دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکدهٔ مهندسی برق - گروه مهندسی کنترل

درس یادگیری ماشین

پاسخ مینی‌پروژه اول

نام و نام خانوادگی	ایمان گندمی
شمارهٔ دانشجویی	۴۰۲۲۳۷۰۴
تاریخ	فروردین ۱۴۰۳

فهرست مطالب

فهرست تصاویر

فهرست جداول

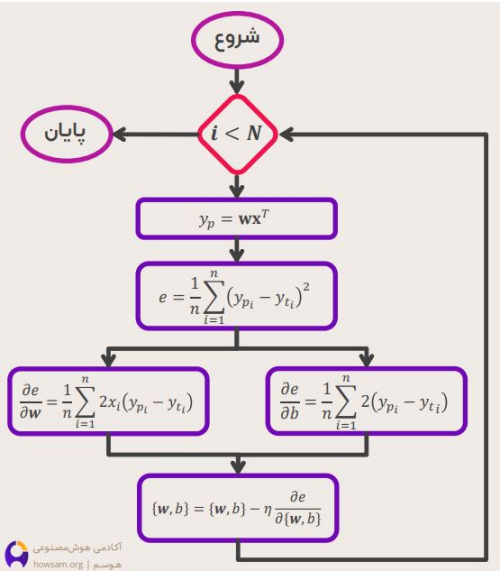
فهرست برنامه‌ها

لینک پوشهٔ گیت‌هاب:
https://github.com/ImanGandomi/MachineLearning2024/tree/main/ml_MiniProject_1
لینک گوگل کولب:
https://colab.research.google.com/drive/1NwZ3BcL3-qJCNFXE9iJAdm-rj0CTM8ec?usp=sharing

۱ سوال اول

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

ساده‌ترین بلوک دیاگرام یک طبقه‌بند خطی را می‌توان بصورت زیر در نظر گرفت. مطابق این سیستم در محلهٔ اول با تعریف مجموعه‌دادهٔ مورد نظر و تعداد دورهٔ آموزشی، مقداردهی اولیه پارامترها صورت می‌پذیرد. این پارامترها شامل مقادیر وزن‌ها و بایاس‌ها است. در قدم بعدی مقدار وزن و ورودی داده‌شده از مجموعه داده مورد ارزیابی و محاسبه قرار گرفته و مقدار خروجی y-h محاسبه می‌شود. در قرم بعدی نوبت به محاسبهٔ اتلاف می‌رسد. اتلاف در واقع اختلاف اختلاف مقدار پیش‌بینی‌شده و مقدار واقعی برچسب دادهٔ مورد نظر است. این اتلاف با توجه به تابع اتلاف مورد نظر ما محاسبه می‌شود. سپس باید گرادیان پارامترها محاسبه شود. در واقع گرادیان مقدار مشتق آن پارامترها است. همانطور که در ریاضیات برای رسیدن به مقدار بهینهٔ یه پارامتر از آن مشتق میگرفتیم، اینجا هم همین اتفاق می‌افتد. در ادامه این مقدار محاسبه‌شده برای گرادیان این پارامترها باید با در نظر گرفتن یک مقدار کنترل‌شده‌ای به اسم نرخ یادگیری، از مقدار قبلی آن پارامتر کاسته شود تا به سمت مقادیر بهینه حرکت کنیم. این کار تا پایان دوره‌های آموزشی تکرار می‌شود. مرحلهٔ ارزیابی پس از متوقف شدن و ذخیره شدن بهترین مدل انجام می‌پذیرد. به این صورت که وزن‌های بهترین مدل که ذخیره شده‌اند، مورد استفاده قرار گرفته و روی داده‌های ارزیابی تست می‌شود.



شکل ۱: بلوک دیاگرام مدل طبقه‌بند خطی

اگر بخواهیم در یکی مدل طبقه‌بند خطی نوع طبقه‌بندی را از حالت دوکلاسه به حالت چندکلاسه تغییر دهیم، در قسمت تعریف توع تابع اتلاف باید تغییراتی ایجاد کنیم. در تصویر مربوطه از تابع اتلاف BCE استفاده شده‌است که مناسب حالت‌های دوکلاسه است. این قسمت باید با توجه به چندکلاسه بودن تغییر کند و تابعی مورد استفاده قرار گیرد که مناسب باشد. همچنین هر قسمتی که دارای تعیرف دوکلاسه باشد باید تغییر کرده و حالت چندکلاسه به خود بگیرد.

