Generated Dataset (3D)')
ax.legend()
plt.show()
```

بر این اساس خروجی‌های زیر بدست آمدند.



شکل ۱- نمایش دو بعدی داده‌ها



شکل ۲- نمایش سه بعدی داده‌ها

حال اگر بخواهیم داده پیچیده‌تری تولید کنیم، می‌توان روش‌های زیر را به کار گرفت:

۱. افزایش تعداد کلاس‌ها: افزودن کلاس‌های بیشتر می‌تواند کار طبقه‌بندی را سخت‌تر کند، به خصوص اگر تعداد نمونه‌ها در هر کلاس ثابت نگه داشته شود.

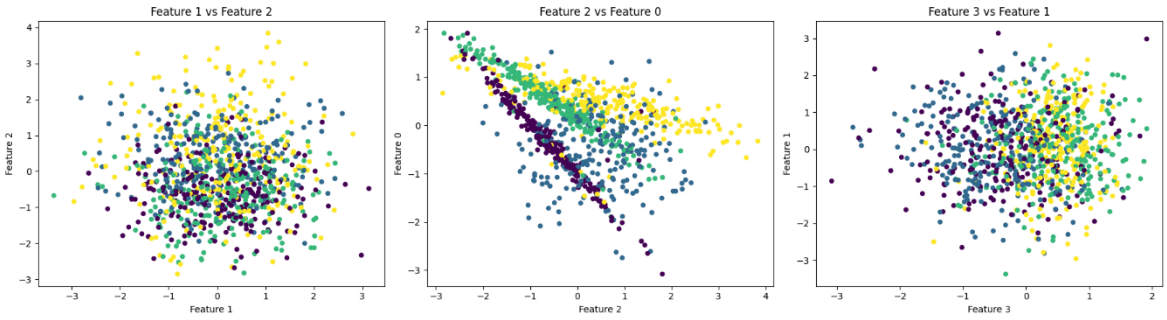
۲. معرفی نویز: افزودن نویز به مقادیر ویژگی می‌تواند مرزهای طبقه‌بندی را کمتر واضح کند، بنابراین دشواری طبقه‌بندی صحیح هر نمونه را افزایش می‌دهد.

۳. افزایش همپوشانی کلاس: کاهش پارامتر n_clusters_per_class به کمتر از ۱ یا تنظیم پارامتر جداسازی کلاس می تواند همپوشانی بین کلاس ها را در فضای ویژگی افزایش دهد و تشخیص بین آنها را دشوارتر کند.
۴. افزودن ویژگی‌های اضافی و/یا غیر اطلاعاتی: افزایش تعداد ویژگی‌ها در حالی که فقط چند مورد از آن‌ها را آموزنده نگه می‌دارد (با تنظیم پارامترهای n_informative و n_redundant) می‌تواند پیچیدگی بیشتری ایجاد کند.
۵. تغییر تعداد نمونه‌ها را در هر کلاس: ایجاد یک مجموعه داده نامتعادل، که در آن برخی از کلاس‌ها نمونه‌های بسیار بیشتری نسبت به سایرین دارند، می‌تواند چالش‌هایی را در یادگیری طبقه‌بندی مؤثر کلاس‌های اقلیت ایجاد کند.

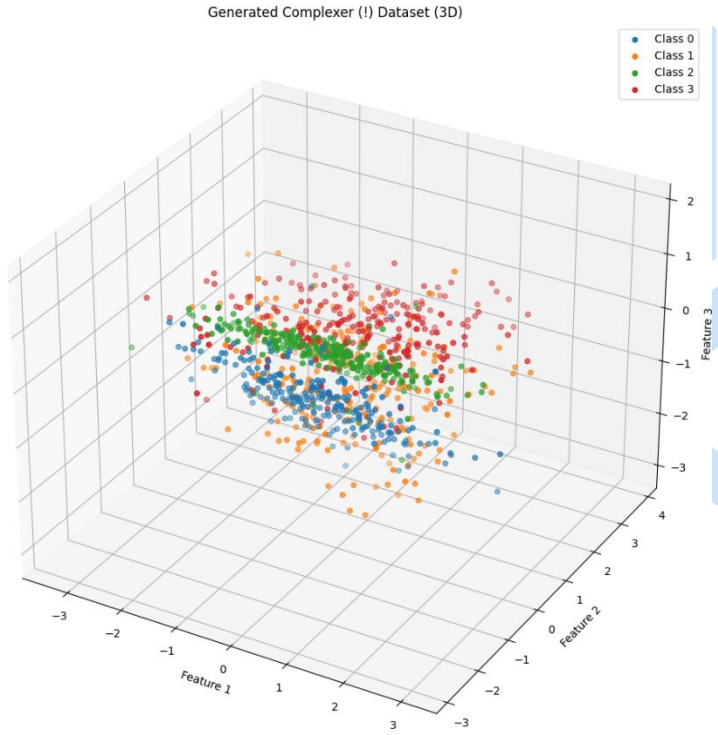
البته اگر محدودیت برای تغییر ندادن تعداد کلاس‌ها و ویژگی‌ها داشته باشیم تمامی موارد بالا قاب اجرا نخواهند بود. با همین فرض سعی شد تا دیتاست پیچیده‌تری ایجاد شود که کد آن به صورت زیر می‌باشد:

```
"""## Generate a more complex dataset with 1000 samples, 4 classes, and 3 features"""

X_2, y_2 = make_classification(n_samples=1000, n_features=3, n_classes=4, n_informative=2, n_redundant=0, n_clusters_per_class=1, flip_y=0.05, class_sep=0.5, random_state=14)

# Visualize the dataset
fig, axs = plt.subplots(1, 3, figsize=(18, 5))

for i in range(3):
    # Pick one feature as x and another as y for plotting, skipping the diagonal
    axs[i].scatter(X_2[:, i], X_2[:, (i+1)%3], c=y_2, cmap='viridis', s=20)
    axs[i].set_title(f'Feature {i+1} vs Feature {(i+2)%3}')
    axs[i].set_xlabel(f'Feature {i+1}')
    axs[i].set_ylabel(f'Feature {(i+2)%3}')

plt.tight_layout()
plt.show()

from mpl_toolkits.mplot3d import Axes3D

# Visualizing the dataset in 3D
fig = plt.figure(figsize=(16, 12))
ax = fig.add_subplot(111, projection='3d')

for i in range(4):
    ax.scatter(X_2[y_2 == i][:, 0], X_2[y_2 == i][:, 1], X_2[y_2 == i][:, 2], label=f'Class {i}')

ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.set_title('Generated Complexer (!) Dataset (3D)')
ax.legend()
plt.show()
```

مجموعه داده با معرفی نویز برچسب(label noise) و کاهش تفکیک طبقات(class separation) چالش برانگیزتر شده است، در حالی که به محدودیتی که افزایش خوشه‌ها در هر کلاس را به دلیل تعداد ویژگی‌های اطلاعاتی محدود می‌کند، پایبند است. خروجی آن نیز به صورت زیر می‌باشد.



شکل ۳- نمایش داده‌های پیچیدتر در فضای دو بعدی



شکل ۴- نمایش داده‌های پیچیده‌تر در فضای سه بعدی

هرچند که در نمای ۲ بعدی داده، داده پیچیده‌تر بنظر ساده‌تر جدا می‌شود، اما در کل شکل داده درهم تنیدگی بیشتری به خود گرفته‌است.

۳ و ۴

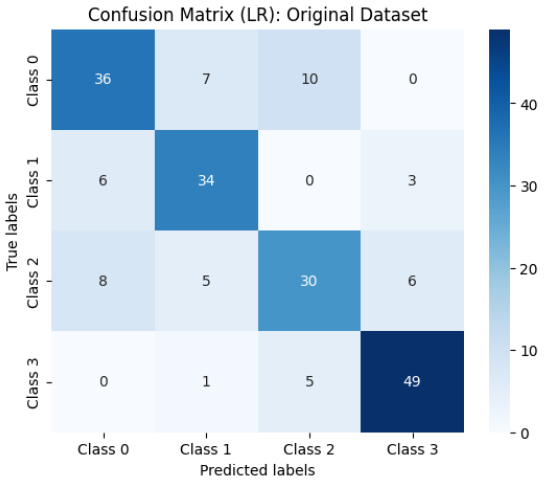
از طبقه بندهای زیر استفاده شده است:

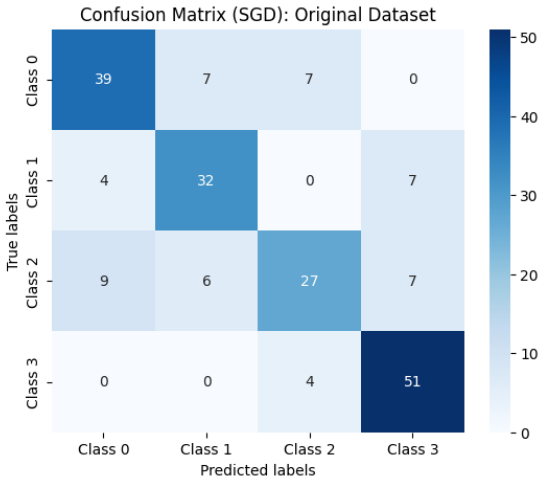
رگرسیون لجستیک(Logistic Regression): می تواند طبقه بندی چند طبقه را با استفاده از تکنیک هایی مانند "یک در مقابل استراحت" (OvR) یا رگرسیون لجستیک چند جمله ای انجام دهد.

SGDClassifier: یک طبقه بندی کننده خطی (SVM، رگرسیون لجستیک و غیره) با آموزش شیب نزولی تصادفی (SGD). به دلیل قابلیت تنظیم برای مدل های خطی مختلف با تنظیم پارامتر تلفات، کارایی عمومی دارد.

خروجی مدل ها به صورت زیر می‌باشند:

