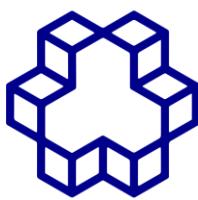


به نام خدا



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

مینی پروژه اول

Github link:

<https://github.com/FatemehShokrollahiMoghadam>

google drive link:

<https://drive.google.com/drive/folders/1J257gP6OFWAWBdwDVx3DuEpFa299DFbz?usp=sharing>

نگارنده:

فاطمه شکراللهی مقدم

۴۰۲۰۷۳۶۴

۱۴۰۳ بهار

فهرست

۳	پیشگفتار
۴	سوال اول
۴	۱-۱
۵	۱-۲
۷	۱-۳
۱۱	۱-۴
۱۲	۱-۵
۱۵	سوال دوم
۱۵	۲-۱
۱۷	۲-۲
۱۷	آ
۱۸	ب
۱۹	ج
۲۰	د
۲۱	۲-۳
۲۴	۲-۴
۲۷	سوال سوم
۲۷	۳-۱
۳۰	۳-۲
۳۸	۳-۳
۴۱	مراجع

پیشگفتار

در صفحه اول گزارش لینگ گیت هاب و لینک گوگل درایو مربوط به نوت بوک های هر سوال آورده شده است.

در اینجا نیز لینک گوگل کولب نوت بوک هر سوال به ترتیب آورده می شود:

لینک نوت بوک سوال اول:

https://colab.research.google.com/drive/1arc7kPY61ZHGsdl6j5we1-_U5VCJE_V31?usp=sharing

لینک نوت بوک سوال دوم:

https://colab.research.google.com/drive/1Q_VQbsO9Ma_rQqu52HXVx09ZkaWovCOV?usp=sharing

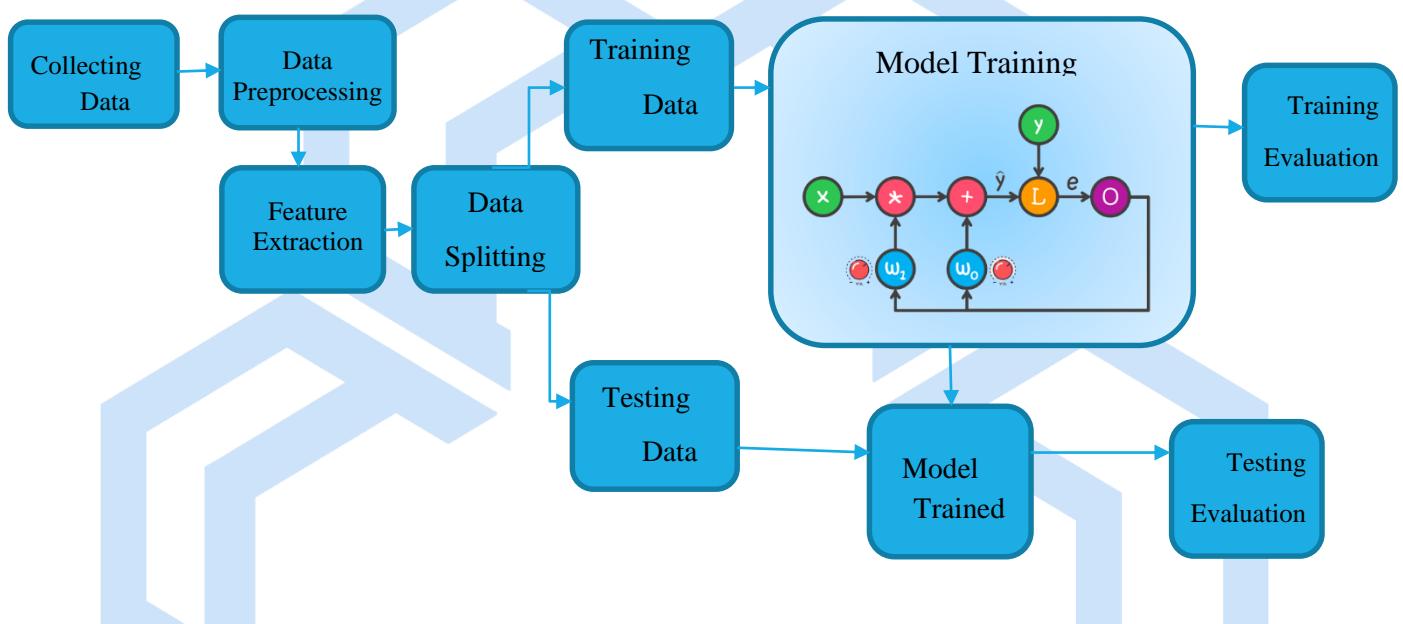
لینک نوت بوک سوال سوم:

https://colab.research.google.com/drive/1kIFWUfOQYCATFBfIP_IshPWkO0RNT5Wo?usp=sharing

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 چند کلاسه در کدام قسمت از این دیاگرام بلوکی تغییراتی ایجاد می کند؟ توضیح دهید.



اولین مرحله، جمع آوری داده است. سپس پیش پردازش داده ها بگونه ای انجام می گیرد تا داده برای عملیات آموزش آماده شود. این مرحله شامل حذف داده های پوچ و نرمالیزه کردن داده ها در صورت نیاز می باشد. سپس ویژگی های مورد نیاز از داده های پیش پردازش شده استخراج می شود. در مرحله بعد، داده ها را به دو دسته آموزش و ارزیابی با نسبت مناسبی تقسیم می کنیم. داده های آموزش را به مدل طبقه بندی خطی می دهیم (مدل نشان داده شده در این قسمت، برگرفته از اسلاید های تدریسیاری و مربوط به طبقه بندی دو کلاسه است). مدل طبقه بندی خطی با استفاده از ویژگی های استخراج شده و برچسب های متناظر آموزش داده می شود. مدل، رابطه بین ویژگی های ورودی و هدف را با استفاده از یک الگوریتم طبقه بندی خطی مانند رگرسیون لجستیک یاد می گیرد. برای مثال در رگرسیون لجستیک داده با وزن اولیه ای به تابع سیگموئید داده می شود سپس تابع اتلافی مانند کراس انتروپی برای آن تعریف می شود که توسط این تابع خروجی تابع سیگموئید با خروجی هدف مقایسه می شود و توسط بهینه

سازی مانند الگوریتم های بر پایه گرادیان، ماتریس وزن اولیه آپدیت می شود تا مدل به دقت قابل قبولی، دست یابد.

در مرحله بعد داده تست که مدل پیش از این، آن ها را ندیده، داده می شود تا دقیق و کیفیت مدل ارزیابی شود. اگه مرحله ارزیابی نتیجه قابل قبول داشت، مدل به عنوان مدل بهینه معرفی می شود. در غیر اینصورت با تغییر های پارامترها (ماتریس وزن اولیه، نرخ یادگیری و...) مدل را بهبود می بخشیم.

با افزایش کلاس، تعداد پارامتر های مدل طبقه بند خطی افزایش می یابند. در بلوک دیاگرام فوق و در بلوک Model Training با توجه به تعداد کلاس $w_1, w_2, w_3, \dots, w_n$ خواهیم داشت.

۱-۲

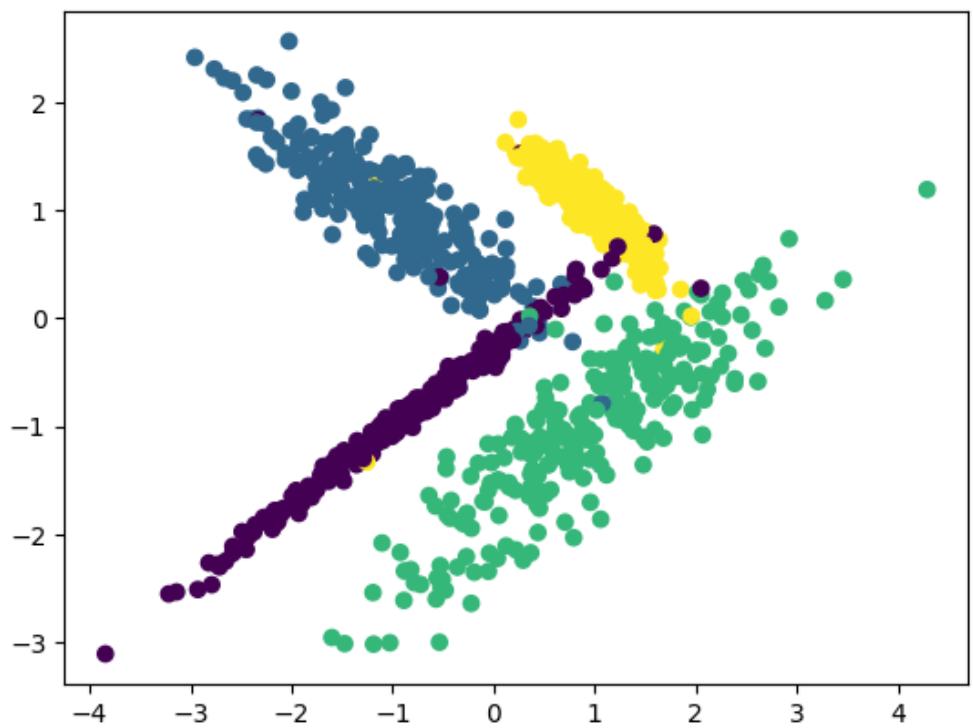
با استفاده از `sklearn.datasets` یک دیتاست با ۱۰۰۰ نمونه، ۴ کلاس و ۳ ویژگی تولید کنید و آن را به صورتی مناسب نمایش دهید. آیا دیتاستی که تولید کردید چالش برانگیز است؟ چرا؟ به چه طریقی می توانید دیتاست تولیدشده خود را چالش برانگیزتر و سخت تر کنید؟

با استفاده از `make_classification` دیتاست مذکور را تولید می کنیم:

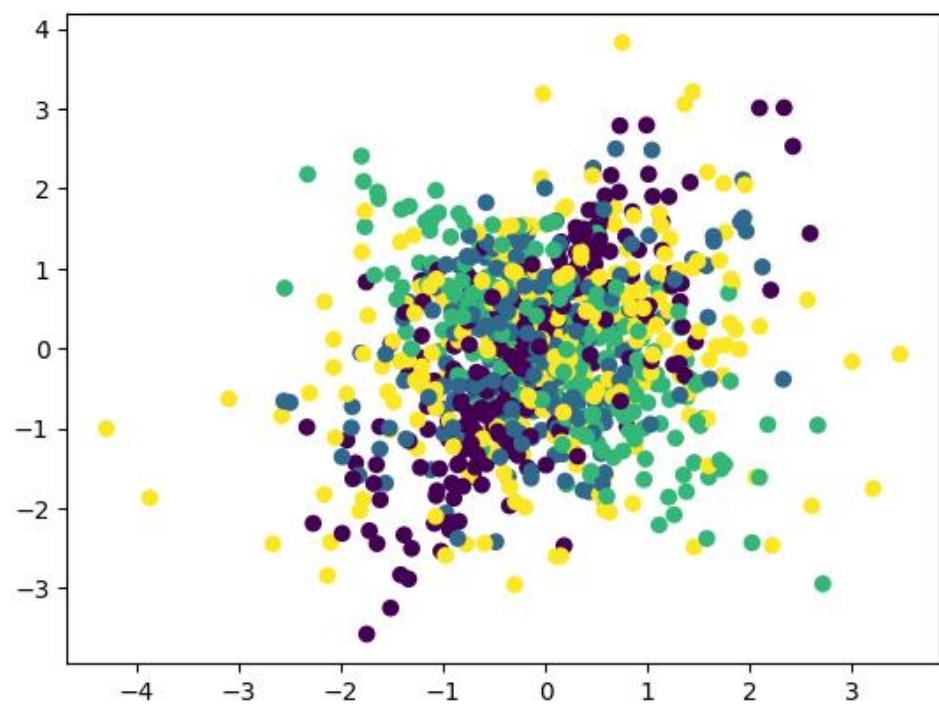
```
[2]: import matplotlib.pyplot as plt  
x, y = make_classification(n_samples=1000, n_features=3, n_redundant=0, n_clusters_per_class=1, class_sep=1, n_classes=4, n_repeated=0, n_informative=2, random_state=64)  
print(x.shape, y.shape)  
plt.scatter(x[:, 0], x[:, 1], c=y)
```

این دستور ورودی هایی می گیرد که با استفاده از آن می توان تعداد نمونه، تعداد ویژگی و تعداد کلاس را تعیین کرد. علاوه بر این، `n_informative` تعداد ویژگی های دارای اطلاعات، `n_redundant` تعداد ویژگی افزونه، `n_repeated` تعداد ویژگی تکراری، `n_clusters_per_class` تعداد خوش در هر کلاس، `class_sep` که هر چه زیادتر باشد در هم رفتگی داده ها را کم می کند.

دیتاست تولید شده با `class_sep=1` و `n_informative=2` و `n_redundant=0` و `n_repeated=0` بصورت زیر است:



همانطور که در تصویر بالا پیداست، در نقاطی داده های هر کلاس در هم رفتگی دارند. برای ایجاد چالش بیشتر می توان، `class_sep` را کاهش داد و `n_informative=3` و `n_clusters_per_class = 2` قرار داد:



۱-۳

با استفاده از حداقل دو طبقه بند خطی آماده پایتون و در نظر گرفتن فراپارامترهای مناسب، چهار کلاس موجود در دیتاست قسمت قبلی را از هم تفکیک کنید. ضمن توضیح روند انتخاب فراپارامترها

(مانند تعداد دوره آموزش و نرخ یادگیری)، نتیجه دقت آموزش و ارزیابی را نمایش دهید. برای بهبود نتیجه از چه تکنیک هایی استفاده کردید؟

با استفاده از LogisticRegression، SGDClassifier از طبقه بند های sklearn.linear_model استفاده می کنیم.

داده ها را به نسبت ۸۰ به ۲۰ به دو دسته آموزش و ارزیابی تقسیم می کنیم.

```
[15]: 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, random_state=64)
       x_train.shape, y_train.shape, x_test.shape, y_test.shape
((800, 3), (800,), (200, 3), (200,))
```

:LogisticRegression

ورودی هایی به شرح زیر دریافت می کند:

C: معکوس قدرت تنظیم، یعنی هرچه مقدار کمتری داشته باشد قدرت تنظیم بیشتری دارد.

Solver: الگوریتم بهینه سازی را مشخص می کند. به صورت پیشفرض lbfgs است.

وزن های اولیه هر کلاس را مشخص می کند و بصورت پیشفرض مقدار یک دارد.

max_iter: مانند تعداد تکرار را مشخص می کند و به صورت پیشفرض مقدار ۱۰۰ دارد.