۲.۱ با استفاده از datasets.sklearn 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 می‌توانید دیتاست تولیدشده خود را چالش برانگیزتر و سخت‌تر کنید؟

ابتدا کتابخانه‌های مورد نیاز را import می‌کنیم.

```
from sklearn.datasets import load_breast_cancer, make_classification,
    make_blobs, make_circles, load_digits
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from mpl_toolkits.mplot3d import Axes3D
from mlxtend.plotting import plot_decision_regions
```

Code 1: import libraries (Python)

سپس با استفاده از sklearn.datasets و مطابق ویژگی‌های مطرح شده توسط صورت سوال دیتاستی مناسب طبقه‌بندی و با ۱۰۰۰ نمونه و ۴ کلاس و ۳ ویژگی تولید می‌کنیم. برای تعداد نمونه ها با samples-n و برای تعداد کلاس ها با classes-n و برای تعداد ویژگی ها با features-n کار می کنیم و برابر با مقادیر خواسته شده در صورت سوال، قرارشان می دهیم. در این قسمت random-state را مطابق دو رقم آخر شماره دانشجویی و برابر ۰۴ می‌گیریم. class-sep میزان تفکیک داده ها را از هم مشخص می کند. اگر مقدارش کم باشد، داده ها به هم نزدیک‌تر هستند و اگر بیشتر باشد، داده ها از هم دورتر می‌شوند. n-cluster-per-class تعداد خوشه های مربوط به هر کلاس را مشخص می کند. وقتی آن را برابر یک قرار می دهیم، یعنی هر کلاس یک خوشه جدا برای خود دارد.

```
# Generate dataset
X, y = make_classification(n_samples=1000,
                           n_features=3,
```

```
                                n_classes=4,
                                n_redundant=0,
                                n_clusters_per_class=1,
                                class_sep=1 ,
                                random_state=4)
```

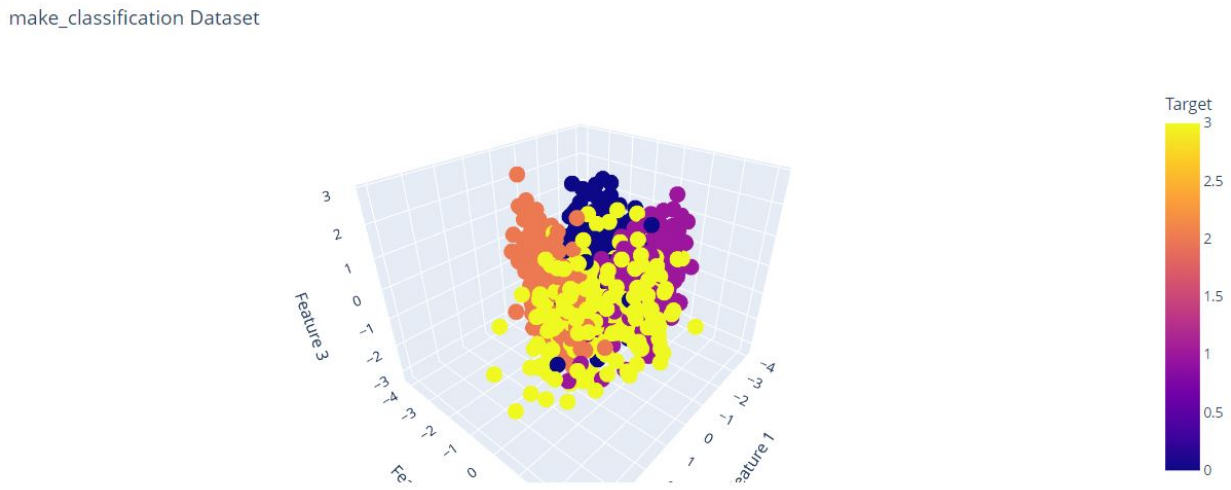
Code 2: Generate Dataset

برای رسم مجموعه‌داده بصورت ۳D از کتابخانهٔ plotly.express استفاده می‌کنیم. بدین منظور ابتدا داده‌های تولید شده را بصورت دیتافریم درمی‌آوریم و سه ویژگی و لیبل مخصوص هر داده را به عنوان ورودی به تابع مورد نظر می‌دهیم.

```
# Create a DataFrame from the generated dataset
df = pd.DataFrame(X, columns=['Feature 1', 'Feature 2', 'Feature 3'])
df['Target'] = y

# Plot the dataset using Plotly Express
fig = px.scatter_3d(df, x='Feature 1', y='Feature 2', z='Feature 3', color='
    Target', title='make_classification Dataset')
fig.show()
```

Code 3: Plot Dataset in 3D

خروجی مورد نظر بصورت زیر است. این نمودار بصورت سه بعدی بوده و امکان داشتن موقعیت مکانی هر داده و بررسی دقیق‌تر با بزرگنمایی را دارد.