```
Log regression accuracy: 0.745
SGD accuracy: 0.745
```

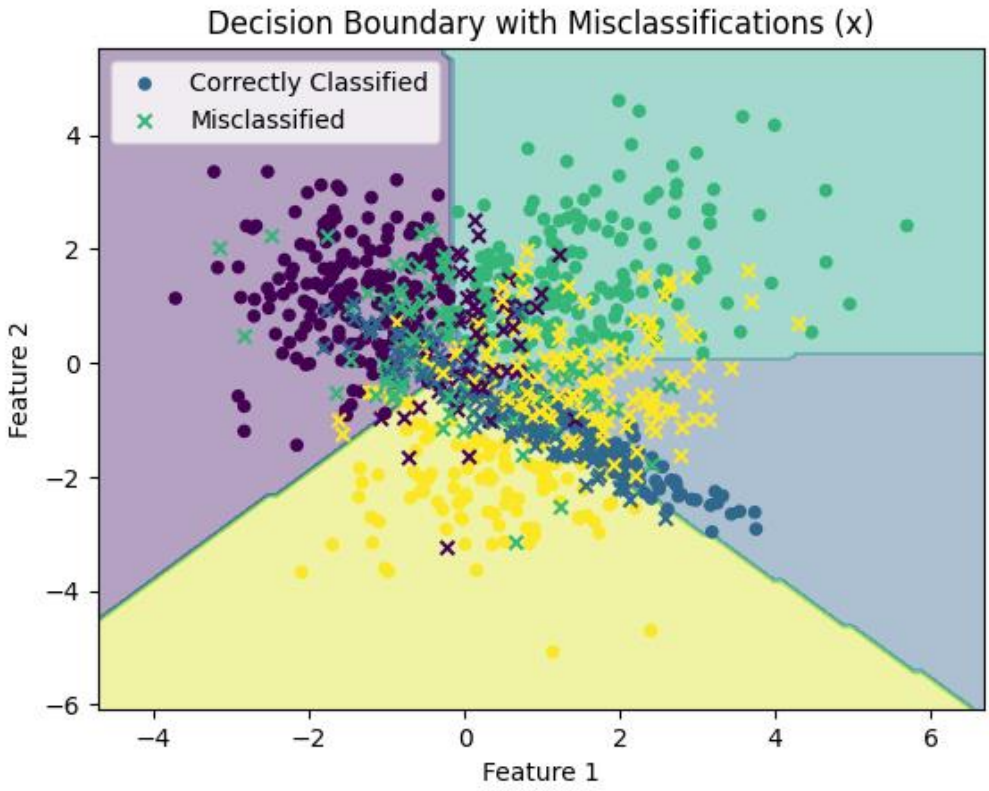




خطوط مرزی و داده‌هایی که درست طبقه بندی نشده‌اند، نیز به کمک کد زیر رسم می‌شوند. در این کد داده‌های اشتباه با x نمایش داده می‌شوند:

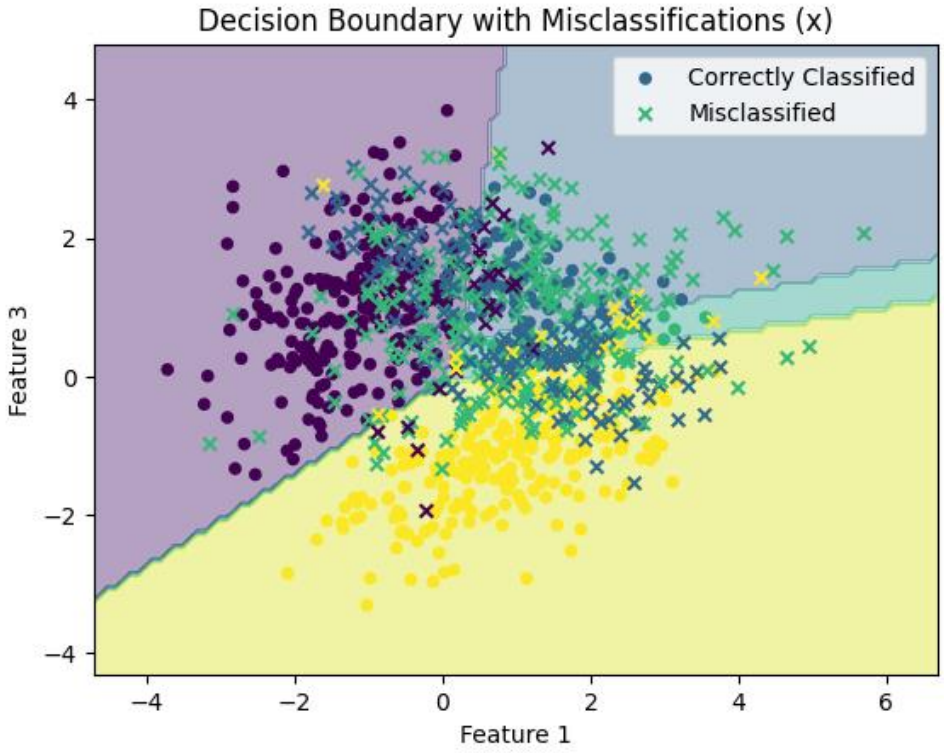
```python
def plot_decision_boundaries(X, y, classifier, feature_indices=(0, 1)):
    # Fit model to the specified features
    classifier.fit(X[:, feature_indices], y)

    # Define meshgrid for the background colors
    x_min, x_max = X[:, feature_indices[0]].min() - 1, X[:, feature_indices[0]].max() + 1
    y_min, y_max = X[:, feature_indices[1]].min() - 1, X[:, feature_indices[1]].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))

    # Predict on mesh to get the decision boundaries
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Predict on the original data points to find misclassifications
    y_pred = classifier.predict(X[:, feature_indices])
    misclassified = y != y_pred

    # Plot decision boundaries
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X[:, feature_indices[0]][~misclassified], X[:, feature_indices[1]][~misclassified], c=y[~misclassified], s=20, label='Correctly Classified')
    plt.scatter(X[:, feature_indices[0]][misclassified], X[:, feature_indices[1]][misclassified], c=y[misclassified], s=30, marker='x', label='Misclassified')
    plt.xlabel(f'Feature {feature_indices[0]+1}')
    plt.ylabel(f'Feature {feature_indices[1]+1}')
    plt.title('Decision Boundary with Misclassifications (x)')
    plt.legend()
```

خروجی این کد برای مدل اول به صورت زیر است:



شکل ۵- خطوط مرزی بدست آمده با روش **LR**

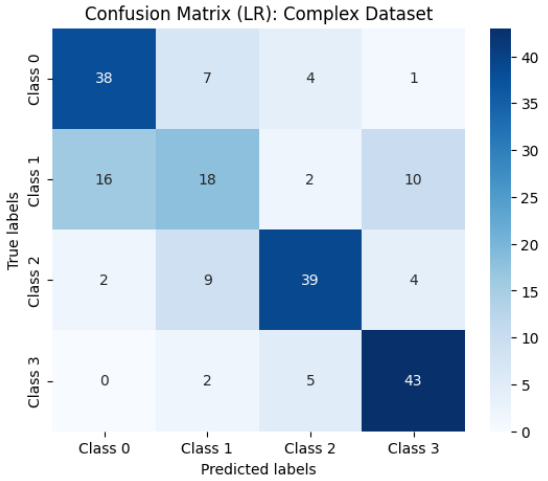
و برای مدل دوم به صورت زیر می‌باشد:

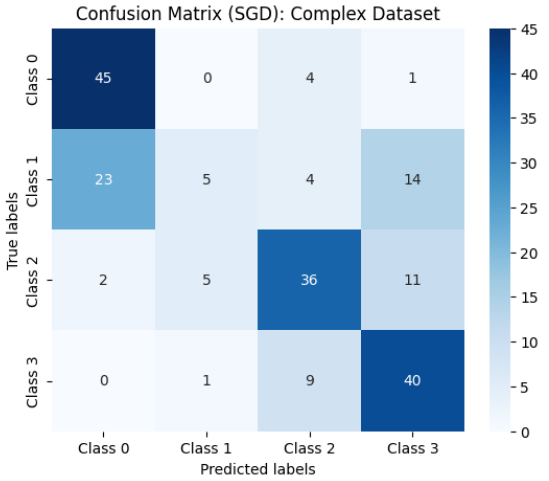


شکل ۶- خطوط مرزی بدست آمده با روش **SGD**

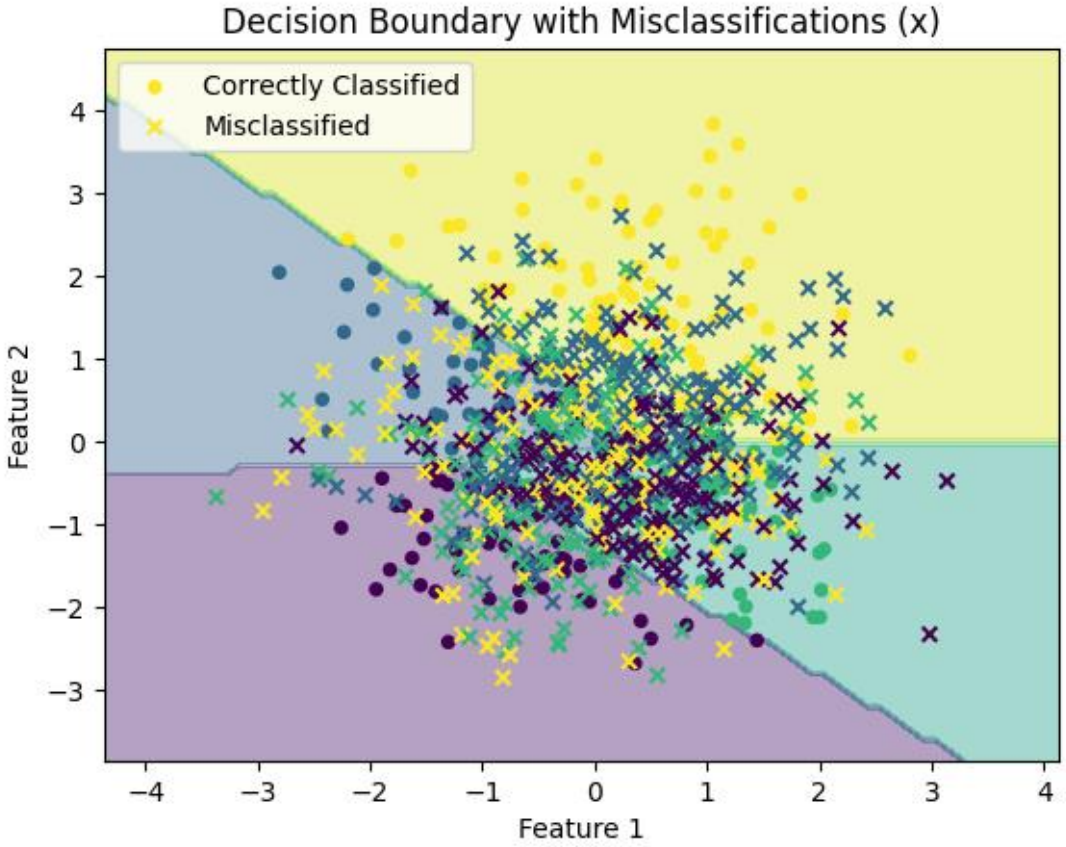
همین متد بر روی داده‌های پیچیده‌تر نیز اعمال شد:

```
Log regression accuracy: 0.69
SGD accuracy: 0.63
```

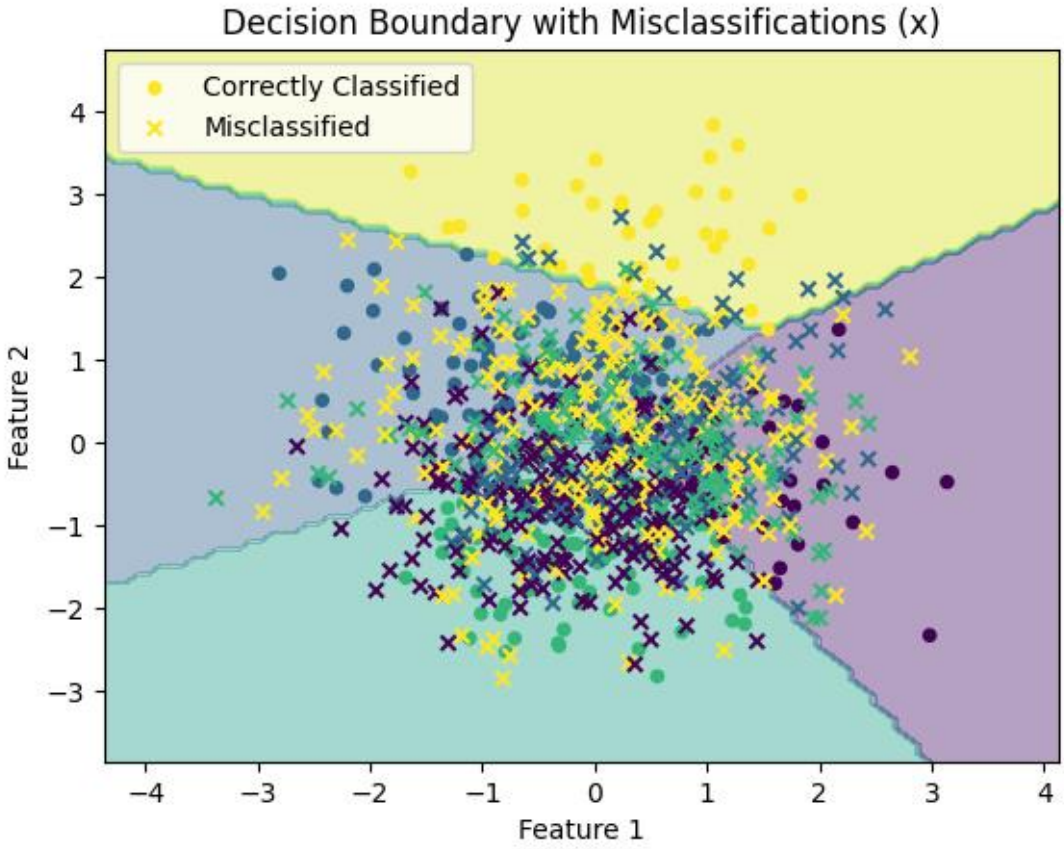




خطوط مرزی نیز به صورت زیر بدست آمدند.



شکل ۷- خطوط مرزی بدست آمده برای داده‌های پیچیده‌تر با روش **LR**



شکل ۸- خطوط مرزی بدست آمده برای داده‌های پیچیده‌تر با روش **SGD**

این نتایج نشان می‌دهد که هر دو طبقه‌بندی‌کننده روی مجموعه داده اصلی خوب عمل می‌کنند، و روش LR که کمی بهتر از SGDClassifier است. با این حال، در مجموعه داده‌های پیچیده‌تر، عملکرد هر دو مدل کاهش یافته، روش LR همچنان بهتر از SGDClassifier اما با تفاوت قابل‌توجهی در مقایسه با عملکرد آن در مجموعه داده اصلی عمل کرده‌است.

روش های مورد استفاده برای بهبود نتایج بعد از آزمون و خطا روی داده‌های ساده و پیچیده به صورت زیر بود:

به عنوان solver در روش LR از «saga» استفاده شده که از رگولاریزاسیون L1 و L2 پشتیبانی می‌کند و با رگرسیون لجستیک چند جمله‌ای به خوبی کار می‌کند.

برای تابع زیان در روش SGD از «modified_huber» استفاده شده که یک تابع ضرر نرم است که tolerance به مقادیر پرت و همچنین تخمین‌های احتمال اضافه می‌کند.

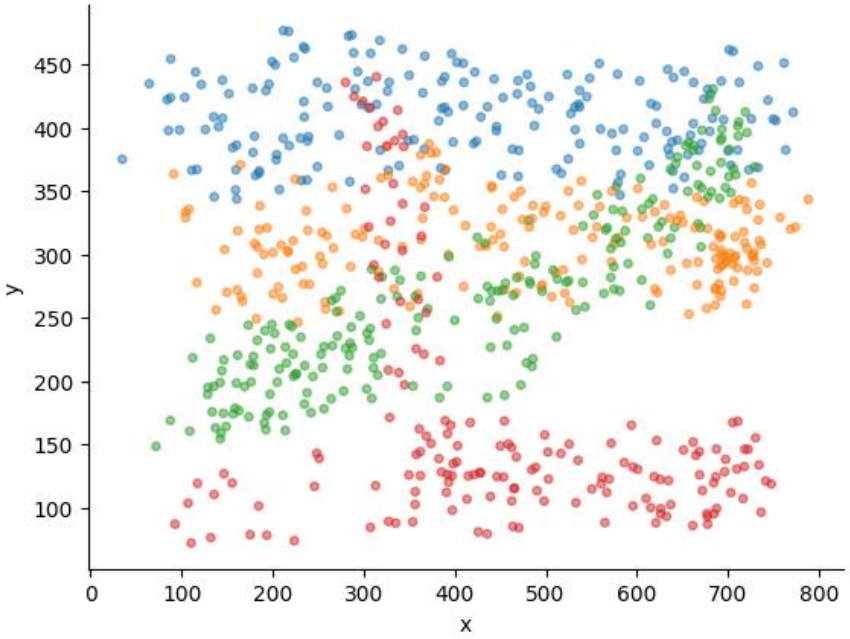
روش‌هایی مانند اسکیل کردن داده‌ها در این روش خصوصا روی داده‌های پیچیده کارساز نبودند.

۵

ابزار drawdata با کمک گیت هاب آن به صورت زیر استفاده شد:

```
from drawdata import ScatterWidget

widget = ScatterWidget()
widget
```

داده‌های کشیده شده پس از استخراج به صورت دیتافریم، به صورت زیر به نمایش در آمدند:



شکل ۹- داده‌های تولید شده با **drawdata**

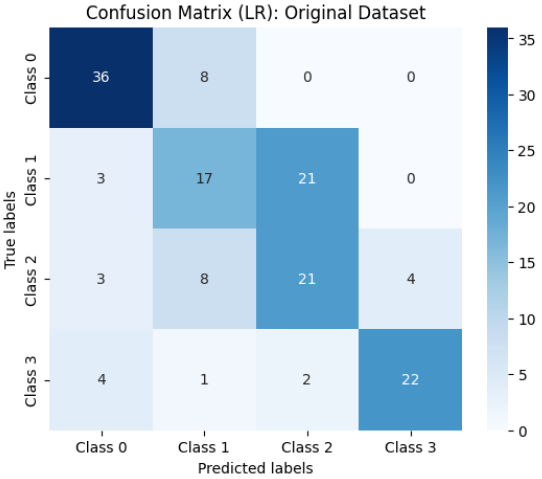
این داده دارای ۲ ویژگی و ۴ کلاس است. مشخصا داده پیچیده‌تر و سخت‌تری برای طبقه بندی خطی ساده است و می‌توان ضعف عملکرد این روش‌ها را برای این داده پیش بینی کرد.

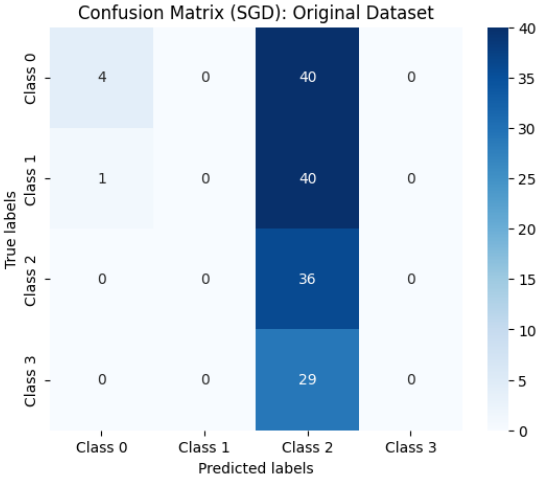
این بار برای بهبود نتایج، مدل‌ها به صورت زیر تعریف شدند:

```
log_reg = LogisticRegression(multi_class='multinomial', solver='newton-cg', max_iter=10000, C=0.5, tol=1e-4, random_state=14)
sgd_clf = SGDClassifier(loss='log_loss',max_iter=1000, tol=1e-5, alpha=1e-6, random_state=14)
```

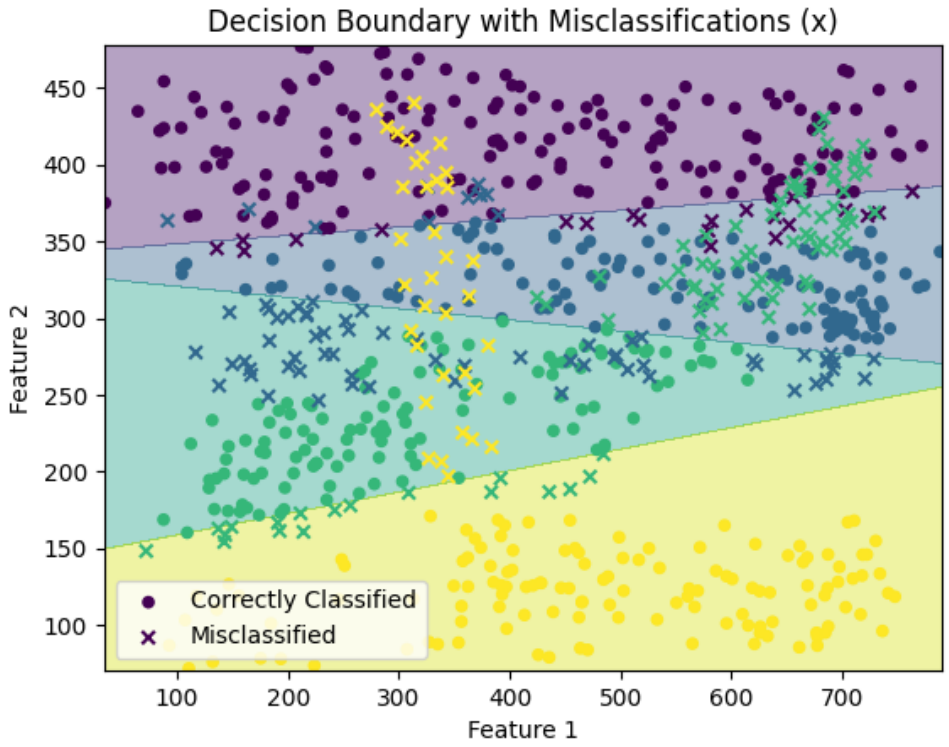
نتایج به صورت زیر می‌باشد:

```
Log regression accuracy: 0.۶۴
SGD accuracy: 0.۲۶
```

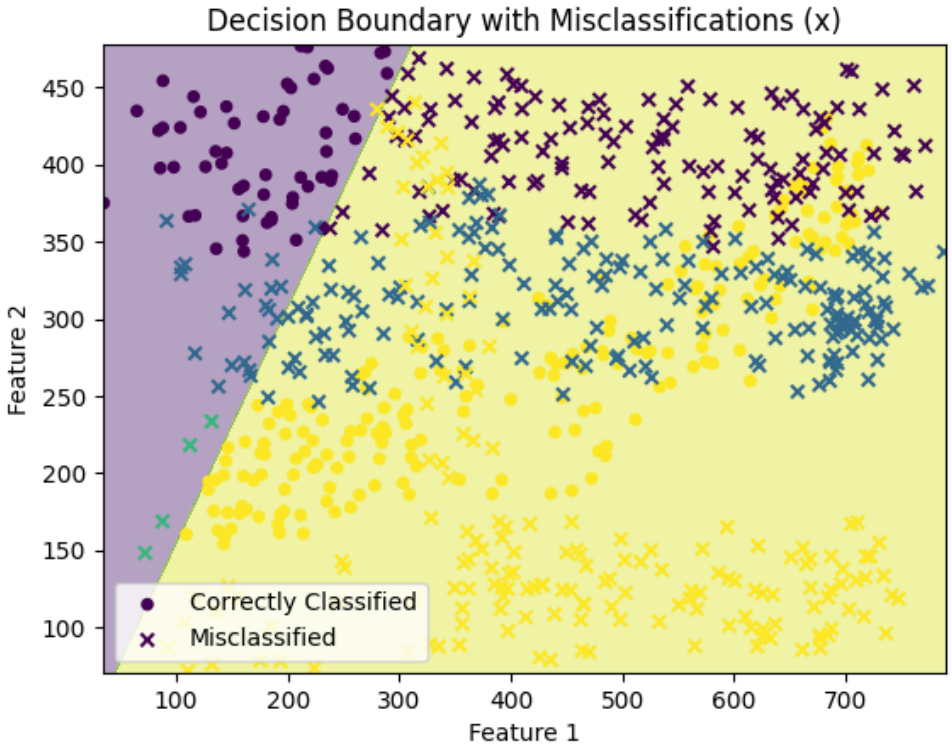




خطوط مرزی نیز به صورت زیر بدست آمدند:



شکل ۱۰- خطوط مرزی بدست آمده برای داده‌های **drawdata** با روش **LR**



شکل ۱۱- خطوط مرزی بدست آمده برای داده‌های **drawdata** با روش **SGD**

مشاهده می‌شود که طبق پیش بینی، مدل‌ها به خوبی عمل نکرده‌اند و مدل SGD کلا نتوانسته طبقه بندی را انجام دهد.

سوال ۲

۲

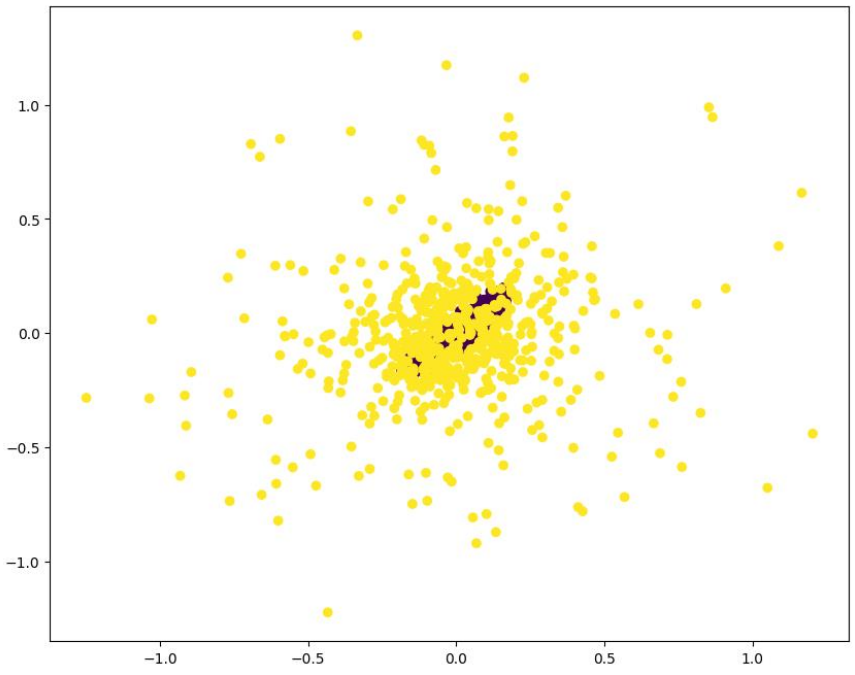
الف

برای استخراج داده‌ها از داده اصلی به صورت زیر عمل شده‌است:

```
import pandas as pd
import scipy.io as scio
import numpy as np

mat = scio.loadmat('99.mat')
variables = scio.whosmat('99.mat')

data1 = mat['X098_DE_time']
data2 = mat['X098_FE_time']

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

normal=[]

for i in range(1000):
    a=np.concatenate([df1.values[i*200:200+i*200], df2.values[i*200:200+i*200]])
    a=np.reshape(a,(200,2))
    normal.append(a)


normal=np.squeeze(normal)
```

```
mat = scio.loadmat('107.mat')
variables = scio.whosmat('107.mat')

data1 = mat['X107_DE_time']
data2 = mat['X107_FE_time']

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

f1=[]

for i in range(600):
    a=np.concatenate([df1.values[i*200:200+i*200], df2.values[i*200:200+i*200]])
    a=np.reshape(a,(200,2))
    f1.append(a)


f1=np.squeeze(f1)
```

در کل، ۱۰۰۰ داده ۲۰۰ در ۲ از کلاس نرمال و ۶۰۰ داده ۲۰۰ در ۲ از کلاس فالت بدست آمد که با هم concat شدند و به ازای هر اندیس متعلق به آن کلاس در کل داده برای داده‌های نرمال، مقدار ۰ و برای داده‌های فالت مقدار ۱ به عنوان برچسب تعریف شد. نمایش داده‌ها در یک فضای دو بعدی به صورت زیر می‌باشد:



شکل ۱۲- توزیع داده‌ها (نرمال با زرد و فالت با بنفش)

توزیع داده‌ها نشان‌گر درهم تنیدگی داده‌های فالت با داده‌های نرمال است. همچنین واریانس داده‌های نرمال بسیار بیشتر از داده‌های خطاست.

ب

ویژگی‌های زیر استخراج شدند.