ابتدا با استفاده از مقادیر پیشفرض طبقه بندی را انجام می دهیم.

```
#logisticRegression()
model = LogisticRegression(random_state=64)
model.fit(x_train, y_train)
model.predict(x_test), y_test
(array([2, 1, 0, 1, 0, 1, 2, 0, 1, 3, 3, 2, 3, 1, 3, 3, 0, 1, 3, 1, 0, 1,
       3, 0, 0, 3, 1, 1, 2, 2, 3, 2, 3, 1, 0, 3, 1, 3, 0, 3, 0, 1, 1, 1,
       0, 3, 1, 2, 3, 3, 2, 0, 3, 0, 2, 1, 2, 2, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 3, 3, 2, 3, 0, 3, 3, 1, 3, 1, 1, 3, 2, 3, 0, 2, 0, 1, 2, 2, 1,
       1, 3, 1, 0, 2, 3, 2, 1, 1, 0, 3, 1, 1, 2, 1, 0, 0, 2, 1, 3, 0, 1,
       3, 3, 2, 0, 2, 2, 3, 0, 2, 3, 2, 0, 0, 2, 0, 3, 0, 2, 2, 0, 0, 3,
       1, 0, 3, 1, 2, 3, 1, 2, 2, 2, 0, 1, 0, 1, 2, 2, 2, 3, 0, 3, 3, 3,
       0, 3, 0, 3, 0, 1, 3, 1, 2, 3, 1, 2, 1, 3, 3, 1, 1, 0, 2, 0, 2, 0,
       0, 1, 3, 1, 3, 0, 0, 2, 2, 3, 0, 2, 2, 0, 0, 2, 1, 0, 3, 1, 3, 1,
       3, 3]), array([2, 1, 0, 1, 0, 1, 2, 0, 1, 3, 3, 2, 3, 1, 3, 3, 0, 1, 3, 1, 0, 0,
       3, 0, 0, 3, 1, 1, 2, 2, 3, 2, 3, 1, 0, 3, 1, 0, 0, 3, 0, 1, 1, 1,
       0, 3, 1, 2, 3, 3, 2, 0, 3, 0, 2, 1, 1, 2, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 3, 3, 2, 2, 0, 3, 3, 1, 3, 1, 1, 3, 2, 3, 0, 2, 0, 1, 2, 2, 1,
       1, 3, 1, 0, 2, 3, 2, 1, 1, 0, 0, 1, 1, 2, 1, 0, 2, 2, 0, 3, 0, 1,
       0, 3, 2, 0, 2, 2, 3, 0, 2, 3, 2, 0, 0, 2, 0, 3, 0, 2, 2, 0, 0, 3,
       1, 0, 3, 1, 3, 0, 1, 3, 1, 2, 3, 1, 2, 1, 3, 0, 1, 1, 0, 2, 0, 2,
       0, 1, 3, 1, 0, 0, 0, 2, 2, 3, 0, 2, 2, 0, 0, 2, 1, 0, 3, 1, 3, 1,
       3, 3]))
```

حال مقادیر بهینه ورودی های LogisticRegression را با استفاده از RandomizedSearchCV بدهست آورده و مجددا با این مقادیر طبقه بندی را انجام می دهیم. الگوریتم جستجوی تصادفی یک الگوریتم بهینه سازی است و در اینجا برای بهینه سازی هایپر پارامتر های LogisticRegression از from sklearn.model_selection import SGDClassifier استفاده می شود. در اینجا با استفاده از RandomizedSearchCV اجرا می شود. ورودی های RandomizedSearchCV به شرح زیر است:

: تخمین گر مورد استفاده برای دیتا (در اینجا مدل های LogisticRegression یا

(SGDClassifier

: مواردی که میخواهیم بهینه شوند را بصورت لیست یا دیکشنری در این param_distributions ورودی، وارد می کنیم. (مانند ماتریس تعداد دور آموزش، نرخ یادگیری، بهترین الگوریتم برای آموزش مدل، وزن اولیه و ...)

: استراتژی ارزیابی عملکرد مدل که در اینجا معیار Accuracy مورد توجه است.

```
[63] #optimize hyperparameter using random search
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
from scipy.stats import uniform

# Set the random seed
np.random.seed(64)

# Define hyperparameter distributions
param_dist = {
    'max_iter': randint(100, 1000),
    'C': [0.1, 1, 10],
    'class_weight': [{0: w, 1: 1 - w} for w in uniform(loc=0, scale=1).rvs(10)],
    'solver': ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga']
}

# Perform random search
random_search = RandomizedSearchCV(model, param_dist, scoring='accuracy', random_state=64)
random_search.fit(x_train, y_train)

# Best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

Best Hyperparameters: {'C': 10, 'class_weight': {0: 0.3790985254337701, 1: 0.6209014745662299}, 'max_iter': 725, 'solver': 'saga'}
```

تعداد دفعات تکرار ۷۲۵ است و وزن اولیه و الگوریتم بهینه و قدرت تنظیم بهینه در قطعه کد بالا قابل مشاهده است.

```
#LogisticRegression after hyperparameter optimization
from sklearn.linear_model import LogisticRegression
# Define class weights
class_weights = {0: 0.3790985254337701, 1: 0.6209014745662299}
# Initialize the Logistic Regression model with class weights
NEW_model = LogisticRegression(solver='saga', max_iter=725, class_weight=class_weights, C=10, random_state=64)
# Fit the model
NEW_model.fit(x_train, y_train)
# Make predictions
NEW_model.predict(x_test)
# Compare predictions with true labels
NEW_model.predict(x_test), y_test

(array([2, 1, 0, 1, 0, 1, 2, 0, 1, 3, 3, 2, 3, 1, 3, 3, 0, 1, 3, 1, 0, 1,
       3, 0, 0, 3, 1, 1, 2, 2, 3, 2, 3, 1, 0, 3, 1, 3, 0, 3, 0, 1, 1, 1,
       0, 3, 1, 2, 3, 3, 2, 0, 3, 0, 2, 1, 2, 2, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 3, 3, 2, 3, 0, 3, 3, 1, 3, 1, 1, 3, 2, 3, 0, 2, 0, 1, 2, 2, 1,
       1, 3, 1, 0, 2, 3, 2, 1, 1, 0, 3, 1, 1, 2, 1, 2, 0, 2, 1, 3, 0, 1,
       3, 3, 2, 0, 2, 2, 3, 0, 2, 3, 2, 0, 0, 2, 0, 3, 0, 2, 2, 0, 0, 3,
       1, 0, 3, 1, 2, 3, 1, 2, 2, 0, 1, 0, 1, 2, 2, 2, 3, 0, 3, 3, 3,
       0, 3, 0, 3, 0, 1, 3, 1, 2, 3, 1, 2, 1, 3, 3, 1, 1, 0, 2, 0, 2, 0,
       0, 1, 3, 1, 1, 3, 0, 0, 2, 2, 3, 0, 2, 2, 0, 0, 2, 1, 0, 3, 1, 3, 1,
       3, 3]), array([2, 1, 0, 1, 0, 1, 2, 0, 1, 3, 3, 2, 3, 1, 3, 3, 0, 1, 3, 1, 0, 0,
       3, 0, 0, 3, 1, 1, 2, 2, 3, 2, 3, 1, 0, 3, 1, 0, 0, 3, 0, 1, 1, 1,
       0, 3, 1, 2, 3, 3, 2, 0, 3, 0, 2, 1, 1, 2, 1, 0, 0, 0, 1, 1, 0,
       1, 3, 3, 2, 2, 0, 3, 3, 1, 3, 1, 1, 3, 2, 3, 0, 2, 0, 1, 2, 2, 1,
       1, 3, 1, 0, 2, 3, 2, 1, 1, 0, 3, 1, 1, 2, 1, 2, 0, 2, 3, 0, 3, 0, 1,
```

دقت آموزش و ارزیابی را قبل و بعد از بهینه سازی پارامترها در طبقه بند LogisticRegression

در جدول زیر می بینیم:

LogisticRegression		
	Befor optimizing parameters	After optimizing parameters
Train	96.125%	96.25%
Test	93%	93.5%

با توجه به اینکه دیتاست اولیه چالش زیادی نداشت، شاهد بهبود چشمگیری در دقت آموزش و ارزیابی قبل و بعد از بهینه سازی نیستیم.

:SGDClassifier

حال به طبقه بندی با استفاده از SGDClassifier می پردازیم. این طبقه بند یک مدل یادگیری براساس گرادیان نزولی را پیاده سازی می کند.

SGDClassifier ورودی هایی به شرح زیر دریافت می کند:

Loss: تابع اتلاف را مشخص می کند. بصورت پیشفرض رو hinge است.

Alpha: ثابتی که هرچه بیشتر باشد قدرت تنظیم بیشتر است. بصورت پیشفرض 0.0001 است.

max_iter: حداقل تعداد دور آموزش که بصورت پیشفرض 1000 است.

learning_rate: نرخ یادگیری را مشخص می کند و به صورت پیشفرض روی 'optimal' است.

eta0: نرخ یادگیری اولیه را نشان می دهد و به صورت پیشفرض صفر است.

پیش تر داده ها را به آموزش و ارزیابی تقسیم کرده ایم.

جز در مورد تابع loss که آن را روی log_loss قرار می دهیم و میخواهیم با استفاده از کراس آنتروپی طبقه بندی انجام شود، ابتدا با استفاده از مقادیر پیشفرض طبقه بندی SGD را انجام می دهیم:

```
#SGDClassifier()
SGDModel = SGDClassifier(loss='log_loss', random_state=64)
SGDModel.fit(x_train, y_train)
```

```
SGDClassifier
SGDClassifier(loss='log_loss', random_state=64)
```

حال مقادیر بهینه ورودی های SGDClassifier را با استفاده از RandomizedSearchCV بدست آورده و مجددا با این مقادیر طبقه بندی را انجام می دهیم:

```
#optimize hyperparameter using random search
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

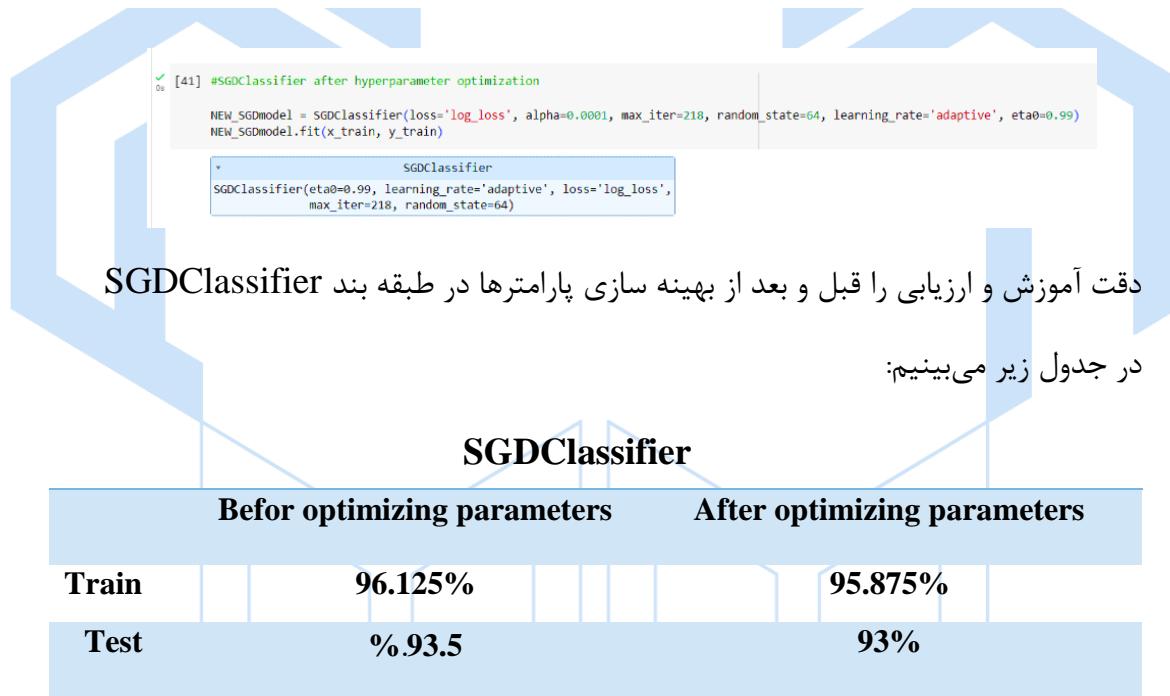
# Define hyperparameter
np.random.seed(64)
param_dist = {
    'alpha': [0.0001, 0.001, 0.01, 0.1, 10], # Regularization parameter
    'max_iter': randint(100, 1000), # Maximum number of iterations
    'learning_rate': ['optimal', 'constant', 'adaptive'], # Learning rate schedule
    'eta0' :[0.99, 0.95, 0.91, 0.87, 0.83] #initial learning rate
}

# Perform random search
random_search = RandomizedSearchCV(SGDmodel, param_dist, scoring='accuracy')
random_search.fit(x_train, y_train)

# Best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

Best Hyperparameters: {'alpha': 0.0001, 'eta0': 0.99, 'learning_rate': 'adaptive', 'max_iter': 218}
```

همانطور که از نتیجه کد فوق پیداست، بیشترین تعداد دور آموزش را ۲۱۸ و نرخ یادگیری را ‘۰.۹۹’ و نرخ یادگیری اولیه را در نظر گرفته و با این مقادیر طبقه بندی را انجام می دهیم:



با توجه به جدول بالا پیداست الگوریتم جستجوی تصادفی دقت را بهبود نبخشیده است. اما با تعداد دور آموزش بسیار کمتری نسبت به حالت پیشفرض به دقت مطبوبی رسیده است.

۱-۴

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

برای رسم مرز و نواحی تصمیم گیری از `mlxtend.plotting import plot_decision_regions` استفاده می کنیم.

```
from mlxtend.plotting import plot_decision_regions
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# pca for reducing features
pca = PCA(n_components=2)
D = pca.fit_transform(X)

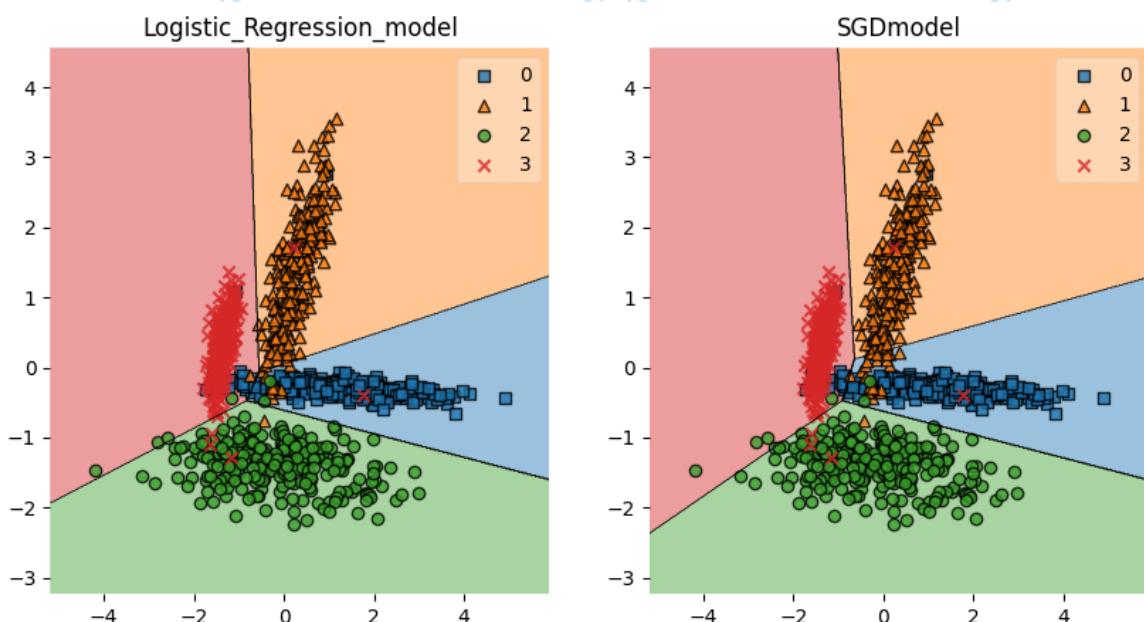
# Plot decision regions for the first model (NEW_model)
clf = NEW_model
clf.fit(D, y)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plot_decision_regions(D, y, clf=clf)
plt.title('Logistic_Regression_model')

# Plot decision regions for the second model (SGDmodel)
clf1 = SGDmodel
clf1.fit(D, y)
plt.subplot(1, 2, 2)
plot_decision_regions(D, y, clf=clf1)
plt.title('SGDmodel')

plt.show()
```

رسم مرز تصمیم گیری برای دیتا، با بیش از ۲ ویژگی امکان پذیر نبود، به همین دلیل با استفاده از کتابخانه `from sklearn.decomposition import PCA` تعداد ویژگی را به ۲ کاهش داده و نواحی و مرز تصمیم گیری را ترسیم می کنیم:

مرز و نواحی تصمیم گیری برای مدل های `SGDClassifier` و `LogisticRegression` در شکل زیر ترسیم شده است:



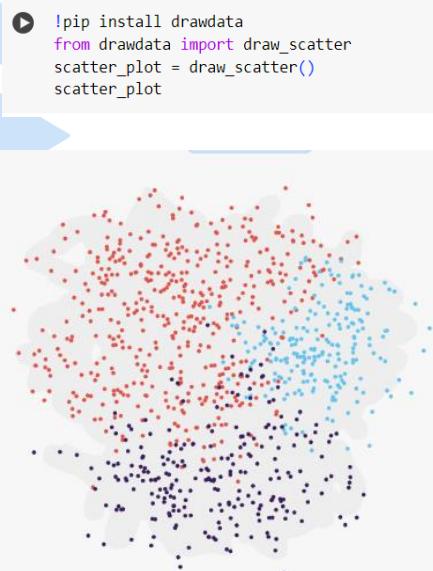
همانطور که در شکل بالا پیداست، داده ها در هر دو مدل طبقه بندی تا حد قابل قبولی طبقه بندی شده اند. برخی داده ها از کلاس متفاوت در کلاس دیگری قرار گرفته اند که با رنگ متمایز قابل مشاهده اند.