شکل ۲: 3D plot of dataset

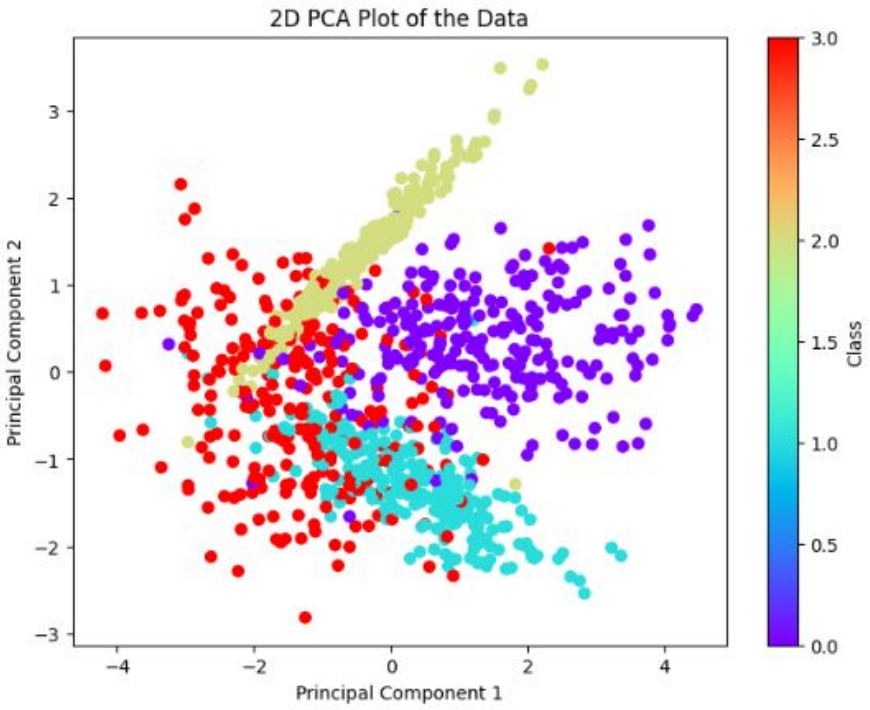
همچنین امکان رسم مجموعه‌داده بصورت دوبعدی و درک بهتر نیز فراهم است. برای این کار از کتابخانهٔ sklearn.decomposition استفاده شده است. PCA می‌تواند کاهش بعد برای مجموعه‌داده را فراهمی کند بصورتی که با ادغام ویژگی‌های موجود در مجموعه‌داده و ترکیب آن‌ها با هم، از سه ویژگی حاضر در دیتاست آن را به دو ویژگی کاهش میدهد که قابل رسم در در فضای دوبعدی باشد.

```
# Perform PCA for dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot the data
plt.figure(figsize=(8, 6))

# Scatter plot
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap=plt.cm.rainbow)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('2D PCA Plot of the Data')

plt.colorbar(label='Class')

plt.show()
```

Code 4: Plot Dataset in 2D

خروجی این کد بصورت شکل ۳ زیر است.

چالش برانگیز بودن مجموعه‌دادهٔ تولید شده با توجه به تعداد کلاس‌ها و ویژگی‌های مشخص شده را می‌توان با مقادیر n-clusters-per-class که میزان چند قسمتی بودن هر کلاس را مشخص می‌کند و class-sep که میزان دوری یا درهم‌آمیختگی دیتاست را مشخص می‌کند، تنظیم کرد. در این مسئله این مقادیر برابر با ۱ قرار داده شده است. بنابراین از نظر چالش‌برانگیز بودن در حالت متوسط قرار دارد. اما می‌توان برای چالش‌برانگیزتر کردن class-sep را کاهش و یا n-clusters-per-class را افزایش داد. یک نمونه مثال در شکل ۴ آورده شده است. در این قسمت با افزایش تعداد n-clusters-per-class به مقدار ۲ و همچنین افزایش class-sep مقدار چالش‌برانگیز شدن دیتاست تغییر پیدا کرده است. اگر بخواهیم این چالش زیاد شود باشد مقدار این پارامترها را به ترتیب بزرگتر و کوچکتر کرد.