```
std_X= X.std(axis=1)
peak_X= X.max(axis=1)

from scipy.stats import skew, kurtosis
skewness_X= skew(X, axis=1)
kurtosis_X= kurtosis(X, axis=1)

crest_factor_X = np.max(X, axis=1) / np.sqrt(np.mean(X**2, axis=1))
ptp_X= np.ptp(X, axis=1)
mean_X= np.mean(X, axis=1)
rms_X = np.sqrt(np.mean(X**2, axis=1))
abs_mean_X= np.mean(np.abs(X), axis=1)
```

```
X_new=np.concatenate([std_X, peak_X, skewness_X, kurtosis_X, crest_factor_X, ptp_X, mean_X, rms_X, abs_mean_X],axis=1)
```

```
X_new.shape
```

```
(1600, 18)
```

ج

به هم ریختن یا بُر زدن داده‌ها و تقسیم به مجموعه های آموزش و تست، مراحل بسیار مهمی در آماده سازی داده ها برای مدل های یادگیری ماشین است. این فرآیندها به دستیابی به یک مدل قوی‌تر و کلی‌تر کمک می کنند.

درهم ریختن داده‌ها، فرآیندی شامل مرتب سازی مجدد به طور تصادفی نقاط داده در مجموعه داده است. دلیل اصلی آن، این است که اطمینان حاصل شود که داده های آموزشی و آزمایشی هیچ گونه سوگیری احتمالی موجود در ترتیب داده ها را به ارث نمی‌برند. این امر به ویژه در مواردی مانند همین مجموعه داده که مجموعه داده ممکن است مرتب شده باشد یا در یک توالی خاص است، می تواند منجر به یادگیری جانبدارانه شود، مهم است. برای مثال، اگر این مجموعه داده که بر اساس برچسب‌ها مرتب شده‌اند به مجموعه‌های آموزشی و آزمایشی بدون در هم ریختگی تقسیم شود، مدل ممکن است فقط یاد بگیرد که زیرمجموعه‌ای از برچسب‌ها را به طور دقیق پیش‌بینی کند.

پس از بر زدن، داده ها به مجموعه های آموزشی و آزمایشی تقسیم می شوند. مجموعه آموزشی برای آموزش مدل یادگیری ماشین استفاده می شود، در حالی که مجموعه تست برای ارزیابی عملکرد آن استفاده می شود. یک نسبت تقسیم معمول ۸۰٪ برای آموزش و ۲۰٪ برای آزمایش است.

کد این بخش به صورت ساده به شرح زیر است:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data.values, y, test_size=0.2, random_state=14) #shuffling is always true
```

د

نرمال سازی داده ها یک مرحله پیش پردازش است که برای مقیاس بندی داده های عددی به منظور قرار گرفتن در یک محدوده خاص، اغلب بین ۰ و ۱ یا -۱ و ۱ استفاده می شود. این فرآیند برای بسیاری از الگوریتم های یادگیری ماشین، به ویژه الگوریتم‌هایی که فاصله بین نقطه داده را محاسبه می کنند، بسیار مهم است، زیرا تضمین می کند که همه ویژگی ها به طور مساوی در محاسبات فاصله مشارکت دارند. دو روش متداول زیر وجود دارند:

Min-Max Scaling: این روش داده ها را بین یک محدوده مشخص (معمولاً ۰ تا ۱) با استفاده از حداقل و حداکثر مقادیر داده ها مقیاس می کند.

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

Z-Score Normalization (Standardization): این روش داده ها را به گونه ای مقیاس بندی می کند که میانگین ۰ و انحراف معیار ۱ داشته باشد.

$$X_{\text{norm}} = \frac{X - \mu}{\sigma}$$

در اینجا از روش اول استفاده شده که به صورت زیر اجرا شد:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

در اینجا، داده‌های بخش «ارزیابی» در فرآیند نرمال‌سازی استفاده نشده‌است. این امر از نشت اطلاعات جلوگیری می‌کند، که در آن مدل به طور غیرمستقیم اطلاعاتی را از مجموعه تست در طول آموزش می‌آموزد که منجر به تخمین عملکرد بیش از حد خوش‌بینانه می‌شود. اسکیلر برای یادگیری پارامترهای نرمال‌سازی روی داده‌های آموزشی فیت می‌شود و سپس برای مقیاس‌بندی دقیق داده‌ها به طور پیوسته روی داده‌های آموزشی و آزمایشی اعمال می‌شود.

۳

مرحله به مرحله، به صورت زیر پیش می‌رویم:

مرحله ۱: مدل رگرسیون لجستیک

مدل رگرسیون لجستیک احتمال تعلق یک ورودی داده شده به یک کلاس خاص را پیش‌بینی می‌کند. پیش بینی $\hat{y}$ با استفاده از تابع سیگموئید محاسبه می‌شود:

$$\hat{y} = \frac{1}{1 + e^{-(w^T x + b)}}$$

که x بردار ویژگی ورودی، w بردار وزن، و b بایاس است.

مرحله ۲: تابع زیان

از تابع زیان آنتروپی متقاطع باینری استفاده می‌شود. این تابع با قرار دادن محدودیت برای احتمالات پیش‌بینی شده اطمینان از ثبات عددی را فراهم می‌آورد تا از مقادیر بی‌نهایتی که می‌توانند منجر به عملیات لگاریتمی تعریف نشده شوند، جلوگیری کند. تابع میانگین ترکیب زیان‌ها برای هر دو کلاس نرمال

و فالت را برای تمام پیش‌بینی‌ها برمی‌گرداند که برای ارزیابی و بهینه‌سازی عملکرد مدل دسته‌بندی دودویی استفاده می‌شود. معادله آن به صورت زیر است:

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

مرحله ۳: الگوریتم یادگیری - نزول گرادیان

وزن‌های w و بایاس b را با استفاده از گرادیان تابع زیان و با توجه به w و b قبلی به‌روز می‌کنیم.

$$w := w - \alpha \frac{\partial L}{\partial w}$$

$$b := b - \alpha \frac{\partial L}{\partial b}$$

مرحله ۴: معیارهای ارزیابی

دقت: کسری از پیش‌بینی‌ها که مدل ما درست انجام شد.

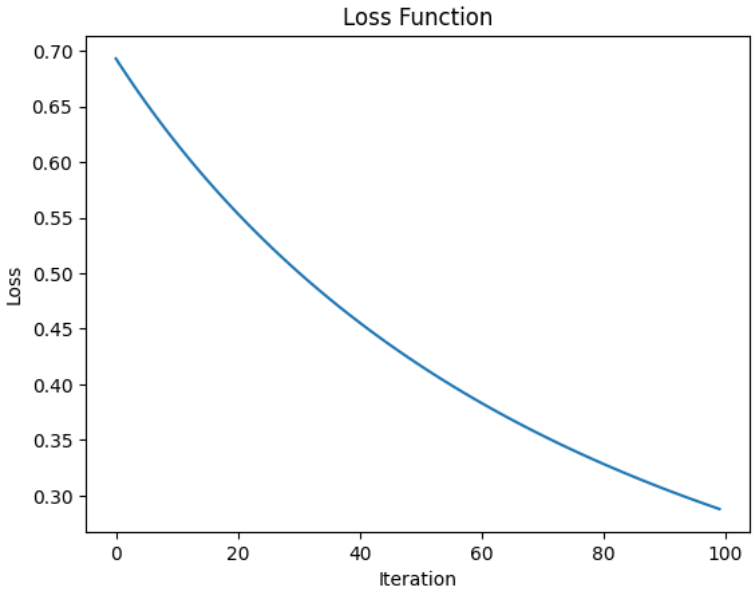
Precision: از بین کلاس‌های مثبتی که پیش‌بینی کردیم، چه تعداد واقعاً مثبت هستند.

مراحل بالا بعد از تعریف با تعاریف زیر اجرا می‌شوند:

```
# Model initialization
w = np.zeros((X_train_normalized.shape[1], 1))
b = 0
y_train = y_train.reshape(-1, 1)
learning_rate = 0.05
iterations = 100
```

خروجی آن به صورت زیر است:

```
Iteration 90: Loss 0.3079461191877987
Iteration 91: Loss 0.3058194667027207
Iteration 92: Loss 0.30371868024052523
Iteration 93: Loss 0.3016433390710418
Iteration 94: Loss 0.2995930306063449
Iteration 95: Loss 0.297567350225839
Iteration 96: Loss 0.2955659011054113
Iteration 97: Loss 0.29358829405053627
Iteration 98: Loss 0.2916341473332233
Iteration 99: Loss 0.2897030865327033
Iteration 100: Loss 0.2877947443797544

Test Accuracy: 1.0, Test Precision: 1.0
```



شکل ۱۳- نمودار زیان مدل ساخته شده به صورت دستی

نمودار تابع ضرر معمولا نشان می دهد که چگونه ضرر مدل در طول تکرار کاهش می یابد. یک نمودار به آرامی کاهش نشان می دهد که مدل به درستی یاد می گیرد. تغییرات یا افزایش شدید ممکن است نشان دهنده مشکلاتی در میزان یادگیری یا داده باشد. با این حال، عملکرد مدل را در داده‌های دیده نشده (داده‌های آزمایشی) تضمین نمی‌کند. تطابق بیش از حد با داده های آموزشی ممکن است منجر به کاهش زیان در فرایند آموزش اما عملکرد تست ضعیف شود.