۱-۵

فرآیندی مشابه قسمت ۱-۲ را با تعداد کلاس و ویژگی دلخواه؛ اما با استفاده از ابزار drawdata تکرار کنید. قسمت های ۱-۳ و ۱-۴ را برای این داده های جدید تکرار و نتایج را به صورتی مناسب نشان دهید.

ابتدا drawdata را نصب می کنیم و سپس داده ها را در سه کلاس a, b, c، بطور دلخواه ترسیم می کنیم:

```
▶ !pip install drawdata  
from drawdata import draw_scatter  
scatter_plot = draw_scatter()  
scatter_plot
```



سپس داده رسم شده را بصورت CSV دانلود و در درایو بارگذاری می کنیم و با دستور gdown آن را در گوگل کولب برای مراحل بعد فراخوانی می کنیم. دیتای ایجاد شده به صورت زیر است:

	x	y	z	
0	334.163308	460.503200	c	
1	297.112122	377.037338	c	
2	293.701649	398.825620	c	
3	320.289971	413.079830	c	
4	330.419033	371.602652	c	
...	
911	330.539597	173.853976	a	
912	296.319831	144.407915	a	
913	298.399113	147.862162	a	
914	297.976774	174.090900	a	
915	306.265947	173.489178	a	
916	rows × 3 columns			

مرحله ۳-۱ را مجددا برای دیتای این قسمت پیاده سازی می کنیم:

با استفاده از `from sklearn.preprocessing import LabelEncoder` لیبل ها را به عدد های

0,1,2 تبدیل می کنیم:

حال طبقه بند LogisticRegression را ابتدا بدون بهینه سازی های پر امترها و سپس با استفاده از الگوریتم جستجوی تصادفی که پیشتر به آن پرداختیم، اجرا و نتایج را گزارش می‌کنیم:

در دفترچه Q1 قسمت ۱-۵ مراحل تقسیم داده به آموزش و ارزیابی و ایجاد مدل های SGDClassifier و LogisticRegression و بهینه سازی آنها قابل مشاهده است. در اینجا به ذکر نتایج دقیق مدل ها قبل و بعد از بهینه سازی در هر طبقه بند اکتفا می کنیم:

```
✓ [144] #from sklearn.model_selection import RandomizedSearchCV
      from scipy.stats import randint
      from scipy.stats import uniform

      # Set the random seed
      np.random.seed(64)

      # Define hyperparameter distributions
      param_dist = {
          'max_iter': randint(100, 1000),
          'C': [0.1, 1, 10],
          'class_weight': [(0: w, 1: 1 - w) for w in uniform(loc=0, scale=1).rvs(10)],
          'solver': ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga']
      }

      # Perform random search
      random_search = RandomizedSearchCV(model1, param_dist, scoring='accuracy', random_state=64)
      random_search.fit(x1_train, y1_train)
      # Best hyperparameters
      print("Best Hyperparameters:", random_search.best_params_)
```

مقداری بهینه شده در پایین قطعه کد بالا قابل مشاهده است. از این مقادیر برای مقدار دهی هایپر پارامترها استفاده می کنیم.

Logistic Regression

	Befor optimizing parameters	After optimizing parameters
Train	85.25%	85.25%
Test	83.69%	83.69%

همانطور که می‌بینیم تغییری قیا، و بعد از بینه سازی در دقت مدل بوجود نیامد.

برای بهینه سازی طبقه بند **SGDClassifier** داریم:

```
#optimize hyperparameter using random search
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Define hyperparameter
np.random.seed(64)
param_dist = {
    'alpha': [0.0001, 0.001, 0.01, 0.1, 10], # Regularization parameter
    'max_iter': randint(100, 1000), # Maximum number of iterations
    'learning_rate': ['optimal', 'constant', 'adaptive'], # Learning rate schedule
    'eta0': [0.99, 0.95, 0.91, 0.87, 0.83] #initial learning rate
}

# Perform random search
random_search = RandomizedSearchCV(SGDmodel1, param_dist, scoring='accuracy', random_state=64)
random_search.fit(x_train, y_train)

# Best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

Best Hyperparameters: {'alpha': 0.0001, 'eta0': 0.99, 'learning_rate': 'adaptive', 'max_iter': 218}
```

مقادیر بهینه شده در پایین قطعه کد بالا قابل مشاهده است. و از این مقادیر برای طبقه بندی **SGDClassifier** استفاده شده است:

SGDClassifier

	Befor optimizing parameters	After optimizing parameters
Train	67.62%	85.92%
Test	71.19%	83.15%

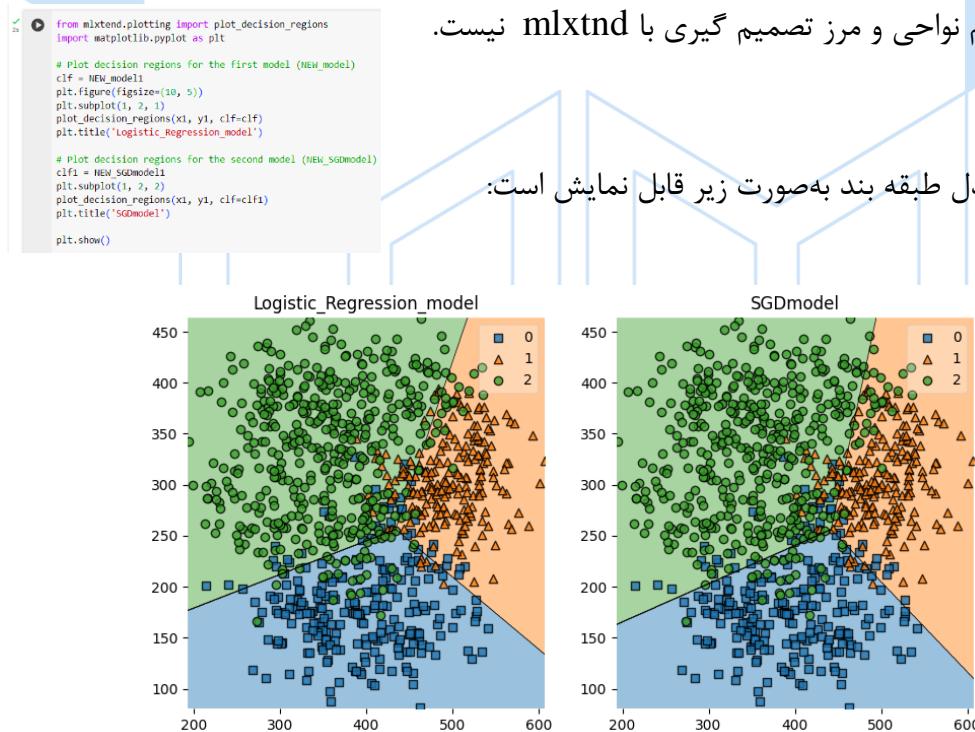
همانطور که از جدول فوق پیداست، بهینه سازی پارامترها نتایج را تا حد خوبی بهبود بخشید.

حال به رسم نواحی و مرزهای تصمیم گیری می پردازیم:

مانند قسمت ۱-۵ عمل می کنیم با این تفاوت که اینجا دیتا دارای دو ویژگی است و نیازی به کاهش

ویژگی برای رسم نواحی و مرز تصمیم گیری با **mlxtnd** نیست.

نتایج برای دو مدل طبقه بند به صورت زیر قابل نمایش است:



سوال دوم

۲-۱

با مراجعه به صفحه دیتاست CWRU Bearing با یک دیتاست مربوط به حوزه تشخیص عیب آشنا شوید. با جستجوی آن در اینترنت و مقالات، توضیحاتی از اهداف، ویژگی ها و حالت های مختلف این دیتاست را در ادامه، ابتدا به صفحه داده های سالم مراجعه کنید و داده های کلاس سالم Normal_X را دریافت کنید. سپس به صفحه داده های عیب در حالت 12k مراجعه کنید و داده های کلاس عیب (IR007_X) را دریافت کنید.

مجموعه داده CWRU Bearing به عنوان مرجع استاندارد و مجموعه داده پایه برای اعتبارسنجی عملکرد الگوریتم های مختلف یادگیری ماشین (ML) و یادگیری عمیق (DL) شناخته می شود. دیتاست یکی از دیتاست های شناخته شده و پرکاربرد در حوزه تشخیص عیب یاتاقان های Bearing CWRU غلتی است که توسط دانشگاه Case Western Reserve ایجاد شده است.

مجموعه داده های یاتاقان CWRU با استفاده از آزمایشی شامل یک موتور القایی الکتریکی ۲ اسب بخار، یک مبدل گشتاور، یک دینامومتر و الکترونیک کنترل به دست آمد. مجموعه داده شامل عیب های مختلفی است. مجموعه داده شامل ۱۶۱ رکورد است که در چهار کلاس سازماندهی شده اند:

- 48k normal-baseline
- 48k drive-endfault
- 12k drive-endfault
- 12k fan-end fault

هر کلاس فوق به مجموعه داده هایی برای انواع خطاهای مختلف تقسیم می شود:

- Ball bearing (B) fault
- Inner-race fault
- Outer-race faults, classified based on location relative to the load zone:
 - ‘Centered’ (fault in the 6.00 o’clock position)
 - ‘Orthogonal’ (3.00 o’clock)
 - ‘Opposite’ (12.00 o’clock)

داده ها در محیط MATLAB مورد پردازش قرار گرفتند و تمامی فایل های داده به دست آمده با فرمت (.mat) ذخیره شدند. هر فایل شامل یک یا چند مجموعه از داده های Fan-، Drive-End (DE) و شتاب صفحه پایه (BA) است. جمع آوری داده ها شامل دو فرکانس نمونه گیری متفاوت ۱۲ کیلوهرتز و ۴۸ کیلوهرتز بود.

داده های ارتعاش برای بارهای موتور در محدوده ۰ تا ۳ اسب بخار، با سرعت های موتور بین ۱۷۲۰ و ۱۷۹۷ RPM ثبت شد. برای آزمایش های بلبرینگ انتهای درایو، داده ها هم در ۱۲k و هم ۴۸k جمع آوری شدند. داده های انتهای فن با نرخ ۱۲ هزار نمونه در ثانیه جمع آوری شد. جمع آوری داده های پایه عادی دارای فرکانس نمونه برداری ۴۸ هزار نمونه در ثانیه بود.

حرف اول نام فایل های داده، موقعیت خطا را نشان می دهد، سه عدد بعدی نشان دهنده قطر عیب و آخرین عدد نشان دهنده بارهای تحمل کننده است. به عنوان مثال، فایل داده "B007_0" حاوی داده های خطای بلبرینگ با قطر عیب ۰,۰۰۷ است که تحت بار موتور ۰ اسب بخار کار می کند.

باتوجه به دور قم آخر شماره دانشجویی بنده (۶۴)، دیتای کلاس سالم Normal_0 که در دور موتور ۱۷۹۷RPM است. دیتای IR007_0 نیز به عنوان کلاس عیب در نظر گرفته شد.

دیتاست مذکور همانطور که گفته شد به فرمت mat. بود که ابتدا آن را بصورت فایل csv. درآورده و آن را در گوگل درایو آپلود کرده، سپس با دستور gdown آن را در محیط گوگل کولب بارگزاری کردم.

در اینجا از X097_DE_time برای کلاس نرمال و از X105_DE_time برای کلاس عیب استفاده شده است.

ابتدا داده های پوج را با دستور isnull در صورت وجود از هر کلاس با dropna جدا و مابقی مقادیر آنها را با reset_index برمی گردانیم.

```

00  ✓  Normal = pd.read_csv('/content/Normal.csv')

Normal=Normal.X097_DE_time
# Show the number of null values in each column
null_counts = Normal.isnull().sum()
print("Number of null values in each column:")
print(null_counts)

# Remove rows with null values
Normal = Normal.dropna()

# Reset the index after removing rows
Normal = Normal.reset_index(drop=True)

# Now, the DataFrame 'df' contains no rows with null values
print("DataFrame after removing rows with null values:")
print(Normal)

```

```

00  ✓  Fault = pd.read_csv('/content/Fault.csv')
Fault=Fault.X105_DE_time
# Show the number of null values in each column
null_counts = Fault.isnull().sum()
print("Number of null values in each column:")
print(null_counts)

# Remove rows with null values
Fault = Fault.dropna()

# Reset the index after removing rows
Fault = Fault.reset_index(drop=True)

# Now, the DataFrame 'df' contains no rows with null values
print("DataFrame after removing rows with null values:")
print(Fault)

```

```

Number of null values in each column:
0
DataFrame after removing rows with null values:
0      -0.083004
1      -0.195734
2       0.233419
3      0.103958
4     -0.181115
...
121260   0.324545
121261   0.142456
121262  -0.316424
121263  -0.063675

```

از هر کلاس M نمونه با طول N جدا کنید (M حداقل ۱۰۰ و N حداقل ۲۰۰ باشد). یک ماتریس از داده های هر دو کلاس به همراه برچسب مربوطه تشکیل دهید. می توانید پنجره ای به طول N در نظر بگیرید و در نهایت یک ماتریس $N \times M$ از داده های هر کلاس استخراج کنید.