شکل ۳: 2D plot of dataset

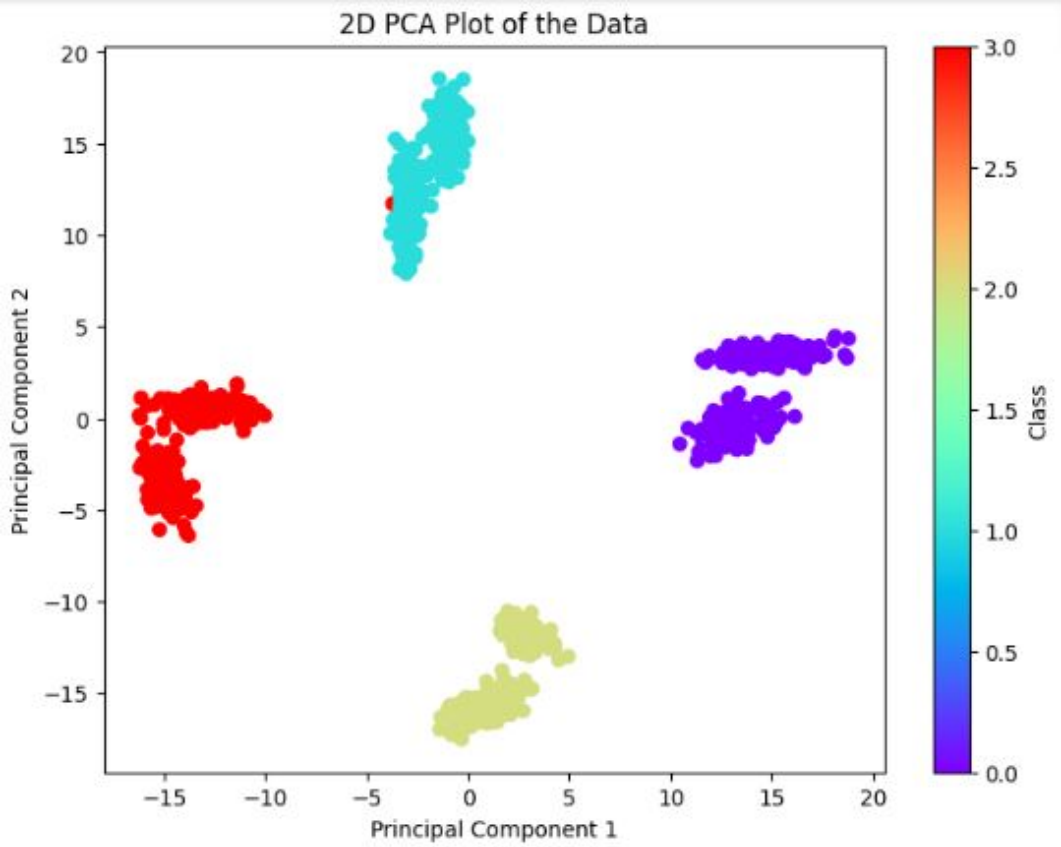
۳.۱ با استفاده از حداقل دو طبقه بند خطی آمادهٔ پایتون (در model-linear.sklearn)و در نظر گرفتن فراپارامترهای مناسب، چهار کلاس موجود در دیتاست قسمت قبلی را از هم تفکیک کنید. ضمن توضیح روند انتخاب فراپارامترها (مانند تعداد دورهٔ آموزش و نرخ یادگیری)، نتیجهٔ دقت آموزش و ارزیابی را نمایش دهید. برای بهبود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

در این مرحله ابتدا نیاز است که مجموعه‌دادهٔ خود را به دو قسمت آموزش و ارزیابی تقسیم کنیم. این مرحله بصورت آمده در زیر انجام می‌شود. مقدار ۲۰ درصد از داده‌ها را به قسمت ارزیابی اختصاص می‌دهیم. باید توجه داشت که این درصد بستگی به میزان داده‌های کل می‌تواند متفاوت باشد. هر چه تعداد کل داده‌ها بیشتر باشد، می‌شود درصد کمتری را به مجموعه‌داده‌های ارزیابی اختصاص داد. به عنوان مثال در یک مجموعه‌دادهٔ یک میلیونی میزان یک درصد از داده‌ها که برابر ۱۰ هزار داده است هم شاید برای مجموعه‌دادهٔ ارزیابی زیاد باشد و از این هم کمتر می‌شود اختصاص داد.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=4)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Code 5: split dataset

در مرحلهٔ بعد لازم است که مقداری در مورد مجموعه‌دادهٔ خود اطلاعاتی بدست آوریم. بدین منظور با استفاده از کد زیر سعی شده است بازهٔ تغییرات ویژگی‌های آموزش مورد بررسی قرار گیرد.



شکل ۴: n-clusters-per-class=2, class-sep=10

```python
# Compute the range for each feature
feature_ranges = [(np.min(X_train[:, i]), np.max(X_train[:, i])) for i in
    range(X_train.shape[1])]

# Print the range for each feature
for i, (min_val, max_val) in enumerate(feature_ranges):
    print(f"Feature {i+1}: Range = ({min_val}, {max_val})")
```

Code 6: feature range

خروجی نشان‌دهندهٔ این است که بازه تغییرات هر سه ویژگی در یک محدودهٔ مشخص است و تغییرات قابل توجه توجهی نسبت به یکدیگر ندارند.

به عنوان اولین طبقه‌بند آمادهٔ پایتون از LogisticRegression استفاده کرده می‌کنیم. این طبقه‌بند پارامترهای مختلفی را به عنوان ورودی می‌پذیرد. به عنوان مثال پارامتر solver نوع بهینه‌ساز را مشخص می‌کند و پارامتر C مقدار نرخ بهینه سازی را مشخص می‌کند. در این قسمت سعی می‌شود که به طور سازماندهی شده‌ای مقادیر مختلف برای این پارامترها انتخاب شود که بهینه‌ترین نتایج حاصل شود. بدین منظور از حلقه‌های تکرار برای بررسی مقادیر بهینهٔ پارامترها استفاده می‌شود. پارامترهای در نظر گرفته شده برای تغییر در طول

```
Feature 1: Range = (-3.802236114367438, 3.716621174692759)
Feature 2: Range = (-4.126933452031901, 3.1082741888927683)
Feature 3: Range = (-3.2146554969049794, 2.8715878457267467)
```