راه حل برای درک بهتر عملکرد مدل، ارزیابی آن با استفاده از مجموعه تست با معیارهایی مانند دقت و precision است. تکنیک‌های بیشتر مانند اعتبارسنجی متقاطع می‌توانند تخمین عملکرد قابل اعتمادتری ارائه دهند.

۴

نزدیک‌ترین روش بر این اساس، استفاده از SGD به صورت زیر است:

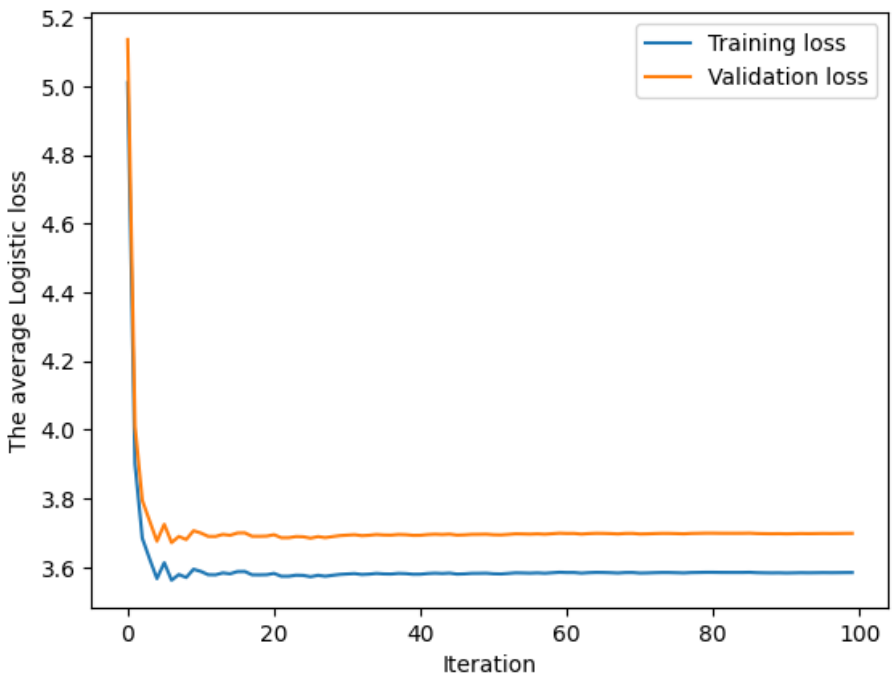
```
# Reshape to be 1-dimensional
y_train_ravel = y_train.ravel()
y_test_ravel = y_test.ravel()

model = SGDClassifier(loss='log_loss', max_iter=100 , learning_rate='constant', eta0=0.05)

model.fit(X_train_normalized, y_train_ravel)
```

اما این روش دسترسی به ما برای محاسبه loss را نداریم. برای همین باید به صورت زیر عمل کنیم:

```python
model = SGDClassifier(loss='log_loss', max_iter=1 , learning_rate='optimal', eta0=0.05)

# Reshape to be 1-dimensional
y_train_ravel = y_train.ravel()
y_test_ravel = y_test.ravel()

train_losses = []
val_losses = []

# Manual training loop
for _ in range(100):
    model.partial_fit(X_train_normalized, y_train_ravel, classes=np.unique(y))  # Partially fit the model

    # Calculate probabilities
    train_probs = model.predict_proba(X_train_normalized)[:, 1]
    val_probs = model.predict_proba(X_test_normalized)[:, 1]

    # Calculate and record the custom loss
    train_loss = compute_loss(y_train, train_probs)
    val_loss = compute_loss(y_test, val_probs)

    train_losses.append(train_loss)
    val_losses.append(val_loss)
```

باید توجه شود که تابع زیان انتخاب شده برای این روش کاملا یکسانی با روش قبلی داشته باشد. پس از یکسری از بررسی‌ها تابع log_loss نزدیک‌ترین گزینه به روش اصلی بود. نتایج بدست آمده به صورت زیر است:

```
Train Accuracy: 1.0, Test Accuracy: 1.0
Train Precision: 1.0, Test Precision: 1.0
```



شکل ۱۴- نمودار زیان بدست آمده با کتابخانه **sklearn**

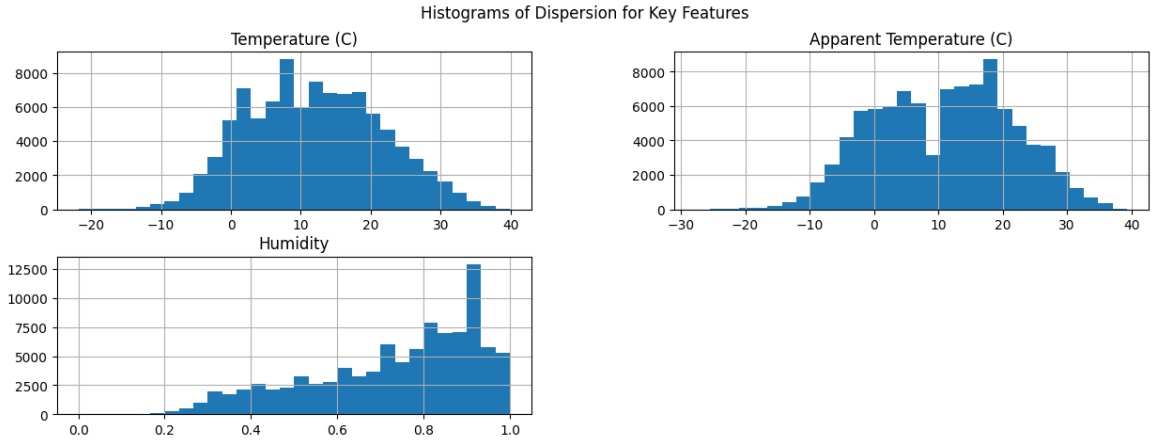
رویکرد دستی وزن‌ها و بایاس‌ها را در یک حلقه for با گرادیان‌های محاسبه‌شده به صراحت به‌روزرسانی می‌کند، در حالی که روش Scikit-Learn از partial_fit استفاده می‌کند که ممکن است شامل بهینه‌سازی‌های اضافی باشد. از طرفی، روش اول مستلزم تنظیم تعداد ثابتی از تکرارها و نرخ یادگیری است که بسته به مقادیر انتخاب شده می تواند همگرا باشد یا نباشد. روش دوم نرخ یادگیری را برای اطمینان از همگرایی تنظیم می کند.

سوال ۳

۱

نمودارهای خواسته شده از داده به صورت زیر است:

شکل ۱۵- هیت مپ ماتریس همبستگی داده‌های عددی



شکل ۱۶- نمودار هیستوگرام داده‌های مورد بررسی

بر اساس ماتریس همبستگی، بین داده‌هایی که نیاز به بررسی است موارد زیر قابل توجه است:

دما و دمای ظاهری: این دو دارای بالاترین همبستگی مثبت ۰.۹۹ هستند، که نشان می دهد با افزایش یا کاهش دما، دمای ظاهری (آنچه انسان درک می کند) تقریباً دقیقاً دنبال می شود.

دما و رطوبت: یک همبستگی منفی قوی بین ۰.۶۳- بین دما و رطوبت وجود دارد. این نشان می دهد که دماهای بالاتر اغلب همزمان با سطوح رطوبت پایین تر است و بالعکس.

دما و رطوبت ظاهری: این جفت همچنین دارای همبستگی منفی قوی مشابه ۰.۶۰- است که نشان می دهد با کاهش رطوبت، دمای ظاهری تمایل به افزایش و با افزایش رطوبت، دمای ظاهری تمایل به کاهش دارد.

حال اگر به هیستوگرام‌ها نیز توجه کنیم:

۱. دما: توزیع دما یک توزیع تقریباً دو وجهی با دو قله را نشان می دهد که دو حالت مختلف را در مجموعه داده نشان می دهد. این می تواند تغییرات دمای فصلی را منعکس کند. به نظر می رسد بخش عمده ای از نقاط داده دما در حدود ۱۰ درجه سانتی گراد تا ۲۰ درجه سانتی گراد متمرکز شده است، که می تواند نشان دهنده آب و هوای معتدل یا رایج ترین محدوده دمایی برای مکان(هایی) باشد که داده ها از آنجا جمع آوری شده اند.

۲. دمای ظاهری: مشابه دما، دمای ظاهری نیز توزیع دووجهی را نشان می‌دهد که با همبستگی بالایی که بین دما و دمای ظاهری در ماتریس همبستگی مشاهده می‌شود، همسو می‌شود. توزیع تقریباً متقارن در حدود ۱۰ درجه سانتیگراد تا ۲۰ درجه سانتیگراد است که احتمالاً مطابق با محدوده دمایی است که انسان راحت می یابد. گسترش و شکل توزیع دمای ظاهری مشابه دمای واقعی است و رابطه قوی بین این دو را تقویت می کند.