برای جدا کردن نمونه ها از هر کلاس، از یک حلقه for استفاده شده است که در هر تکرار، یک نمونه با طول N از داده های کلاس مربوطه را بر می گرداند. در این عملیات M نمونه تصادفی از هر کلاس انتخاب می شوند و در ماتریسی قرار می گیرند. (برای اینکه انتخاب رندوم در تکرار بعد ثابت باشد از random.seed(64) استفاده شد.) در نهایت، یک ماتریس داده (X) از ترکیب نمونه های هر دو کلاس خواهیم داشت (ابعاد این ماتریس 200×200 است). و سپس به بر چسب زدن به نمونه ها با استفاده از np.vstack برای کلاس نرمال و np.zeros برای کلاس عیب می پردازیم (با این کار ابعاد ماتریس 200×201 می شود.). الحق داده ها و بر چسب ها با دستور np.vstack انجام می شود.

```

Normal_data = Normal
Fault_data = Fault

# Extract 100 samples with length of 200
sample_length = 200
num_samples = 100

Normal_samples = []
Fault_samples = []

# Extract samples from class Normal
for i in range(num_samples):
    np.random.seed(64)
    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)

# Extract samples from class Fault
for i in range(num_samples):
    np.random.seed(64)
    start_idx = np.random.randint(0, len(Fault_data) - sample_length + 1)
    sample = Fault_data[start_idx:start_idx + sample_length]
    Fault_samples.append(sample)

# Convert lists of samples to numpy arrays
Normal_samples = np.array(Normal_samples)
Fault_samples = np.array(Fault_samples)

# Create labels for the samples
Normal_labels = np.ones((num_samples, 1)) # Assuming class Normal is labeled as 1
Fault_labels = np.zeros((num_samples, 1)) # Assuming class Fault is labeled as 0

# Concatenate the data and labels for both classes
data_matrix = np.vstack((Normal_samples, Fault_samples))
print("Data Matrix:")
print(data_matrix)
print(data_matrix.shape)
labels = np.vstack((Normal_labels, Fault_labels))
#print("Labels")
#print(labels)
#print(labels.shape)
main_matrix = np.hstack((data_matrix, labels))
print("Data & Label Matrix:")
print(main_matrix)

```

ب

در مورد اهمیت استخراج ویژگی در یادگیری ماشین توضیحاتی بنویسید. سپس، با استفاده از حداقل ۸ عدد از روش های ذکر شده در جدول زیر ویژگی های دیتابست قسمت ۲-آ استخراج کنید و یک دیتابست جدید تشکیل دهید.

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

استخراج ویژگی (Feature Extraction) یکی از مهمترین مراحل در یادگیری ماشین است. داده های خام عموماً دارای ابعاد بالایی هستند که میتواند پردازش را پیچیده و زمان برکند. استخراج ویژگی های مناسب میتواند ابعاد داده را کاهش دهد و تنها اطلاعات مفید را حفظ نماید. استخراج ویژگی های مفید سبب می شود الگوریتم یادگیری روابط والگوهای داده را به خوبی تشخیص دهد. همچنین استخراج ویژگی های مناسب می تواند نویز و داده های اضافی را که برای مدل مفید نیستند حذف کند.

در اینجا ویژگی های زیربا توجه به روابط آنها در جدول بالا استخراج می شوند:

mean, Standard Deviation, Peak, Root Mean Square, Crest Factor, Peak to Peak,
Absolute Mean, Impulse Factor

سپس با دستور pd.DataFrame دیتای جدید با ویژگی های استخراج شده را نمایش می دهیم:

```
#create features
mean_values = np.mean(data_matrix, axis=1)
std_dev_values = np.std(data_matrix, axis=1)
peak_values = np.max(np.abs(data_matrix), axis=1)
rms_value = np.sqrt(np.mean(data_matrix**2, axis=1))
crest_factor = peak_values / rms_value
peak_to_peak_value=np.max(data_matrix, axis=1)-np.min(data_matrix, axis=1)
Abs_Mean_value=np.mean(np.abs(data_matrix), axis=1)
Impulse_factor=peak_values/Abs_Mean_value

# Concatenate mean and standard deviation as features
features = np.column_stack((mean_values, std_dev_values, peak_values, rms_value, crest_factor, peak_to_peak_value, Abs_Mean_value, Impulse_factor))

# Create DataFrame
Data = pd.DataFrame(features, columns=['Mean', 'Standard Deviation', 'Peak', 'RMS', 'Crest Factor', 'Peak to Peak', 'Absolute Mean', 'Impulse Factor'])

print(Data)

      Mean  Standard Deviation     Peak      RMS  Crest Factor \
0  0.016739          0.086732  0.292148  0.088333  2.288487
1  0.016739          0.086732  0.292148  0.088333  2.288487
2  0.016739          0.086732  0.292148  0.088333  2.288487
3  0.016739          0.086732  0.292148  0.088333  2.288487
4  0.016739          0.086732  0.292148  0.088333  2.288487
...
195  0.013727          0.266337  0.902977  0.266690  3.385863
196  0.013727          0.266337  0.902977  0.266690  3.385863
197  0.013727          0.266337  0.902977  0.266690  3.385863
198  0.013727          0.266337  0.902977  0.266690  3.385863
199  0.013727          0.266337  0.902977  0.266690  3.385863

      Peak to Peak  Absolute Mean  Impulse Factor
0       0.386982        0.075379   2.681759
1       0.386982        0.075379   2.681759
2       0.386982        0.075379   2.681759
3       0.386982        0.075379   2.681759
4       0.386982        0.075379   2.681759
...
195    1.679254        0.196583   4.595240
196    1.679254        0.196583   4.595240
197    1.679254        0.196583   4.595240
198    1.679254        0.196583   4.595240
199    1.679254        0.196583   4.595240
```

[200 rows x 8 columns]

ج

ضمن توضیح اهمیت فرآیند برزدن (مخلوط کردن)، داده ها را در صورت امکان مخلوط کرده و با نسبت تقسیم دلخواه و معقول به دو بخش «ارزیابی» و «آموزش» تقسیم کنید.

فرآیند برزدن (مخلوط کردن) داده های یک دیتاست در یادگیری ما شین بسیار مهم است. اگر تمام نمونه های یک کلاس در ابتدای دیتاست قرار گرفته باشند، الگوریتم ممکن است فقط از آن قسمت دیتاست یاد بگیرد و نتواند روی دیگر قسمت ها عملکرد مطلوبی داشته باشد. برزدن داده ها باعث می شود مدل از تمام داده های موجود در دیتاست یاد بگیرد و به یک مدل با قابلیت تعمیم پذیری بالاتری دست یابیم. زمانی که داده ها مخلوط می شوند، تقسیم آنها به دسته آموزش و ارزیابی به صورت تصادفی انجام می شود و سبب معرفی ارزیابی عملکرد واقعی تری از مدل می شود.

در اینجا ماتریس شامل دیتا همراه با برچسب را با دستور `np.random.shuffle` مخلوط می کنیم. برای تکرار پذیری نتایج از `random.seed(64)` استفاده شد که باعث می شود نتایج آموزش و ارزیابی مدل در هر اجرا یکسان باشد.

```
#shuffling Data
Labeled_Data=np.hstack((Data, labels))
np.random.seed(64)
np.random.shuffle(Labeled_Data)
print(Labeled_Data)

[[0.0167393  0.08673215  0.20214831 ...  0.075379   2.68175906 1.
 [0.01372739  0.26633675  0.90297689 ...  0.19650265  4.59524026 0.
 [0.0167393  0.08673215  0.20214831 ...  0.075379   2.68175906 1.
 ...
 [0.01372739  0.26633675  0.90297689 ...  0.19650265  4.59524026 0.
 [0.01372739  0.26633675  0.90297689 ...  0.19650265  4.59524026 0.
 [0.01372739  0.26633675  0.90297689 ...  0.19650265  4.59524026 0.]]
```

داده ها را با دستور `pd.DataFrame` به صورت دیتافریم درآورده، سپس آنها را با نسبت ۸۰ به ۲۰ به دو دسته آموزش و ارزیابی و با دستور `train_test_split` از کتابخانه `sklearn.model_selection` تقسیم می کنیم:

```
#Train & Test
from sklearn.model_selection import train_test_split
Labeled_Data=pd.DataFrame(np.hstack((Data, labels)), columns=['Mean', 'Standard Deviation','Peak', 'RMS', 'Crest Factor', 'Peak to Peak', 'Absolute Mean', 'Impulse Factor', 'Target'])
X = Labeled_Data[['Mean', 'Standard Deviation','Peak', 'RMS', 'Crest Factor', 'Peak to Peak', 'Absolute Mean', 'Impulse Factor']].values
y = Labeled_Data[['Target']].values

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=64)
x_train.shape, x_test.shape, y_train.shape, y_test.shape

((160, 8), (40, 8), (160, 1), (40, 1))
```

حداقل دو روش برای نرمال سازی داده ها 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؟

هدف از نرمال سازی داده ها، تبدیل مقادیر ویژگی ها به یک محدوده مشخص است تا از تأثیر بیش از حد ویژگی های با مقادیر بزرگ بر روی مدل جلوگیری شود. بسیاری از الگوریتم های یادگیری ما شین، مانند رگرسیون لجستیک عملکرد بهتری روی داده های نرمال شده نشان می دهند. نرمال سازی داده سبب همگرایی سریعتر و کاهش احتمال گیر افتادن الگوریتم در کمینه های محلی می شود.

دو روش رایج برای نرمال سازی داده ها معرفی می شوند:

1. MinMaxScaler

نرمال سازی حداقل_حداکثر، همه ویژگی ها را به یک بازه که معمولا $[0, 1]$ است، تبدیل می کند. این تبدیل به صورت زیر انجام می شود:

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

نرمال سازی حداقل_حداکثر در پایتون با استفاده از کتابخانه `from sklearn.preprocessing` امکان پذیر است. این کار با تعریف `MinMaxScaler()` و با دستور `scaler.fit_transform(x)` انجام می شود.

2. StandardScaler

نرمال سازی استاندارد مقادیر ویژگی ها را به یک توزیع نرمال استاندارد با میانگین صفر و واریانس یک تبدیل می کند. اگر میانگین داده های اصلی برابر با μ و انحراف معیار آنها نیز σ باشد، داده استاندارد شده z از رابطه زیر بدست می آید:

$$Z = \frac{x - \mu}{\sigma}$$

نرمال سازی استاندارد در پایتون با استفاده از کتابخانه `from sklearn.preprocessing` و `StandardScaler()` امکان پذیر است. این کار با تعریف `StandardScaler()` و با دستور `scaler.fit_transform(x)` انجام می شود.

در این سوال از نرمال سازی حداقل_حداکثر استفاده شده است:

```

#Normalizing Data
from sklearn.preprocessing import MinMaxScaler
# Create a MinMaxScaler instance
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform it
x_train_normalized = scaler.fit_transform(x_train)

# Transform the test data using the same scaler
x_test_normalized = scaler.transform(x_test)

# Check the shapes of the normalized data
x_train_normalized.shape, x_test_normalized.shape, y_train.shape, y_test.shape
((160, 8), (40, 8), (160, 1), (40, 1))

```

داده های ارزیابی نیز به همان روشی که داده های آموزش نرمال سازی شده اند، نرمالیزه می شوند. این مسئله به دلیل حفظ روابط و الگوهای یادگرفته شده توسط مدل انجام می شود. نرمال سازی داده های ارزیابی کمک می کند تا تطابق بین دو مجموعه داده آموزش و ارزیابی حفظ شود که در به دست آوردن نتایج قابل قبول اهمیت دارد.

۲-۳

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

در این قسمت با توجه به توضیحات ویدوی تدریسیاری جلسه ۸ بدون استفاده از کتابخانه های اماده و صرفا با تعریف توابع بصورت def و نوشتن مراحل یادگیری شامل تابع سیگموئید، تابع اتلاف BCE، فرآیند گرادیان نزولی و شاخص ارزیابی مدل طبقه بند رگرسیون لجستیک را پیاده سازی می کنیم:

در قسمت های قبل دیتاست را به دو دسته آموزش و ارزیابی تقسیم کرده ایم و آنها را نرمال سازی نموده ایم. همچنین ویژگی های دیتاست را استخراج کرده ایم. ماتریس دیتا 8×200 و بردار برچسب ها 1×200 است.

معیار ارزیابی صحت و بازیابی (recall) مورد بررسی قرار گرفتند. با معیار صحت در ویدئو تدریسیاری آشنا شدیم. معیار بازیابی (recall) حاصل تقسیم تعداد مواردی که توسط مدل درست تشخیص داده اند شده بر تعداد کل مواردی که توسط مدل ایجاد شده اند.

```
[86] #Logistic Regression (from Scratch)

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

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

#Binary Cross Entropy (BCE)

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

#Gradient

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

#Gradient Descent

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

#Accuracy

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

#recall
def recall(y, y_hat):
    # Calculate true positives and false negatives
    true_positives = np.sum(np.round(y_hat) == y)
    false_negatives = np.sum(np.round(y_hat) != y)
    recall = true_positives / (true_positives + false_negatives) if (true_positives + false_negatives) != 0 else 0
    return recall
```

اکنون به هایپر پارامتر ها مقدار می دهیم:

```
[38] #hyper parameter
m = 8
np.random.seed(64)
w = np.random.randn(m+1, 1)
print(w.shape)

eta = 0.01
n_epochs = 1000

(9, 1)

[39] x_train = np.hstack((np.ones((len(x_train), 1)), x_train))
x_train.shape

(160, 9)
```

m تعداد ویژگی دیتا و w ماتریس وزن اولیه است که به صورت رندوم انتخاب شده است. به منظور تکرار پذیری نتایج، از np.random.seed(64) استفاده شده است.

حال خطا را در هر ایپاک محاسبه و آن را آرایه ای ذخیره می کنیم و توسط حلقه for این کار برای هر ایپاک تکرار می شود:

```
error_hist = []
for epoch in range(n_epochs):
    # predictions
    y_hat = logistic_regression(x_train, w)

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

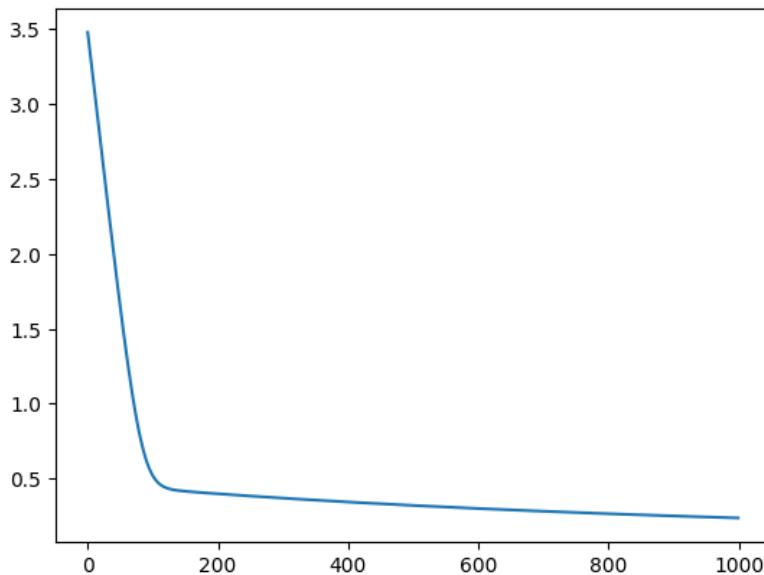
    # gradients
    grads = gradient(x_train, y_train, y_hat)

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

    if (epoch+1) % 100 == 0:
        print(f'Epoch={epoch}, \t E={e:.4f}, \t w={w.T[0]}')
```

Epoch=99, E=0.08297, w=[3.19137323 -1.94052026 -1.39253706 0.96251396 -2.13743419 1.76296947 -3.37457915 -1.16137858 -1.27775833]
Epoch=199, E=0.08117, w=[3.21106257 -1.94010189 -1.39612389 0.94583456 -2.14095274 1.77567897 -3.40505435 -1.16346499 -1.28136334]
Epoch=299, E=0.07945, w=[3.23034622 -1.93969213 -1.39963694 0.92949833 -2.14439892 1.7881258 -3.43490257 -1.16550851 -1.28489546]
Epoch=399, E=0.07779, w=[3.24923989 -1.93929066 -1.40307966 0.91349199 -2.14777553 1.80032017 -3.46414802 -1.16751077 -1.28835745]
Epoch=499, E=0.0762, w=[3.26775839 -1.93889716 -1.40645295 0.89780304 -2.1510852 1.81227169 -3.49281358 -1.16947336 -1.29175191]
Epoch=599, E=0.07467, w=[3.28591574 -1.93851133 -1.40976116 0.88241963 -2.15433044 1.82398948]

نمودار تابع خطا بر حسب ایپاک بصورت زیر است:



همانطور که از شکل بالا پیداست، با افزایش ایپاک خطا کاهش یافته است.