شکل ۵: بازهٔ تغییرات ویژگی‌های دیتاست

آموزش‌های پی‌درپی ،C solver و tol هستند. تعداد دورهٔ آموزش را هم با توجه به بررسی‌های صورت گرفته برابر ۱۰۰۰ قرار دادیم چون تعداد آموزش‌های بیشتر از آن به نتایج بهتر منجر نمی‌شد.

```
C_values = np.linspace(0.01, 1, 50)
solver_list = ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', '
    saga']
best_result = 0
for C in C_values:
  for solver in solver_list:
    for i in np.linspace(1e-5, 1e-3, 50):
      LR_model = LogisticRegression(penalty='l2',
                                    dual=False,
                                    tol = i,
                                    solver = solver,
                                    max_iter=1000,
                                    C = C,
                                    random_state=4)
      LR_model.fit(X_train, y_train)
      result = LR_model.score(X_train, y_train)
      if result > best_result:
        best_result = result
        print (f"best result is {best_result} with solver: {solver} , C= {C}
    and tol= {i}")
```

Code 7: train model

خروجی مورد نظر با مشخص بودن بهینه‌ترین پارامترها بصورت زیر است.

میزان نزدیکی داده‌ها به یکدیگر در مرحلهٔ تولید مجموعه‌داده باعث چالش‌برانگیز شدن مجموعه‌داده شده است. حال اگر مقدار class-sep را برابر ۲ بگذاریم که این به معنی فاصله بیشتر داده‌ها از یکدیگر است و سپس اموزش را انجام دهیم به نتایج زیر می‌رسیم.

این نتیجه نشان می‌دهد که اهمیت این پارامترها در تولید دیتاست و نتایج حاصل شده بالاست.

نتیجه آموزش و ارزیابی به صورت زیر قابل محاسبه است.

```
best result is 0.775 with solver: lbfgs , C= 0.01  and tol= 1e-05
best result is 0.78375 with solver: lbfgs , C= 0.030204081632653063  and tol= 1e-05
best result is 0.785 with solver: sag , C= 0.39387755102040817  and tol= 0.0005555102040816327
best result is 0.78625 with solver: sag , C= 0.5959183673469388  and tol= 0.00047469387755102045
best result is 0.7875 with solver: lbfgs , C= 0.9393877551020408  and tol= 1e-05
```

شکل ۶: نتیجه آموزش با پارامترهای مختلف

```
best result is 0.96875 with solver: lbfgs , C= 0.01  and tol= 1e-05
best result is 0.97125 with solver: lbfgs , C= 0.030204081632653063  and tol= 1e-05
best result is 0.9725 with solver: sag , C= 0.333265306122449  and tol= 0.0005555102040816327
```

شکل ۷: نتیجه آموزش با پارامترهای مختلف با class-sep=۲

```python
LR_model.fit(X_train, y_train)
LR_model.predict(X_test), y_test

print("LR_model train score = ", LR_model.score(X_train, y_train))
print("LR_model predict score = ", LR_model.score(X_test, y_test))
```

Code 8: train and test score

نتایج را به صورت زیر می‌توان دید.

```
LR_model train score =  0.9725
LR_model predict score =  0.98
```

شکل ۸: نتایج اموزش و ارزیابی

به عنوان دومین طبقه‌بند امادهٔ پایتونی از SGDClassifier استفاده می‌کنیم. این طبقه‌بند نیز دارای پارامترهای ورودی مختلفی است. برای تست کردن اثر مقادیر مختلف پارامترهای و بررسی بهینه‌ترین آن‌ها مشابه قبل از حلقه‌های تکرار استفاده کرده‌ایم. ارامترهای در نظر گرفته شده برای تغییر در طول آموزش‌های پی‌درپی eta۰ که به معنی مقدار اولیه نرخ آموزشی است و نوع learning-rate که شامل دو نوع خودکار optimal و adaptive است و نوع تابع loss است.

کد زیر به عنوان مرحلهٔ آموزش مورد استفاده قرار گرفته است.

```python
loss_list = ['hinge', 'log_loss', 'squared_hinge', 'squared_error', 'huber']
learning_rate = ['optimal', 'adaptive']
best_result = 0
for loss in loss_list:
  for lr in learning_rate:
```

```
    for eta in np.linspace(1e-3, 30, 50):
      SGD_model = SGDClassifier(loss=loss,
                                max_iter =2000,
                                tol = 1e-3,
                                eta0 = eta,
                                learning_rate = lr,
                                random_state=4)
      SGD_model.fit(X_train, y_train)
      result = SGD_model.score(X_train, y_train)
      if result > best_result:
        best_result = result
        print (f"best result is {best_result} with loss: {loss} , learning
   rate= {lr} and eta0= {eta}")
```