۳. رطوبت: توزیع رطوبت به سمت راست منحرف شده است، که نشان می دهد سطوح رطوبت کمتر کمتر از سطوح بالاتر در این مجموعه داده رخ می دهد. تجمع قابل توجهی از نقاط داده در سطوح رطوبت بالاتر، به ویژه در حدود ۰.۸ تا ۱.۰ وجود دارد که می تواند یک محیط به طور کلی مرطوب را نشان دهد. این واقعیت که موارد کمتری از رطوبت کم وجود دارد می‌تواند نشان دهد که داده‌ها اغلب شرایط خشک را نشان نمی‌دهند.

۲

برای استفاده از روش‌های برآورد حداقل مربعات (LS) و حداقل مربعات منظم (RLS، همچنین به عنوان رگرسیون ریج شناخته می‌شود) برای تجزیه و تحلیل روابط موجود در مجموعه داده‌ها، ما بر روی دو تجزیه و تحلیل جداگانه تمرکز خواهیم کرد:

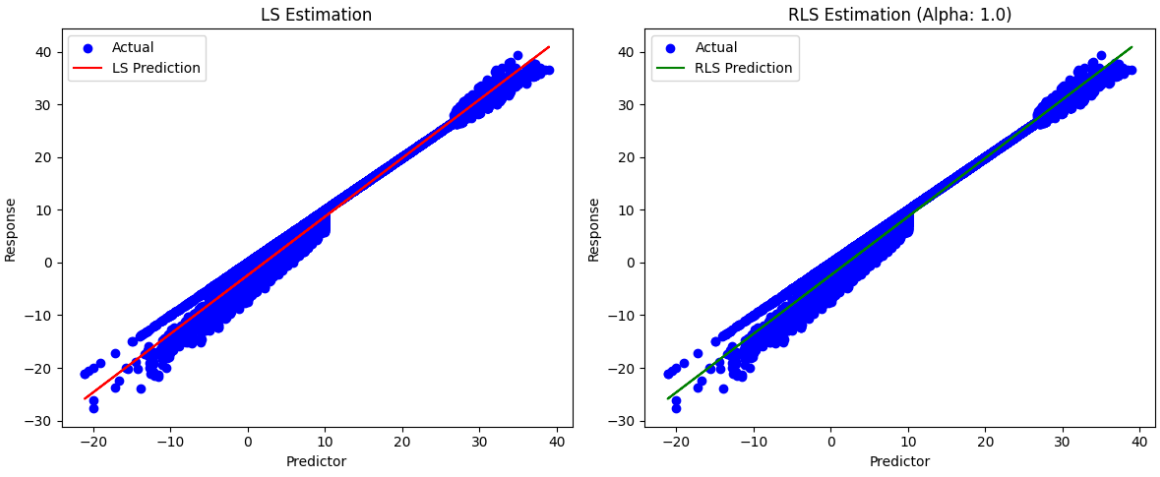
۱- دما ظاهری و رطوبت

۲- دما و رطوبت

در هر دو تحلیل، دما (C) به عنوان یک متغیر پیش‌بینی‌کننده (متغیر مستقل) برای بررسی رابطه آن با رطوبت و دمای ظاهری (C) (متغیرهای وابسته در تحلیل‌های جداگانه) استفاده می‌شود. برای مقایسه تخمین‌های LS و RLS، میانگین مربعات خطا (MSE) را برای هر دو مدل محاسبه کرده و نتایج رگرسیون را رسم می‌کنیم.

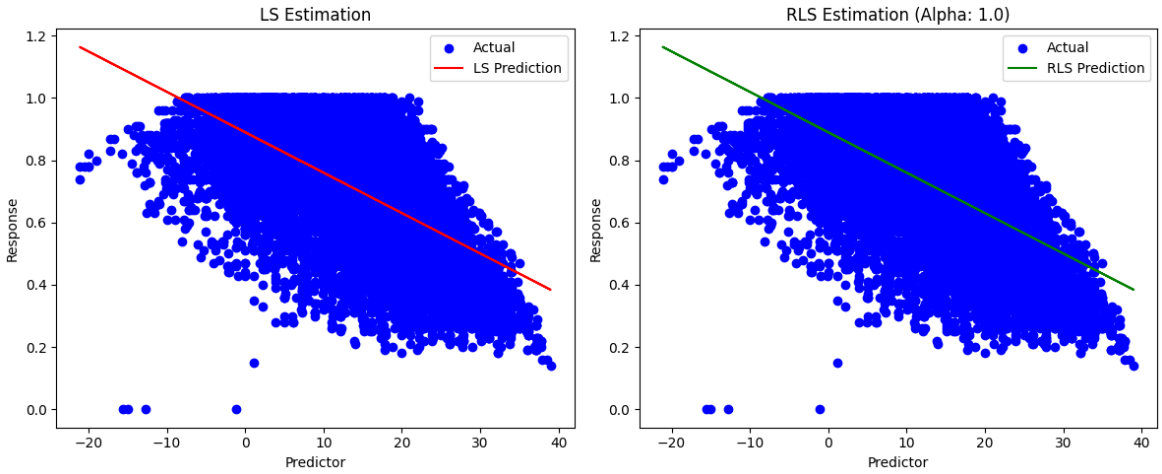
نتایج به صورت زیر است:

```
Analysis 1: Apparent Temperature and Humidity
Mean squared error of LS= 1.6839210909420195
Mean squared error of RLS= 1.683921100386982
```



شکل ۱۷- نمودار تخمین با **LS** و **RLS** برای داده‌های دما ظاهری و رطوبت

```
Analysis 2: Temperature and Humidity
Mean squared error of LS= 0.023128097164260626
Mean squared error of RLS= 0.023128097108714295
```



شکل ۱۸- نمودار تخمین با **LS** و **RLS** برای داده‌های دما و رطوبت

مشخصا در حالت اول به دلیل نوع توزیع داده‌ها، تخمین بهتری نسبت به حالت دوم ارائه شده؛ البته که در حالت دوم رویه کلی داده‌ها پیدا شده اما تنها راستای آن را در بر دارد.

۳

حداقل مربعات وزنی (WLS) نوعی از روش رگرسیون حداقل مربعات معمولی (OLS) است که برای مدیریت ناهمسانی طراحی شده است. ناهمگونی زمانی اتفاق می افتد که واریانس خطاها در مدل رگرسیون در بین مشاهدات ثابت نباشد. در چنین مواردی، تخمین های OLS، در حالی که هنوز بی طرفانه هستند، دیگر بهترین تخمین های خطی بی طرفانه (BLUE) نیستند زیرا کوچکترین واریانس را ندارند. WLS با تخصیص وزن به هر نقطه داده، این موضوع را برطرف می کند و به نقاط داده با واریانس بالاتر اهمیت کمتری می دهد.

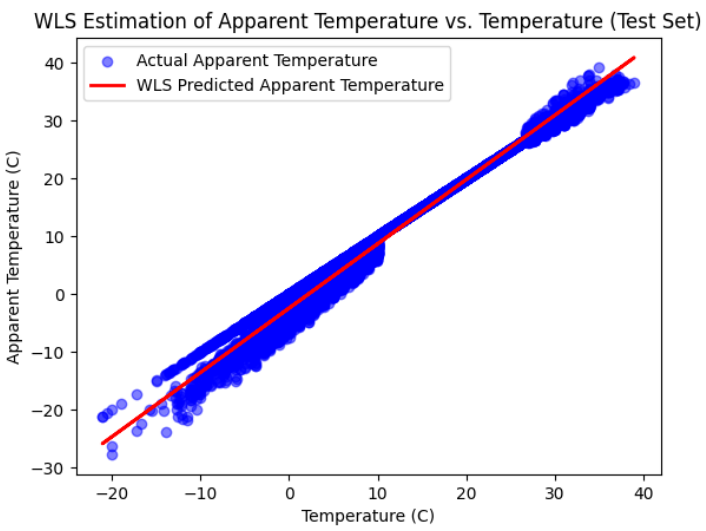
ایده این است که با وزن دهی متفاوت مشاهدات، می توان مجموع مجذورهای باقیمانده وزنی را به حداقل رساند و در صورت نابرابر بودن واریانس، به تخمین های قابل اعتمادتری منجر شد. وزن ها معمولاً به عنوان معکوس واریانس عبارت خطای هر مشاهده انتخاب می شوند، به این معنی که به مشاهدات با واریانس بالاتر (پایایی کمتر) وزن کمتری در تحلیل رگرسیون داده می شود.

برای اعمال WLS به مجموعه داده‌ای که رابطه بین دما و رطوبت و دمای ظاهری و رطوبت را بررسی می‌کند، باید با اطمینان الگوی ناهمسانی را در باقیمانده‌های یک تناسب حداقل مربعات معمولی اولیه (OLS) شناسایی یا فرض کنیم. وزن‌ها را بر اساس متغیر دیگری یا تابعی از متغیرها انتخاب می‌کنیم تا اطمینان حاصل کنیم که واریانس خطاها متناسب با واریانس متغیر پیش‌بینی‌کننده است. کد ساده شده این روش به صورت زیر می‌باشد:

```python
import statsmodels.api as sm

# Split the data for Apparent Temperature and Humidity
X_app_temp = sm.add_constant(X)
y_app_temp_wls = y_app_temp

X_train_app_temp, X_test_app_temp, y_train_app_temp, y_test_app_temp = train_test_split(X_app_temp, y_app_temp_wls, test_size=0.2, random_state=14)

weights_app_temp_train = np.ones_like(y_train_app_temp)

wls_model_app_temp_train = sm.WLS(y_train_app_temp, X_train_app_temp, weights=weights_app_temp_train).fit()

y_pred_wls_app_temp = wls_model_app_temp_train.predict(X_test_app_temp)

mse_wls_app_temp = mean_squared_error(y_test_app_temp, y_pred_wls_app_temp)
print(f'Mean Squared Error of WLS: {mse_wls_app_temp}')
```