اما بدون استفاده از داده ارزیابی و صرفا با اطلاعات فوق نمی‌توان بطور قطعی درباره عملکرد مدل نظر داد. چرا که امکان دارد مدل over fit شده باشد و به دنبال این مسئله نویز احتمالی موجود در داده‌ها را یاد گرفته باشد و نمودار اتلاف به همین دلیل کاهشی باشد. برای رفع این مسئله باید از داده ارزیابی استفاده کرد. با انجام این کار نمودار اتلاف داده ارزیابی تا جایی همگام با نمودار اتلاف داده آموزش کاهشی است اما از نقطه‌ای که over fit رخ داده است، واگرا می‌شود و این نقطه را می‌توان به عنوان تعداد ایپاک بهینه معروفی کرد.

اکنون مدل را با استفاده دو شاخص ارزیابی گفته شده، می‌سنجیم:

```
[139] #Test  
x_test = np.hstack((np.ones((len(x_test), 1)), x_test))  
x_test.shape  
(40, 9)  
  
[140] #Accuracy  
y_hat = logistic_regression(x_test, w)  
accuracy(y_test, y_hat)  
1.0  
  
[87] #Accuracy  
recall(y_test, y_hat)  
1.0
```

همانطور که پیداست هر دو شاخص ارزیابی عملکرد مدل را ۱۰۰٪ گزارش می‌کنند.

فرآیند آموزش و ارزیابی را با استفاده از یک طبقه بند خطی آماده پایتون `model_linear.sklearn` انجام داده و نتایج را مقایسه کنید. در حالت استفاده از دستورات آماده سایکیت لرن، آیا راهی برای نمایش نمودار تابع اتلاف وجود دارد؟ پیاده سازی کنید.

برای این کار با توجه به توضیحات داده شده در پاسخ سوال اول قسمت ج پیش می‌رویم:

با استفاده از دستورات آمده در کتابخانه `model_linear.sklearn` با استفاده از دو روش `SGDClassifier` و `LogisticRegression` فرآیند آموزش و ارزیابی را انجام می‌دهیم.

ابتدا با استفاده از هایپر پارامتر های پیشفرض کتابخانه و سپس با هایپر پارامترهای بهینه شده توسط الگوریتم جستجوی تصادفی به طبقه بندی می پردازیم:

LogisticRegression

طیقه یندی یا استفاده از مقادیر پیشفرض:

```
#logisticRegression_using libraries
from sklearn.linear_model import LogisticRegression, SGDClassifier
model = LogisticRegression(random_state=0)
model.fit(x_train, y_train)
model.predict(x_test), y_test

# /usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel.
y = column_or_1d(y, warn=True)
(array([0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 1.,
       1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 0.]),
array([1., 0., 1., 0., 1., 1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1.,
       1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 0.]),
```

رمهنه ساز، هابس يا امت ها:

```

⑤ # Optimize hyperparameter using random search
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
from scipy.stats import uniform

# Set the random seed
np.random.seed(64)
# Define hyperparameter distributions
param_dist = {
    "max_iter": randint(100, 1000),
    "C": [0.1, 1, 10],
    "class_weight": [{0: w, 1: 1-w} for w in uniform(loc=0.0, scale=1).rvs(10)],
    "solver": ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga']
}

# Perform random search
random_search = RandomizedSearchCV(param_distributions, param_dist, scoring='accuracy', random_state=42)
random_search.fit(x_train, y_train)

# Best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

```

مقدیر بھینہ:

```
y = column_stack((y, weight))  
Best Hyperparameters: {'C': 0.1, 'class_weight': {0: 0.3133373611975704, 1: 0.6866626388024296}, 'max_iter': 219, 'solver': 'newton-cg'}
```

طبقه بندی با استفاده از مقادیر بهینه:

```
In [174]: # logisticRegression after hyperparameter optimization
from sklearn.linear_model import LogisticRegression
# Define class weights
class_weights = {0: 0.3133373011975704, 1: 0.6866626388824296}
# Initialize the logistic regression model with class weights
NEW_model = LogisticRegression(solver='newton-cg', max_iter=219, class_weight=class_weights, C=0.1, random_state=64)
# Fit the model
NEW_model.fit(x_train, y_train)
# Make predictions
NEW_model.predict(x_test)
# Compute predictions with true labels
NEW_model.predict(x_test), y_test
NEW_model.score(x_train, y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
y = column_or_1d(y, warn=True)
1.0
```

```
Out[174]: 1.0
```

LogisticRegression

Befor optimizing parameters After optimizing parameters

Train

100%

100%

Test

100%

100%

SGDClassifier

طبقه بندی با استفاده از مقادیر پیشفرض:

```
[ ] #SGDClassifier
SGDModel = SGDClassifier(loss='log_loss', random_state=64)
SGDModel.fit(x_train, y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
y = column_or_1d(y, warn=True)
SGDClassifier(loss='log_loss', random_state=64)
```

```
[ ] SGDModel.score(x_train, y_train)
1.0
```

```
Out[1]: 1.0
```

بهینه سازی هایپر پارامترها و مقادیر بهینه:

```
[ ] #optimize hyperparameter using random search
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Define hyperparameter
np.random.seed(64)
param_distr = {
    'alpha': [0.0001, 0.001, 0.01, 0.1, 10], # Regularization parameter
    'max_iter': randint(100, 1000), # Maximum number of iterations
    'learning_rate': ['optimal', 'constant', 'adaptive'], # Learning rate schedule
    'eta0' : [0.09, 0.95, 0.91, 0.87, 0.83] #Initial learning rate
}

# Perform random search
random_search = RandomizedSearchCV(SGDModel, param_distr, scoring='accuracy', random_state=64)
random_search.fit(x_train, y_train)

# Best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

Out[1]: Best Hyperparameters: {'alpha': 10, 'eta0': 0.95, 'learning_rate': 'optimal', 'max_iter': 919}
```

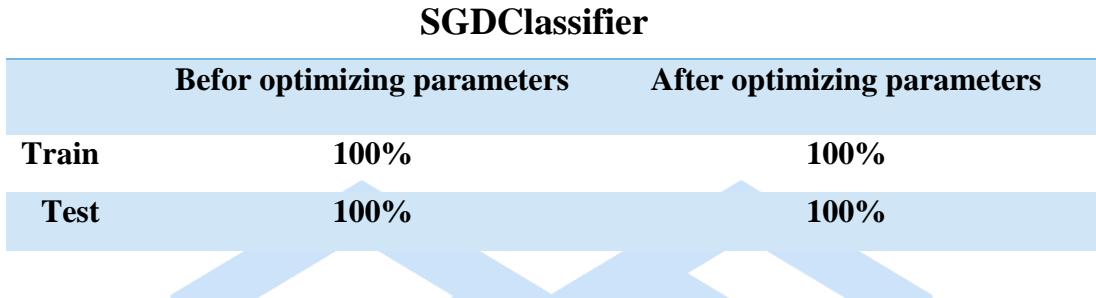
طبقه بندی با استفاده از مقادیر بهینه:

```
[33]: NEW_SGDModel = SGDClassifier(loss='log_loss', alpha=10, max_iter=919, random_state=64, learning_rate='optimal', eta0=0.95)
NEW_SGDModel.fit(x_train, y_train)
NEW_SGDModel.score(x_train, y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
y = column_or_1d(y, warn=True)
1.0
```

```
Out[33]: 1.0
```

نتایج به صورت زیر قابل نمایش است:



همانطور که از نتایج پیداست در هر دو مدل طبقه بند هم قبل و هم بعد از بهینه سازی هایپر پارامترها به عملکرد ۱۰۰٪ رسیدیم.

برای رسم تابع اتلاف ابتدا مقدار تابع اتلاف کراس انتروپی را با استفاده از from sklearn.metrics import log_loss محاسبه و سپس با دستور زیر تابع را رسم می کنیم:

```
from sklearn.metrics import log_loss
error_hist = []

for epoch in range(n_epochs):
    # predictions
    y_hat = model.predict_log_proba(x_test)
    # loss
    e = log_loss(y_test, y_hat)
    error_hist.append(e)

    if (epoch+1) % 100 == 0:
        print(f'Epoch={epoch}, \t E={e:.4}, \t w={w.T[0]}')

plt.plot(error_hist)
```

سوال سوم

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

۳-۱

ابتدا هیئت مپ ماتریس همبستگی و هیستوگرام پراکندگی ویژگی ها را رسم و تحلیل کنید.

دیتاست مذکور را دانلود و در گوگل درایو اپلود کرده و در محیط گوگل کولب با دستور gdown فراخوانی می کنیم. دیتاست را بصورت زیر در حالت دیتا فریم نمایش می دهیم:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00 +0200	Mostly Cloudy	rain	9.377778	0.89	3.9284	204.0	14.9069	0.0	1015.94	Partly cloudy throughout the day.	
3	2006-04-01 03:00:00 +0200	Partly Cloudy	rain	8.298889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.
...
96448	2016-09-19 00:00:00 +0200	Partly Cloudy	rain	26.016667	26.016667	0.43	10.9963	31.0	16.1000	0.0	1014.36	Partly cloudy starting in the morning.
96449	2016-09-19 20:00:00 +0200	Partly Cloudy	rain	24.583333	24.583333	0.48	10.0947	20.0	15.5526	0.0	1015.16	Partly cloudy starting in the morning.
96450	2016-09-20 00:00:00 +0200	Partly Cloudy	rain	22.038889	22.038889	0.56	8.9838	30.0	16.1000	0.0	1015.66	Partly cloudy starting in the morning.
96451	2016-09-20 22:00:00 +0200	Partly Cloudy	rain	21.522222	21.522222	0.60	10.5294	20.0	16.1000	0.0	1015.95	Partly cloudy starting in the morning.
96452	2016-09-23 00:00:00 +0200	Partly Cloudy	rain	20.438889	20.438889	0.61	5.8765	39.0	15.5204	0.0	1016.16	Partly cloudy starting in the morning.

ستون های مربوط به داده های عددی را نیاز داریم بنابراین تنها همین ستون ها را نگه می داریم و

مابقی را حذف می کنیم:

```
[ ] df = pd.DataFrame(df[['Temperature (C)', 'Apparent Temperature (C)', 'Humidity', 'Wind Speed (km/h)', 'Wind Bearing (degrees)', 'Visibility (km)', 'Loud Cover', 'Pressure (millibars)']].values, columns=['Temperature (C)', 'Apparent Temperature (C)', 'Humidity', 'Wind Speed (km/h)', 'Wind Bearing (degrees)', 'Visibility (km)', 'Loud Cover', 'Pressure (millibars)'])
```

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
0	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13
1	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63
2	9.377778	0.89	3.9284	204.0	14.9069	0.0	1015.94	
3	8.298889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41
4	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51
...
96448	26.016667	26.016667	0.43	10.9963	31.0	16.1000	0.0	1014.36
96449	24.583333	24.583333	0.48	10.0947	20.0	15.5526	0.0	1015.16
96450	22.038889	22.038889	0.56	8.9838	30.0	16.1000	0.0	1015.66
96451	21.522222	21.522222	0.60	10.5294	20.0	16.1000	0.0	1015.95
96452	20.438889	20.438889	0.61	5.8765	39.0	15.5204	0.0	1016.16

حال دیتای پوج را در صورت وجود از دیتاست فوق حذف (با دستور df.isnull()) و مابقی مقادیر را

برمی گردانیم:

```

# Show the number of null values in each column
null_counts = df.isnull().sum()
print("Number of null values in each column:")
print(null_counts)

# Remove rows with null values
df = df.dropna()

# Reset the index after removing rows
df = df.reset_index(drop=True)

# Now, the DataFrame 'df' contains no rows with null values
print("DataFrame after removing rows with null values:")
print(df)

```

Number of null values in each column:

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
0	0	0	0	0	0	0	0	0
1	0.375555	0	0	0	0	0	0	0
2	0.377778	0	0	0	0	0	0	0
3	0.377778	0	0	0	0	0	0	0
4	0.755556	0	0	0	0	0	0	0
...
96448	20.816667	20.816667	0.43	18.9953
96449	24.765333	24.765333	0.48	18.9957
96450	22.030888	22.030888	0.56	18.9838
96451	21.522222	21.522222	0.68	18.5294
96452	20.438089	20.438089	0.65	18.5705
96453	21.522222	21.522222	0.68	18.5294
96454	21.522222	21.522222	0.68	18.5294
96455	20.438089	20.438089	0.65	18.5705
96456	21.522222	21.522222	0.68	18.5294
96457	20.438089	20.438089	0.65	18.5705
96458	21.522222	21.522222	0.68	18.5294
96459	20.438089	20.438089	0.65	18.5705
96460	21.522222	21.522222	0.68	18.5294
96461	20.438089	20.438089	0.65	18.5705
96462	21.522222	21.522222	0.68	18.5294
96463	20.438089	20.438089	0.65	18.5705
96464	21.522222	21.522222	0.68	18.5294
96465	20.438089	20.438089	0.65	18.5705
96466	21.522222	21.522222	0.68	18.5294
96467	20.438089	20.438089	0.65	18.5705
96468	21.522222	21.522222	0.68	18.5294
96469	20.438089	20.438089	0.65	18.5705
96470	21.522222	21.522222	0.68	18.5294
96471	20.438089	20.438089	0.65	18.5705
96472	21.522222	21.522222	0.68	18.5294
96473	20.438089	20.438089	0.65	18.5705
96474	21.522222	21.522222	0.68	18.5294
96475	20.438089	20.438089	0.65	18.5705
96476	21.522222	21.522222	0.68	18.5294
96477	20.438089	20.438089	0.65	18.5705
96478	21.522222	21.522222	0.68	18.5294
96479	20.438089	20.438089	0.65	18.5705
96480	21.522222	21.522222	0.68	18.5294
96481	20.438089	20.438089	0.65	18.5705
96482	21.522222	21.522222	0.68	18.5294
96483	20.438089	20.438089	0.65	18.5705
96484	21.522222	21.522222	0.68	18.5294
96485	20.438089	20.438089	0.65	18.5705
96486	21.522222	21.522222	0.68	18.5294
96487	20.438089	20.438089	0.65	18.5705
96488	21.522222	21.522222	0.68	18.5294
96489	20.438089	20.438089	0.65	18.5705
96490	21.522222	21.522222	0.68	18.5294
96491	20.438089	20.438089	0.65	18.5705
96492	21.522222	21.522222	0.68	18.5294
96493	20.438089	20.438089	0.65	18.5705
96494	21.522222	21.522222	0.68	18.5294
96495	20.438089	20.438089	0.65	18.5705
96496	21.522222	21.522222	0.68	18.5294
96497	20.438089	20.438089	0.65	18.5705
96498	21.522222	21.522222	0.68	18.5294
96499	20.438089	20.438089	0.65	18.5705
96500	21.522222	21.522222	0.68	18.5294
96501	20.438089	20.438089	0.65	18.5705
96502	21.522222	21.522222	0.68	18.5294
96503	20.438089	20.438089	0.65	18.5705
96504	21.522222	21.522222	0.68	18.5294
96505	20.438089	20.438089	0.65	18.5705
96506	21.522222	21.522222	0.68	18.5294
96507	20.438089	20.438089	0.65	18.5705
96508	21.522222	21.522222	0.68	18.5294
96509	20.438089	20.438089	0.65	18.5705
96510	21.522222	21.522222	0.68	18.5294
96511	20.438089	20.438089	0.65	18.5705
96512	21.522222	21.522222	0.68	18.5294
96513	20.438089	20.438089	0.65	18.5705
96514	21.522222	21.522222	0.68	18.5294
96515	20.438089	20.438089	0.65	18.5705
96516	21.522222	21.522222	0.68	18.5294
96517	20.438089	20.438089	0.65	18.5705
96518	21.522222	21.522222	0.68	18.5294
96519	20.438089	20.438089	0.65	18.5705
96520	21.522222	21.522222	0.68	18.5294
96521	20.438089	20.438089	0.65	18.5705
96522	21.522222	21.522222	0.68	18.5294
96523	20.438089	20.438089	0.65	18.5705
96524	21.522222	21.522222	0.68	18.5294
96525	20.438089	20.438089	0.65	18.5705
96526	21.522222	21.522222	0.68	18.5294
96527	20.438089	20.438089	0.65	18.5705
96528	21.522222	21.522222	0.68	18.5294
96529	20.438089	20.438089	0.65	18.5705
96530	21.522222	21.522222	0.68	18.5294
96531	20.438089	20.438089	0.65	18.5705
96532	21.522222	21.522222	0.68	18.5294
96533	20.438089	20.438089	0.65	18.5705
96534	21.522222	21.522222	0.68	18.5294
96535	20.438089	20.438089	0.65	18.5705
96536	21.522222	21.522222	0.68	18.5294
96537	20.438089	20.438089	0.65	18.5705
96538	21.522222	21.522222	0.68	18.5294
96539	20.438089	20.438089	0.65	18.5705
96540	21.522222	21.522222	0.68	18.5294
96541	20.438089	20.438089	0.65	18.5705
96542	21.522222	21.522222	0.68	18.5294
96543	20.438089	20.438089	0.65	18.5705
96544	21.522222	21.522222	0.68	18.5294
96545	20.438089	20.438089	0.65	18.5705
96546	21.522222	21.522222	0.68	18.5294
96547	20.438089	20.438089	0.65	18.5705
96548	21.522222	21.522222	0.68	18.5294
96549	20.438089	20.438089	0.65	18.5705
96550	21.522222	21.522222	0.68	18.5294
96551	20.438089	20.438089	0.65	18.5705
96552	21.522222	21.522222	0.68	18.5294
96553	20.438089	20.438089	0.65	18.5705
96554	21.522222	21.522222	0.68	18.5294
96555	20.438089	20.438089	0.65	18.5705
96556	21.522222	21.522222	0.68	18.5294
96557	20.438089	20.438089	0.65	18.5705
96558	21.522222	21.522222	0.68	18.5294
96559	20.438089	20.438089	0.65	18.5705
96560	21.522222	21.522222	0.68	18.5294
96561	20.438089	20.438089	0.65	18.5705
96562	21.522222	21.522222	0.68	18.5294
96563	20.438089	20.438089	0.65	18.5705
96564	21.522222	21.522222	0.68	18.5294
96565	20.438089	20.438089	0.65	18.5705
96566	21.522222	21.522222	0.68	18.5294
96567	20.438089	20.438089	0.65	18.5705
96568	21.522222	21.522222	0.68	18.5294
96569	20.438089	20.438089	0.65	18.5705
96570	21.522222	21.522222	0.68	18.5294
96571	20.438089	20.438089	0.65	18.5705
96572	21.522222	21.522222	0.68	18.5294
96573	20.438089	20.438089	0.65	18.5705

اعداد موجود در ماتریس همبستگی در باز [1,1-] قرار دارند هرچه این اعداد به یک نزدیک تر باشند وابستگی مثبت بین دو ویژگی بیشتر و هرچه به منفی یک نزدیک تر باشند وابستگی منفی بین آن ها شدیدتر است. اعداد روی قطر همبستگی هر ویژگی با خودش را نشان می‌دهد که بطور واضح مقدار یک دارد.

به عنوان مثال برای بررسی رابطه بین دما و رطوبت با مشاهده ستون اول و سطر سوم به مقدار -0.632 می‌رسیم که به منفی یک نزدیک است در نتیجه میان این دو ویژگی وابستگی معکوس داریم و با افزایش دما، رطوبت کاهش می‌یابد. برای بررسی رابطه بین رطوبت و دمای ظاهری نیز با مرجعه به ستون اول و سطر دوم به مقدار 0.993 می‌رسیم که نشان دهنده واسطگی شدید این دو متغیر است و هرچه دمای ظاهری بیشتر باشد، دما نیز بیشتر خواهد بود. وابستگی رطوبت و دمای ظاهری نیز در سطر دوم و ستون سوم پیداست و عدد 0.603 را نشان می‌دهد که نشان از وابستگی معکوس دو متغیر دارد.

بطور کلی هیئت مپ ماتریس همبستگی می‌تواند در تحلیل روابط بین متغیرهای مختلف مفید باشد.

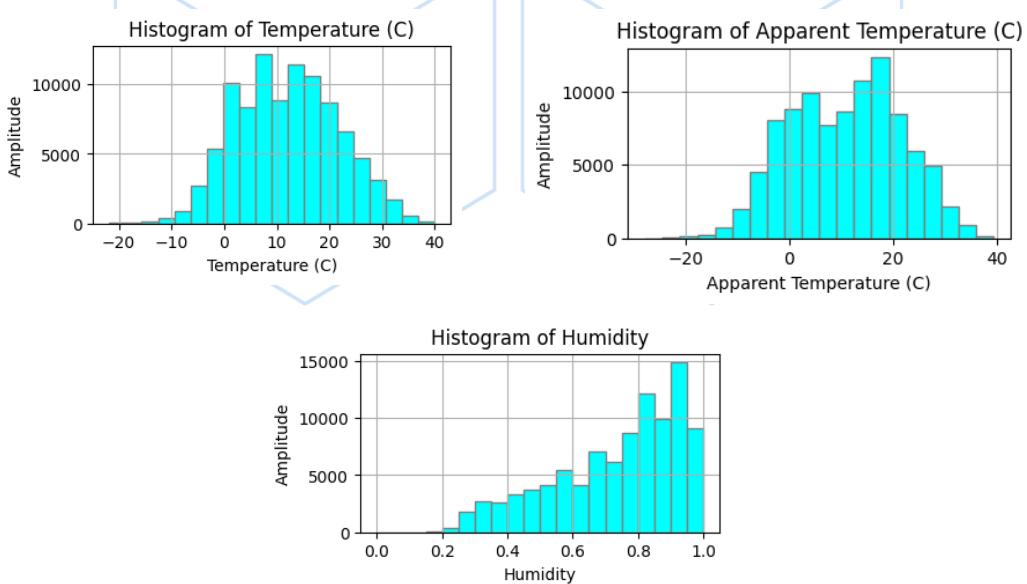
برای رسم نمودار هیستوگرام از plt.hist استفاده می‌کنیم و هیستوگرام هر ویژگی (هر کدام از ستون‌های

دیتا فریم) را رسم می‌کنیم:

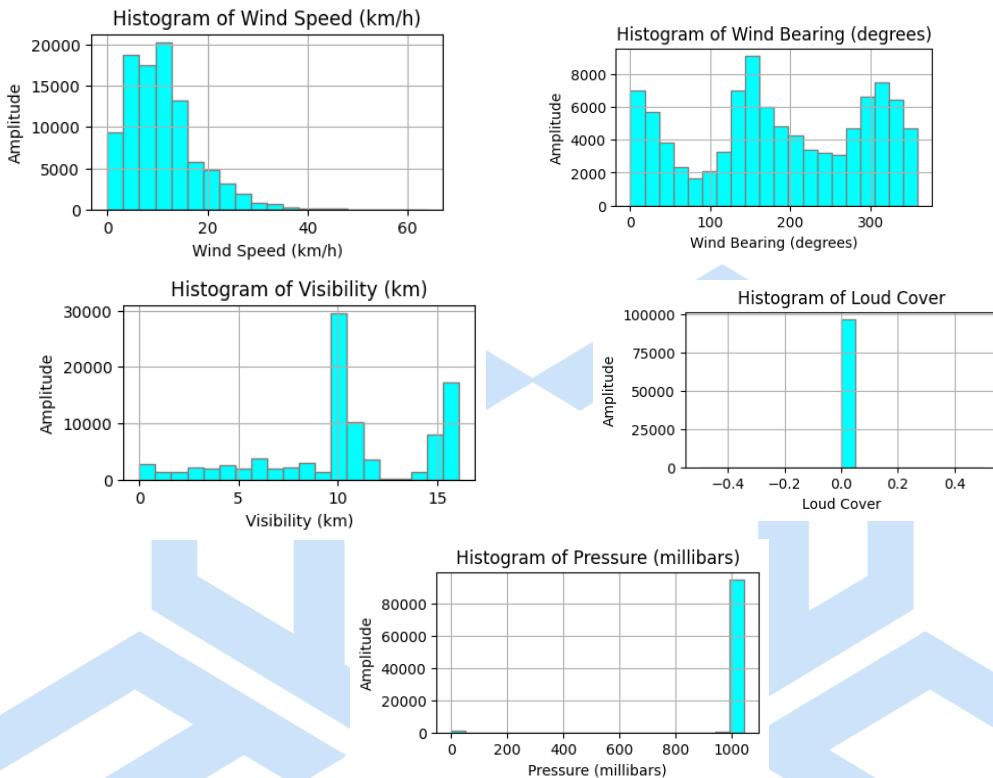
```
import matplotlib.pyplot as plt
# Loop through each column and plot a histogram
for column in df.columns:
    plt.figure(figsize=(4, 2))
    plt.hist(df[column], bins=20, color='cyan', edgecolor='grey')
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()
```

عرض هر ستون (bin) برابر ۲۰ در نظر گرفته شده است. هیستوگرام پراکندگی ویژگی‌ها بصورت زیر

ترسیم می‌شود:



هیستوگرام سایر ویژگی ها به صورت زیر است:



هیستوگرام نمایانگر توزیع تعداد داده ها در هر بازه است. برای مثال هیستوگرام دمای ظاهری تا حدی از توزیع نرمال پیروی می کند.

بطور کلی هرچه پراکندگی کمتر باشد، پیش‌بینی مقدار یک متغیر تصادفی با کمک مقدار میانگینش دقیق‌تر می‌شود؛ به عبارت دیگر، پراکندگی می‌تواند دقت یک پیش‌بینی را نشان دهد.

۳-۲

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

:LS

روش LS یا حداقل مربعات معمولی، یکی از پرکاربردترین روش های تخمین پارامتر در رگرسیون است. در این روش، هدف یافتن مجموعه ای از پارامترها است که مجموع تفاضل مربعات (اختلاف بین مقادیر واقعی و مقادیر پیش‌بینی شده توسط مدل) را به حداقل می‌رساند.

مانند ویدیو تدریسیاری 16 یک مدل رگرسیون خطی ساده را با استفاده از تعاریف متده LS بصورت آموزش می دهیم، سپس عملکرد آن را بر روی داده های ارزیابی می سنجیم و در نهایت نمودار پراکنده گی داده ها و خط رگرسیون برازش شده را رسم می کنیم.

```
import numpy as np
import matplotlib.pyplot as plt

class LinearRegressionLS:
    def __init__(self):
        self.coefficients = None

    def fit(self, X, y):
        # Add a column of ones to account for the intercept term
        X = np.column_stack((np.ones(len(X)), X))

        # Compute the coefficients using the least squares method
        self.coefficients = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

    def predict(self, X):
        # Add a column of ones to account for the intercept term
        X = np.column_stack((np.ones(len(X)), X))

        # Predict the target variable
        return X.dot(self.coefficients)
```

ارتباط میان دما و رطوبت:

ابتدا رطوبت را به عنوان X و دما را به عنوان Y در نظر می گیریم. سپس دیتا را به دو دسته آموزش و ارزیابی تقسیم می کنیم:

```
[10] import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error
     data = pd.DataFrame(df[['Humidity']])
     target = pd.DataFrame(df[['Temperature (C)']])

[11] x = data
     y = target

[12] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=64)
     x_train.shape, x_test.shape, y_train.shape, y_test.shape
((77162, 1), (19291, 1), (77162, 1), (19291, 1))
```

داده را بصورت استاندارد نرمال می کنیم:

```
[36] #Normalizing Data
     from sklearn.preprocessing import StandardScaler
     # Create a StandardScaler instance
     scaler = StandardScaler()
     # Fit the scaler on the training data and transform it
     x_train_normalized = scaler.fit_transform(x_train)
     y_train_normalized = scaler.fit_transform(y_train)
     # Transform the test data using the same scaler
     x_test_normalized = scaler.fit_transform(x_test)
     y_test_normalized = scaler.fit_transform(y_test)
```

میانگین مربعات خطای را با استفاده از `mean_squared_error` محاسبه می کنیم:

```
[37] model = LinearRegressionLS()
     model.fit(x_train_normalized, y_train_normalized)
     y_pred = model.predict(x_test_normalized)

     mse = mean_squared_error(y_test_normalized, y_pred)
     print("mean squared error : ", mse)

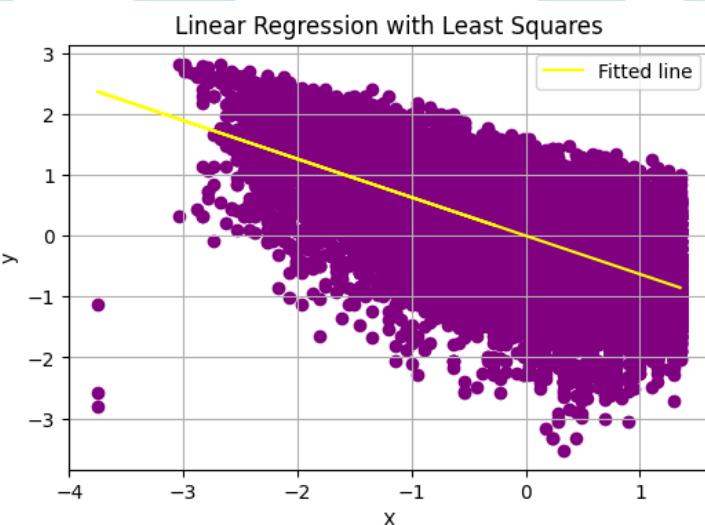
mean squared error :  0.5924361709956495
```