Code 9: SGDClassifier

نتایج اموزش و با خروجی گرفتن بهینه‌ترین مقادیر پارامترهای موجود به صورت زیر است.

```
best result is 0.95875 with loss: hinge , learning rate= optimal and eta0= 0.001
best result is 0.9625 with loss: hinge , learning rate= adaptive and eta0= 0.001
best result is 0.96625 with loss: hinge , learning rate= adaptive and eta0= 0.6132244897959184
best result is 0.9675 with loss: hinge , learning rate= adaptive and eta0= 6.735469387755102
```

شکل ۹: نتایج آموزش با پارامترهای مختلف

کد مورد استفاده برای نتایج آموزش و ارزیابی و خروجی مورد نظر بصورت زیر آمده است.

```
SGD_model.fit(X_train, y_train)
SGD_model.predict(X_test), y_test

print("SGD_model train score = ", SGD_model.score(X_train, y_train))
print("SGD_model predict score = ", SGD_model.score(X_test, y_test))
```

Code 10: SGD train and test score

```
SGD_model train score =  0.9675
SGD_model predict score =  0.975
```

شکل ۱۰: نتایج آموزش و تست SGD

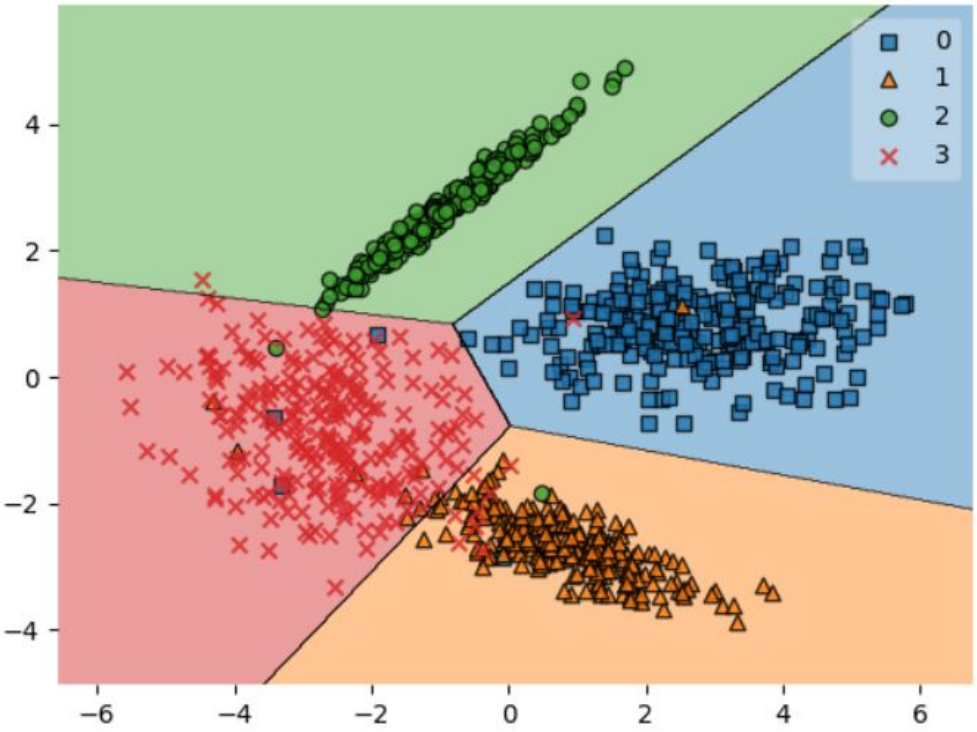
برای بهبود نتیجه در مرحلهٔ اول سعی شد که با استفاده از تغییرات پارامترهای مورد استفاده در هر طبقه‌بند، بهینه‌ترین نتایج حاصل شود. بدین منظور ابتدا تعداد دوره‌های آموزشی افزایش داده شد که به خوبی داده‌های مختلف در شبکه آموزش ببینندو سپس با استفاده از حلقه‌های تکرار سعی شد بهینه‌ترین مقدار برای سایرپارامترها انتخاب شود. مورد دیگری که مورد بررسی قرار گرفت میزان تغییرات ویژگی‌های دیتاست بود. همانطور که پیش از این بررسی شد، تغییرات در مورد این دیتاست به خصوص، دارای تفاوت زیادی در بین ویژگی‌های خود نبود و این یک ویژگی مناسب برای یک دیتاست است. اگر دیتاستی دارای ویژگی‌های با بازه تغییرات بسیار متفاوت باشد، نیاز است ابتدا نرمال‌سازی داده صورت بگیرد. البته در این مرحله نرمال‌سازی داده صورت گرفت و در کد موجود است اما به دلیل اینکه بازه تغییرات خود مجموعه داده مناسب بود و با اعمال نرمال سازی با تابع minmaxscaler نتایج بهتری حاصل نشد، از آوردن در گزارش‌کار خودداری شد.

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

در این قسمت برای رسم داده‌ها به همراه مرز تصمیم‌گیری هر طبقه از کد آماده پایتونی کتابخانهٔ mlxtend استفاده شده است. کد مورد استفاده و نمودار مورد نظر بصورت زیر هستند. همانطور که قابل مشاهده است داده‌های به طور مناسبی داخل مرزهای تصمیم گیری قرار دارند.