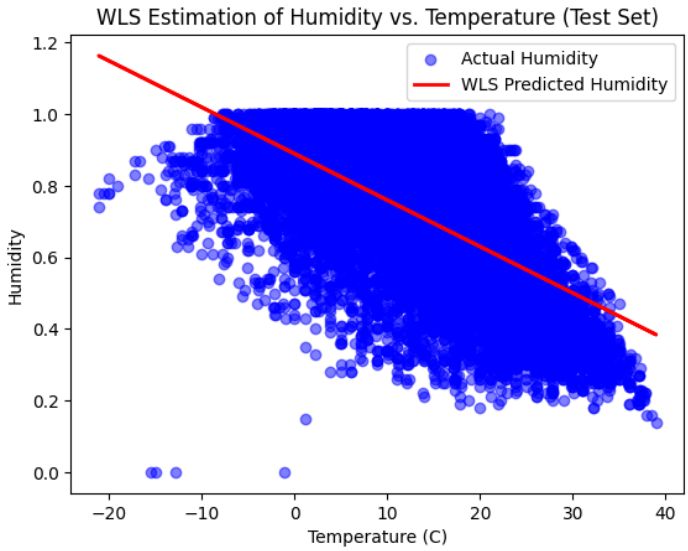
نتایج بدست آمده به صورت زیر است:

```
Mean Squared Error of WLS: 1.6839210909420186
```



شکل ۱۹- نمودار تخمین با **WLS** برای داده‌های دما ظاهری و رطوبت

```
Mean squared error of WLS= 0.02312809716426063
```



شکل ۲۰- نمودار تخمین با **WLS** برای داده‌های دما و رطوبت

نتایج این قسمت با قسمت‌های قبلی هم‌راستا و بسیار نزدیک است.

۴

الگوریتم "کمترین مربعات منظم مبتنی بر تجزیه QR (RLS)" گونه ای از روش تخمین حداقل مربعات است که رگولاریزاسیون را شامل می شود و از تجزیه QR برای حل معادله رگرسیون استفاده می کند. این رویکرد به ویژه برای مقابله با چند خطی، بهبود ثبات عددی، و مدیریت بیش از حد برازش از طریق رگولاریزاسیون مفید است.

حداقل مربعات منظم (RLS)

روش RLS یک عبارت رگولاریزاسیون را به تابع هزینه حداقل مربعات اضافه می‌کند. اصطلاح رگولاریزاسیون معمولاً شامل هنجار L2 بردار ضریب است و آن را شبیه به رگرسیون ریج می کند. هدف به حداقل رساندن تابع هزینه زیر است:

$$\text{Cost} = ||y - X\beta||_2^2 + \lambda||\beta||_2^2$$

بطوری که:

y بردار پاسخ است،

X ماتریس طراحی است،

β بردار ضریب است،

و λ پارامتر رگولاریزاسیون است که میزان انقباض را کنترل می کند.

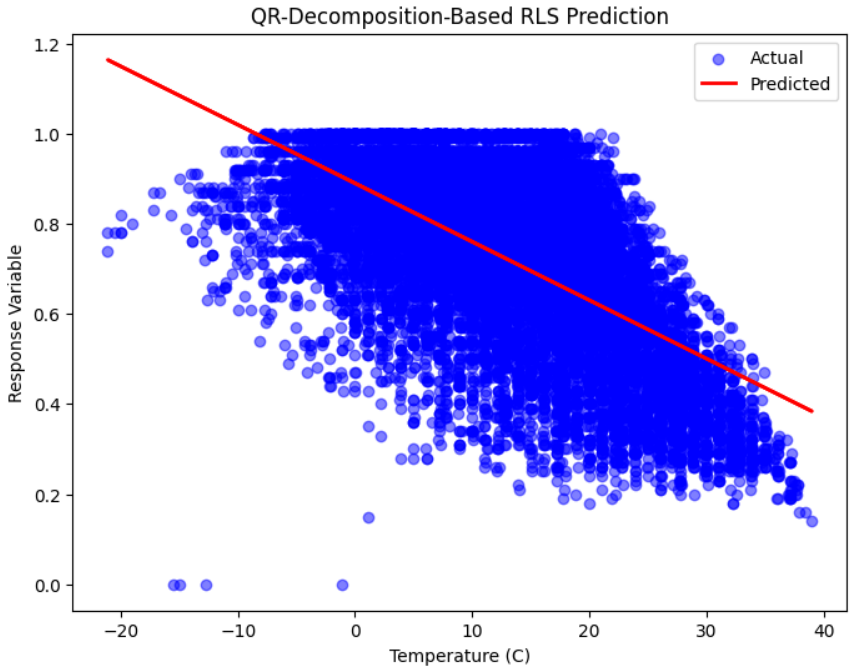
تجزیه QR یک تکنیک فاکتورسازی ماتریس است که یک ماتریس را به حاصلضرب یک ماتریس متعامد (Q) و یک ماتریس مثلثی بالایی (R) تجزیه می‌کند. برای ماتریس طراحی X، تجزیه X=QR است. این تجزیه می تواند حل سیستم خطی را ساده کرده و ثبات عددی را بهبود بخشد.

با کمک کد زیر، این روش را پیاده سازی می‌کنیم:

```python
from numpy.linalg import qr, inv
# Correcting the implementation details for QR-Decomposition-Based RLS and adding plots
def qr_decomposition_based_rls_and_plot(X_train, y_train, X_test, y_test, lam=1.0, title='QR-Decomposition-Based RLS Prediction'):
    n, m = X_train.shape
    XTX = X_train.T @ X_train
    XTy = X_train.T @ y_train
    identity_matrix = np.eye(m)

    # Regularization
    XTX_lam = XTX + lam * identity_matrix

    # QR Decomposition
    Q, R = qr(XTX_lam)

    # Solve for coefficients
    beta = np.linalg.inv(R) @ Q.T @ XTy

    y_pred = X_test @ beta

    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE for QR-Decomposition-Based RLS (λ={lam}): {mse}')

    plt.figure(figsize=(8, 6))
    plt.scatter(X_test[:, 1], y_test, color='blue', alpha=0.5, label='Actual')
    plt.plot(X_test[:, 1], X_test @ beta, color='red', label='Predicted', linewidth=2)
    plt.title(title)
    plt.xlabel('Temperature (C)')
    plt.ylabel('Response Variable')
    plt.legend()
    plt.show()

    return beta

# Prepare the data (with intercept)
X_with_intercept = np.hstack([np.ones((X.shape[0], 1)), X])
X_train, X_test, y_train_humidity, y_test_humidity = train_test_split(X_with_intercept, y_humidity, test_size=0.2, random_state=14)
X_train, X_test, y_train_app_temp, y_test_app_temp = train_test_split(X_with_intercept, y_app_temp, test_size=0.2, random_state=14)

print("QR-Decomposition-Based RLS Analysis for Temperature:")
beta_qr_rls_humidity, mse_qr_rls_humidity = qr_decomposition_based_rls_and_plot(X_train, y_train_humidity, X_test, y_test_humidity, lam=0.1)

print("\nQR-Decomposition-Based RLS Analysis for Apparent Temperature:")
beta_qr_rls_app_temp, mse_qr_rls_app_temp = qr_decomposition_based_rls_and_plot(X_train, y_train_app_temp, X_test, y_test_app_temp, lam=0.1)
```

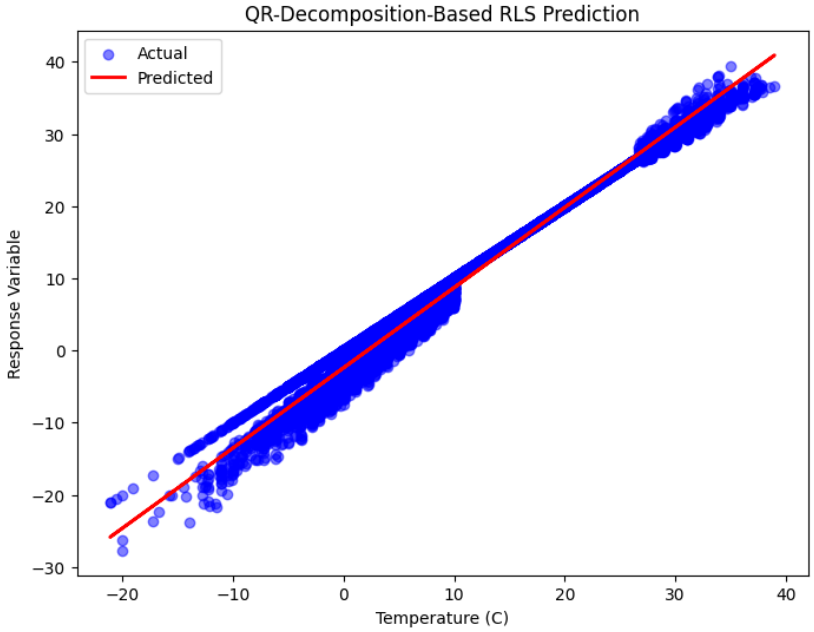
نتایج زیر بدست آمدند:

```
QR-Decomposition-Based RLS Analysis for Temperature:
MSE for QR-Decomposition-Based RLS (λ=0.1): 0.02312809033346163
```



شکل ۲۱- نمودار تخمین با **QR-Decomposition-Based RLS** برای داده‌های دماو رطوبت

```
QR-Decomposition-Based RLS Analysis for Apparent Temperature:
MSE for QR-Decomposition-Based RLS (λ=0.1): 1.6839210510763
```



شکل ۲۲- نمودار تخمین با **QR-Decomposition-Based RLS** برای داده‌های دما ظاهری و رطوبت

مراجع