مقدار خطای : 0.5924361709956495

نمودار پراکندگی داده ها و خط برآذش شده به ترتیب با دستورات plt.plot و plt.scatter رسم می شود:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(6,4))
plt.scatter(x_test_normalized, y_test_normalized, color='purple')
plt.plot(x_test_normalized, y_pred, color='yellow', label='Fitted line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression with Least Squares')
plt.legend()
plt.grid(True)
plt.show()
```

نمودار پراکندگی، داده های ارزیابی را در مقابل خط رگرسیون برآذش یافته نشان می دهد. هر نقطه بنفسنرنگ یک نمونه داده از مجموعه داده های ارزیابی و خط زرد رنگ، خط رگرسیون برآذش شده با روش حداقل مربعات است.



پراکندگی نقاط بنفسنرنگ می دهد که پراکندگی داده زیاد است و کاملاً روی خط رگرسیون قرار نگرفته اند که نشان دهنده آن است که مدل رگرسیون خطی ساده نمی تواند به خوبی داده ها را برآذش کند. شبی نمودار منفی است و مهر تاییدی بر تحلیل ماتریس همبستگی دارد.(با افزایش دما، رطوبت کاهش می یابد)

ارتباط میان دمای ظاهری و رطوبت:

مراحل ذکر شده را این بار برای دیتا بی که X آن رطوبت و Y آن دمای ظاهری است، تکرار می کنیم:
فرآیند تقسیم دیتا به آموزش و ارزیابی و نرمالیزه کردن داده در قطعه کد زیر نشان داده شده است:

```

  ۱ [15] data =pd.DataFrame(df[['Humidity']])
  ۲ target2=pd.DataFrame(df[['Apparent Temperature (C)']])
  ۳ x=data
  ۴ y2=target2
  ۵
  ۶ [16] x_train, x_test, y_train2, y_test2 = train_test_split(x, y2, test_size=0.2, random_state=64)
  ۷ x_train.shape, x_test.shape, y_train2.shape, y_test2.shape
  ۸ ((77162, 1), (19291, 1), (77162, 1), (19291, 1))
  ۹
  ۱۰ #Normalizing Data
  ۱۱ from sklearn.preprocessing import MinMaxScaler
  ۱۲ # Create a StandardScaler instance
  ۱۳ scaler = MinMaxScaler()
  ۱۴
  ۱۵ # Fit the scaler on the training data and transform it
  ۱۶ x_train_normalized = scaler.fit_transform(x_train)
  ۱۷ y_train2_normalized = scaler.fit_transform(y_train2)
  ۱۸ # Transform the test data using the same scaler
  ۱۹ x_test_normalized = scaler.fit_transform(x_test)
  ۲۰ y_test2_normalized = scaler.fit_transform(y_test2)

```

مدل رگرسیون را مشابه حالت قبل آموزش می دهیم و میانگین مربعات خط را گزارش می کنیم:

```

  ۱ model=LinearRegressionLS()
  ۲ model.fit(x_train_normalized, y_train2_normalized)
  ۳ y_pred2=model.predict(x_test_normalized)
  ۴
  ۵ mse2= mean_squared_error(y_test_normalized, y_pred2)
  ۶ print("mean squared error : ", mse2)
  ۷
  ۸ mean squared error :  0.5937921800003714

```

خطا: 0.5937921800003714

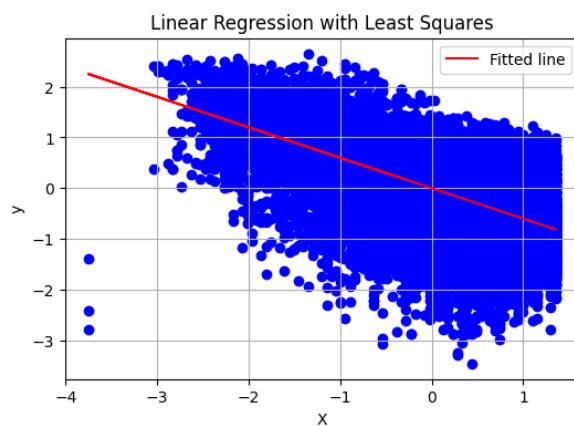
برای رسم نمودار پراکندگی داده و خط برآذش یافته داریم:

```

  ۱ import matplotlib.pyplot as plt
  ۲ plt.figure(figsize=(6,4))#
  ۳ plt.scatter(x_test_normalized, y_test2_normalized, color='blue')#
  ۴ plt.plot(x_test_normalized, y_pred2, color='red', label='Fitted line')
  ۵ plt.xlabel('X')
  ۶ plt.ylabel('Y')
  ۷ plt.title('Linear Regression with Least Squares')
  ۸ plt.legend()
  ۹ plt.grid(True)
  ۱۰ plt.show()

```

نتیجه به صورت زیر است:



به دلیل پراکندگی بالای داده LS نمی تواند به خوبی مدل را تخمین بزند. همچنین اگر به قسمت ماتریس همبستگی برگردیم. سطر دوم و ستون سوم عدد 0.603- را نشان می دهد که بیانگر وابستگی معکوس دو متغیر است. شب نمودار در این قسمت نیز منفی است و وابستگی معکوس را تایید می کند.

:RLS

روش RLS یک الگوریتم بازگشتی برای تخمین پارامترهای یک مدل رگرسیونی است و به ویژه در سیستم های زمان واقعی و پردازش سیگنال کاربرد دارد. در این روش، داده ها به صورت پویا و در لحظه دریافت می شوند و تخمین پارامترها آپدیت می شود. در اینجا برخلاف LS فاکتور فراموشی داریم که به داده ها وزن می دهد. در واقع می خواهیم اثر داده فعلی از داده قبلی در روند آموزش مدل پر رنگ تر باشد.

مطابق ویدئو تدریسیاری متدهای RLS را به صورت def نوشته و در ادامه مدل را با استفاده از این متدهای آموزش می دهیم. فاکتور فراموشی 0.995 در نظر گرفته شده است:

```
import numpy as np
import matplotlib.pyplot as plt

class RecursiveLeastSquares:
    def __init__(self, n_features, forgetting_factor=0.995):
        self.n_features = n_features
        self.forgetting_factor = forgetting_factor
        self.theta = np.zeros((n_features, 1)) # Initialize model parameters
        self.P = np.eye(n_features) # Initialize covariance matrix

    def fit(self, X, y):
        errors = []
        for i in range(len(X)):
            x_i = X[i].reshape(-1, 1)
            y_i = y[i]

            # Predict
            y_pred = np.dot(x_i.T, self.theta)

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

        return errors

    def predict(self, X):
        return np.dot(X, self.theta)
```

ارتباط میان دما و رطوبت:

در این قسمت نیز مانند آنچه در LS بیان شد، با فراخوانی X به عنوان رطوبت Y به عنوان دما، ابتدا داده را به دو دسته آموزش و ارزیابی تقسیم و این بار بصورت استاندارد نرمالیزه می کنیم:

```
[27] data = pd.DataFrame(df[['Humidity']])
target = pd.DataFrame(df[['Temperature (C)']])
x = data
y = target
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=64)

#Normalizing Data
from sklearn.preprocessing import StandardScaler
# Create a StandardScaler instance
scaler = StandardScaler()

# Fit the scaler on the training data and transform it
x_train_normalized = scaler.fit_transform(x_train)
y_train_normalized = scaler.fit_transform(y_train)
# Transform the test data using the same scaler
x_test_normalized = scaler.fit_transform(x_test)
y_test_normalized = scaler.fit_transform(y_test)
```

حال مدل را آموزش می دهیم و خطای را محاسبه می کنیم:

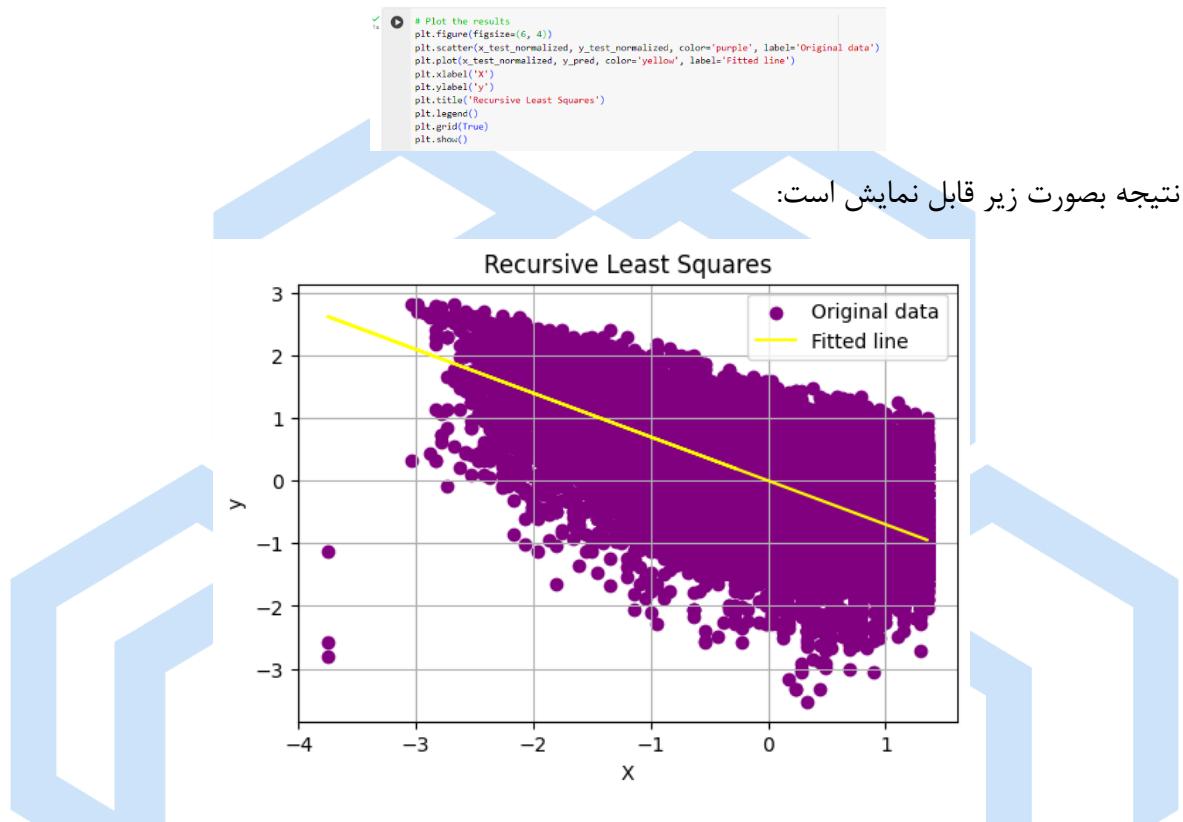
```
rls = RecursiveLeastSquares(n_features=1)
errors=rls.fit(x_train_normalized, y_train_normalized)
# Make predictions
y_pred3 = rls.predict(x_test_normalized)
# Calculate Mean Squared Error
mse3= mean_squared_error(y_test_normalized, y_pred3)
print("mean squared error : ", mse3)
```

مقدار خطای:

0.5959334890141429

در اینجا از متد RLS بصورت آفلاین استفاده شد و همانطور که می‌بینیم، خطا نسبت به روش LS تغییر چندانی نداشته است.

برای رسم نمودار پراکندگی داده و خط برآورد یافته داریم:



RLS در برآورد داده‌ها بطور مناسب عمل نکرده است و صرفاً جهت افزایش داده را تخمین می‌زند.

همچنین با استفاده از کتابخانه sm.RecursiveLS و دستور statsmodels می‌توانیم مدل RLS را ایجاد

کنیم:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from pandas_datareader import DataReader

np.set_printoptions(suppress=True)
mod = sm.RecursiveLS(x_train_normalized, y_train_normalized)
res = mod.fit()

print(res.summary())
```

Statespace Model Results						
Dep. Variable:	y	No. Observations:	77162			
Model:	RecursiveLS	Log Likelihood:	-89928.699			
Date:	Mon, 08 Apr 2024	R-squared:	0.398			
Time:	12:55:23	AIC:	179859.398			
Sample:	0	BIC:	179868.652			
Covariance Type:	nonrobust	HQIC:	179862.240			
		Scale:	0.602			
	coef	std err	z	P> z	[0.025	0.975]
x1	-0.6307	0.003	-225.755	0.000	-0.636	-0.625

Ljung-Box (L1) (Q): 0.19 Jarque-Bera (JB): 2376.12
 Prob(Q): 0.66 Prob(JB): 0.00
 Heteroskedasticity (H): 0.98 Skew: -0.39
 Prob(H) (two-sided): 0.04 Kurtosis: 3.35

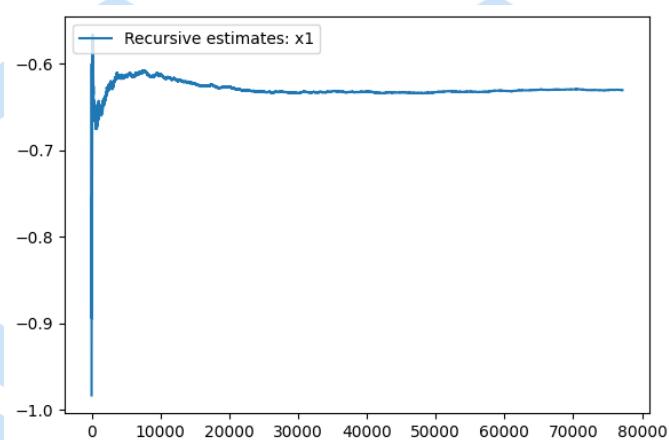
Warnings:
 [1] Parameters and covariance matrix estimates are RLS estimates conditional on the entire sample.