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_pca, y, clf=LR_model_pca)
```

Code 11: Import drawdata library



شکل ۱۱: نتایج آموزش و تست SGD

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

ابتدا لازم است کتابخانهٔ مورد نیاز برای استفاده از این ابزار را فراخوانی کنیم.

```
!pip install drawdata
from drawdata import ScatterWidget
```

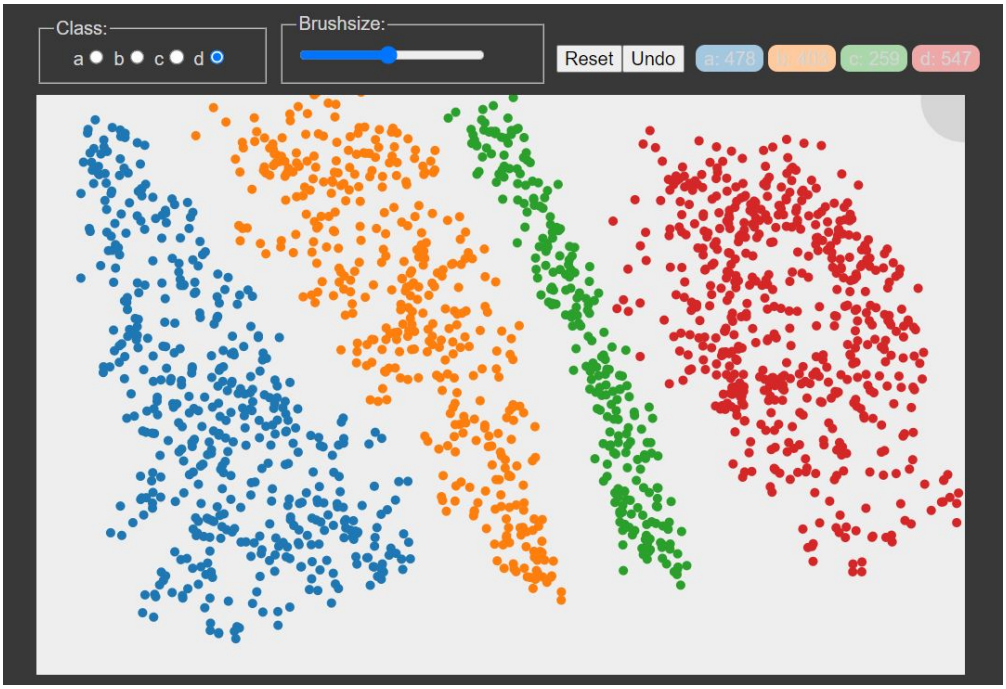
Code 12: Import drawdata library

با استفاده از کد زیر می‌توان ابزار مورد نظر را فراخوانی کرد.

```
widget = ScatterWidget()
widget
```

Code 13: drawdata

پس از فراخوانی با ایجاد پنجره‌ای مشابه زیر دیتاست مورد نظر را طراحی می‌کنیم.



شکل ۱۲: دیتاست طراحی‌شده با drawdata

در این قسمت ابتدا دیتاست طراحی شده را بصورت یک دیتافریم درآورده و مقادیر عددی برای برچسب‌های این داده را متناسب با طبقهٔ آن دیتا به آن اختصاص می‌دهیم. همچنین دو ستون ابتدایی این مجموعه‌داده را که ویژگی‌های آن هستند را به مقادیر X و ستون انتهایی را به y عنوان برچسب اختصاص می‌دهیم.

```
# Get the drawn data as a list of dictionaries
widget.data
# Get the drawn data as a dataframe
draedata_df = widget.data_as_pandas

draedata_df.replace({'a': 1, 'b': 2, 'c': 3, 'd': 4}, inplace=True)
draedata_df

X = draedata_df.iloc[:, :2].values
y = draedata_df.iloc[:, 3].values
```

Code 14: creat dataframe

نیاز است که پیش از آموزش مجموعه‌دادهٔ موجود را به دو بخش آموزش و ارزیابی تقسیم کنیم. این کار بصورت زیر انجام می‌پذیرد. با استفاده از کد موجود در خط دوم نیز می‌توان به تعداد هر کدام از مجموعه‌داده‌های تقسیم شده پی برد.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=4)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Code 15: Split Datase

به عنوان اولین طبقه‌بند از LogesticRegression استفاده می‌کنیم. در این قسمت از مقادیر بهینه‌ای که کتابخانهٔ sklearn بصورت پیش‌فرض پیشنهاد داده است استفاده شده است.

```
LR_model = LogisticRegression(penalty='l2',
                              dual=False,
                              tol = 1e-05,
                              solver='lbfgs',
                              max_iter=1000,
                              C=1,
                              random_state=4)
LR_model.fit(X_train, y_train)
LR_model.predict(X_test)
print("LR_model train score = ", LR_model.score(X_train, y_train))
print("LR_model test score = ", LR_model.score(X_test, y_test))
```

Code 16: Train by LR

نتیجهٔ آموزش و ارزیابی بصورت زیر قابل مشاهده است.