در این کتابخانه با دستور `res.recursive_coefficients` می‌توان ضرایب مدل برآش شده را رصد کرد:

```
print(res.recursive_coefficients)
res.plot_recursive_coefficient(alpha=None, figsize=(6, 4))

[{"filtered": array([[ 0.43668285, -0.983291 , -0.84229504, ..., -0.63070783,
   -0.63070175, -0.63069356]]), "smoothed": array([[1.4441706 , 0.15039868, 0.18867097, ..., 0.0000078 ,
  0.0000078 , 0.0000078 ]]), "smoothed_cov": array([[-0.63069356, -0.63069356, ..., -0.63069356,
   -0.63069356, -0.63069356]]), "offset": 0}]
```

پارامتر خط برآش شده در RLS بصورت زیر است:



ارتباط میان دمای ظاهری و رطوبت:

مانند قسمت قبل ابتدا داده مربوط به رطوبت را در X و دمای ظاهری را در Y قرار می‌دهیم و سپس مدل را با متod RLS آموزش می‌دهیم و خط را محاسبه می‌کنیم:

```
[52] data =pd.DataFrame(df[['Humidity']])
target2=pd.DataFrame(df[['Apparent Temperature (C)']])
x=data
y2=target2
x_train, x_test, y_train2, y_test2 = train_test_split(x, y2, test_size=0.2, random_state=64)
#Normalizing Data
from sklearn.preprocessing import StandardScaler
# Create a StandardScaler instance
scaler = StandardScaler()

# Fit the scaler on the training data and transform it
x_train_normalized = scaler.fit_transform(x_train)
y_train2_normalized = scaler.fit_transform(y_train2)
# Transform the test data using the same scaler
x_test_normalized = scaler.fit_transform(x_test)
y_test2_normalized = scaler.fit_transform(y_test2)

rls2 = RecursiveLeastSquares(n_features=1)
errors=rls2.fit(x_train_normalized, y_train2_normalized)

# Make predictions
y_pred4 = rls2.predict(x_test_normalized)
# Calculate Mean Squared Error
mse4= mean_squared_error(y_test2_normalized, y_pred4)
print("mean squared error : ", mse4)

mean squared error :  0.5932436947514965
```

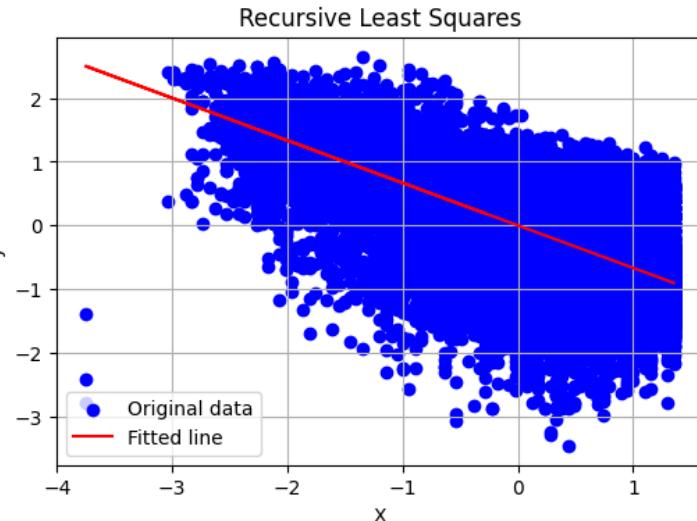
مقدار خط: 0.5932436947514965

در این حالت نیز خطای تغییری نسبت به مدل LS نداشته است. برای رسم نمودار پراکندگی داده ها و خط

برازش شده داریم:

```
# Plot the results
plt.figure(figsize=(6, 4))
plt.scatter(x_test_normalized, y_test2_normalized, color='blue', label='Original data')
plt.plot(x_test_normalized, y_pred2, color='red', label='Fitted line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Recursive Least Squares')
plt.legend()
plt.grid(True)
plt.show()
```

نتیجه به صورت زیر است:



با استفاده از کتابخانه statsmodels و دستور sm.RecursiveLS می توانیم مدل RLS را ایجاد کنیم:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from pandas_datareader import DataReader

np.set_printoptions(suppress=True)
mod = sm.RecursiveLS(x_train_normalized, y_train2_normalized)
res1 = mod.fit()
print(res1.summary())
```

Dep. Variable:	y	No. Observations:	77162			
Model:	RecursiveLS	Log Likelihood:	-92216.272			
Date:	Mon, 08 Apr 2024	R-squared:	0.361			
Time:	13:32:24	AIC:	184434.545			
Sample:	'1960-01-01': '1990-12-31'	BIC:	184437.389			
Covariance Type:	nonrobust	HQIC:	184437.386			
		Scale:	0.639			
	coef	std err	z	P> z	[0.025	0.975]
x1	-0.6008	0.003	-208.780	0.000	-0.006	-0.595
Ljung-Box (L1) (Q):	0.11	Jarque-Bera (JB):	2041.21			
Prob(Q):	0.74	Prob(JB):	0.00			
Heteroskedasticity (H):	0.98	Skew:	-0.39			
Prob(H) (two-sided):	0.07	Kurtosis:	3.15			

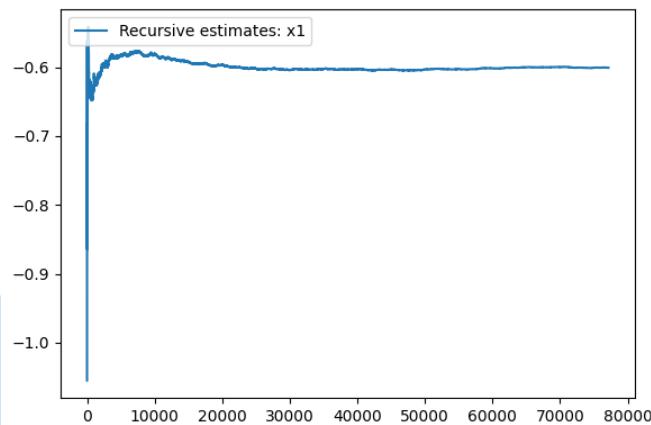
Warnings:
[1] Parameters and covariance matrix estimates are RLS estimates conditional on the entire sample.

در این کتابخانه با دستور res.recursive_coefficients می توان ضرایب مدل برآورد شده را رصد کرد:

```
print(res1.recursive_coefficients)
res1.plot_recursive_coefficient(alpha=None, figsize=(6, 4))

{'filtered': array([[ 0.41641463, -1.0547919 , -0.94435695, ... , -0.60083772,
       -0.60083082, -0.60082147]]), 'filtered_cov': array([[ [1.39344246, 0.19443595, 0.14866539, ... , 0.00000828,
       0.00000828, 0.00000828]]]), 'smoothed': array([-0.60082147, -0.60082147, -0.60082147, ... , -0.60082147,
       -0.60082147, -0.60082147]], 'smoothed_cov': array([[ [0.00000828, 0.00000828, 0.00000828, ... , 0.00000828,
       0.00000828, 0.00000828]]]), 'offset': 0}
```

نتیجه به صورت زیر است:



۳-۳

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

در LS همه مشاهدات وزن یکسانی دارند اما در روش WLS مشاهدات می‌توانند وزن‌های متفاوتی داشته باشند. اساس WLS این است که برخی مشاهدات ممکن است اطلاعات بیشتر یا کیفیت بالاتری نسبت به

مشاهدات دیگر داشته باشند. بنابراین، با اختصاص وزن‌های مناسب، می‌توان تاثیر مشاهدات با کیفیت

بالاتر را در تخمین پارامترها افزایش داد. وزن‌ها می‌توانند بر اساس معیارهای مختلفی انتخاب شوند، مانند کیفیت داده‌ها، میزان اهمیت یا اعتبار هر مشاهده، یا براساس دانش قبلی در مورد داده‌ها.

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

که Q در رابطه بالا ماتریس وزن است که بصورت قطری و به صورت $\frac{1}{var(y)}$ تعریف می‌شود.

برای پیاده سازی الگوریتم از کتابخانه statsmodels.api استفاده می‌کنیم:

ارتباط میان دما و رطوبت:

مجدداً با تعریف داده مربوطه و تقسیم به داده آموزش و ارزیابی و نرمالیزه کردن آنها با استفاده از نرم‌الاسازی استاندارد داریم:

```

import numpy as np
import statsmodels.api as sm

data = pd.DataFrame(np.array([["Humidity"], [Temperature (C)]]))
target = pd.DataFrame(np.array([[y]]))
xdata = data
ytarget = target

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)

#Normalizing Data
from sklearn.preprocessing import StandardScaler
# Create a StandardScaler instance
scaler = StandardScaler()

# Fit the scaler on the training data and transform it
x_train_normalized = scaler.fit_transform(x_train)
y_train_normalized = scaler.fit_transform(y_train)
# Transform the test data using the same scaler
x_test_normalized = scaler.fit_transform(x_test)
y_test_normalized = scaler.fit_transform(y_test)

error_variance = np.var(y_train_normalized)

# Calculate weights based on the estimated variance
weights = 1 / error_variance

# Fit the weighted least squares model
x_train_normalized = sm.add_constant(x_train_normalized) # Add intercept term
model = sm.OLS(y_train_normalized, x_train_normalized, weights=weights)
result = model.fit()
res_ols = result.Sim(y_test_normalized, x_test_normalized).fit()
print(result.summary())

```

ML Regression Results

Dep. Variable:	y	R-squared:	0.398			
Model:	OLS	AIC:	0.398			
Method:	least squares	F-statistic:	5.59e+04			
Date:	Fri, 08 Apr 2020	P-value (F-statistic):	0.398			
Time:	14:05:46	Log-likelihood:	-893.0			
No. Observations:	776.02	AIC:	1.79e+05			
Df Residuals:	776.02	BIC:	1.79e+05			
Df Model:	1					
Covariance Type:	oprobust					
	user	std err	P> t	[0.025	0.975]	
const	3.37e+14	8.003	1.12e+14	0.000	-0.005	0.005
x1	-0.6107	9.803	-25.754	0.000	-0.636	-0.625
Omnibus:	1021.122	Durbin-Watson:	1.990			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2188.141			
Skew:	-0.630	Prob(JB):	0.000			
Kurtosis:	3.178	Cond. No.:	1.00			

Notes:

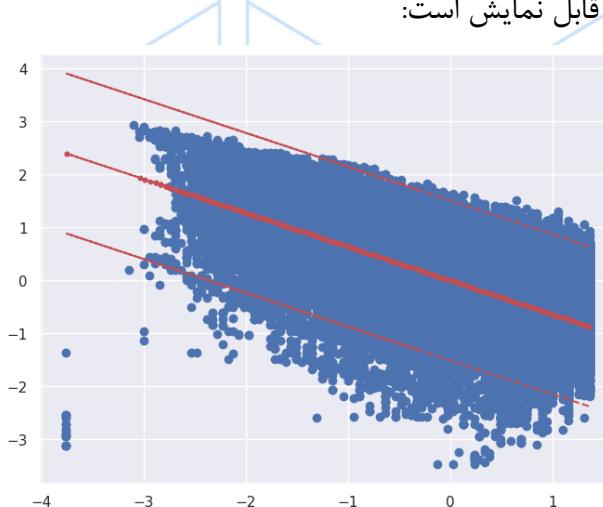
- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

با استفاده از پیش‌بینی مدل که با WLS آموزش دیده شده نمودار پراکندگی داده را به همراه خط برازش شده و همچنین کران بالا و یاپین یا این خط رسم می‌کنیم:

```

pred_wls = res_wls.get_prediction()
iv_l = pred_wls.summary_frame()["obs_ci_lower"]
iv_u = pred_wls.summary_frame()["obs_ci_upper"]
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(x_train_normalized, y_train_normalized, "o", label="Data")
ax.plot(x_test_normalized, res_wls.fittedvalues, "r-.")
ax.plot(x_test_normalized, iv_u, "r--", label="WLS")
ax.plot(x_test_normalized, iv_l, "r--")

```



نتیجه به صورت زیر قابل نمایش است:

پیداست WLS مدل بهتری نسبت به LS و WLS ارائه داده است.

ارتباط میان دمای ظاهری و رطوبت:

با تعریف داده مربوطه و تقسیم به داده آموزش و ارزیابی و نرم‌الیزه کردن آنها داریم:

```
data = pd.DataFrame(pd.read_csv('humidity.csv'))
target2 = pd.DataFrame(pd.read_csv('Apparent_Temperature (C).csv'))
xdata = target2['Humidity']
ytarget2 = target2['Apparent_Temperature (C)']

x_train2, x_test2, y_train2, y_test2 = train_test_split(xdata, ytarget2, test_size=0.2, random_state=42)

# Normalizing data
from sklearn.preprocessing import StandardScaler
# Create a StandardScaler instance
scaler = StandardScaler()

# Fill the scaler on the training data and transform it
x_train2_normalized = scaler.fit_transform(x_train2)
y_train2_normalized = y_train2 # Add intercept term
x_train2_normalized = np.insert(x_train2_normalized, 0, 1, axis=1)
x_test2_normalized = np.insert(scaler.transform(x_test2), 0, 1, axis=1)
y_test2_normalized = y_test2 # Modify this value based on your estimation
error_variance = 1 # Modify this value based on your estimation

# Calculate weights based on the estimated variance
weights = 1 / error_variance

# Fit the weighted least squares model
x_train2_normalized = sm.add_constant(x_train2_normalized) # Add intercept term
model = sm.WLS(y_train2_normalized, x_train2_normalized, weights=weights)
result = model.fit()
res_wls1 = sm.WLS(y_test2_normalized, x_test2_normalized).fit()
print(result.summary())
print(res_wls1.summary())
```

M5 Regression Results

Deg. Variables	Y	R-squared	0.301		
Model:	OLS	F-statistic:	8.301		
Method:	Least Squares	t-statistic:	4.359e+04		
Date:	Fri, 08 Apr 2023	P-value(F-statistic):	0.00		
Time:	15:00:36	Log-Likelihood:	-912.08		
No. Observations:	77202	AIC:	1.844e+05		
De Residuals:	77202	BIC:	1.844e+05		
Df Model:	1				
Covariance Type:	Heteroskedasticity				
coef std err t P> t [0.05 0.95]					
const	1.05e-14	0.000	1.00e-14	0.000	0.000
x1	0.000	0.000	0.775	0.000	0.7295
Omnibus: 2828.761 Durbin-Watson: 1.990					
Prob(Omnibus): 0.000 Jarque-Bera (JB): 314.254					
Skew: -0.496 Prob(JB): 0.00					
Kurtosis: 3.100 Cond. No. 1.80					

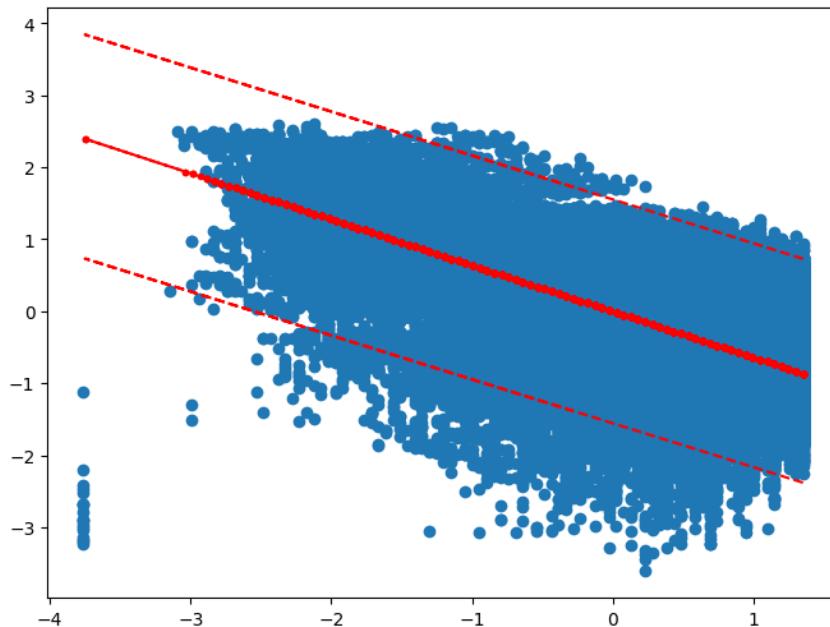
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

نمودار پراکندگی داده را به همراه خط برازش شده و همچنین کران بالا و پایین برای این خط رسم می‌کنیم:

```
pred_wls1 = res_wls1.get_prediction()
iv_l = pred_wls1.summary_frame()['obs_ci_lower']
iv_u = pred_wls1.summary_frame()['obs_ci_upper']
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(x_train2_normalized, y_train2_normalized, 'o', label="Data")
ax.plot(x_train2_normalized, res_wls1.fittedvalues, 'r--', label="WLS")
ax.plot(x_test2_normalized, iv_u, 'r--', label="M5")
ax.plot(x_test2_normalized, iv_l, 'r--')
ax.plot(x_test2_normalized, iv_u, 'r--')
```

نتیجه به صورت زیر قابل نمایش است:



ارتباط میان دمای ظاهری و رطوبت نیز با استفاده از WLS و RLS تخمین زده است.

مراجع

- [1] <https://medium.com/@500087551/all-you-need-to-know-about-cwru-dataset-8d391577d8f2>.
- [2] <https://blog.faradars.org/standardization-and-normalization-in-python/>
- [3] <https://github.com/MJAHMADEE/MachineLearning2024W>