```
LR_model train score =  1.0
LR_model test score =  1.0
```

شکل ۱۳: نتیجهٔ آموزش و ارزیابی با استفاده از Regression Logestic

برای آموزش SGD از حلقه‌های تکرار برای پیدا کردن بهینه‌ترین پارامترها استفاده کردیم.

```
loss_list = ['hinge', 'log_loss', 'squared_hinge', 'squared_error', 'huber']
learning_rate = ['optimal', 'adaptive']
best_result = 0
for loss in loss_list:
  for lr in learning_rate:
    for eta in np.linspace(1e-3, 30, 50):
      SGD_model = SGDClassifier(loss=loss,
                                max_iter =2000,
                                tol = 1e-3,
                                eta0 = eta,
                                learning_rate = lr,
                                random_state=4)
      SGD_model.fit(X_train, y_train)
      result = SGD_model.score(X_train, y_train)
      if result > best_result:
        best_result = result
        print (f"best result is {best_result} with loss: {loss} , learning
   rate= {lr} and eta0= {eta}")
```

Code 17: Train by SGD

و نتایج حاصله به صورت زیر درآمد.

```
best result is 0.396590066716086 with loss: hinge , learning rate= optimal and eta0= 0.001
best result is 0.5982209043736101 with loss: hinge , learning rate= adaptive and eta0= 0.001
best result is 0.8495181616011861 with loss: hinge , learning rate= adaptive and eta0= 0.6132244897959184
best result is 0.8628613787991104 with loss: hinge , learning rate= adaptive and eta0= 1.2254489795918366
best result is 0.8747220163083765 with loss: hinge , learning rate= adaptive and eta0= 3.0621224489795917
best result is 0.8776871756856931 with loss: hinge , learning rate= adaptive and eta0= 5.511020408163266
best result is 0.8888065233506302 with loss: hinge , learning rate= adaptive and eta0= 14.69438775510204
best result is 0.8977020014825797 with loss: squared_hinge , learning rate= adaptive and eta0= 0.001
```

شکل ۱۴: نتیجهٔ آموزش با استفاده از SGD

درنهایت نتایج آموزش و ارزیابی به صورت زیر قابل مشاهده است.

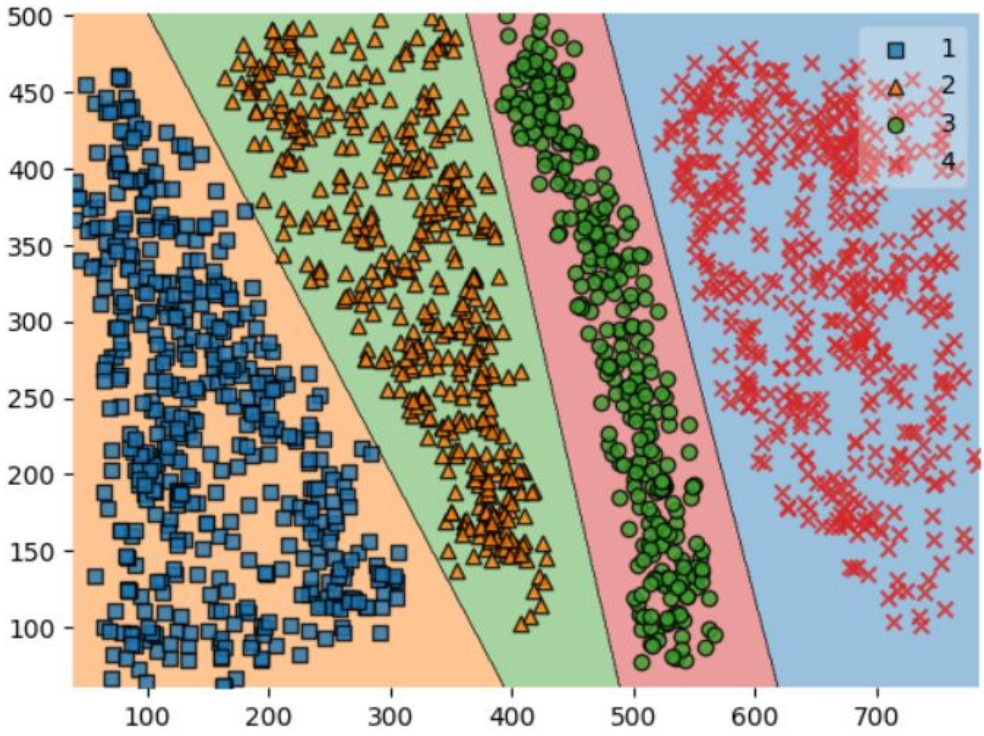
```
SGD_model train score =  0.8977020014825797
SGD_model test score =  0.8816568047337278
```

شکل ۱۵: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

نمودار نهایی داده‌ها به همراه مرز تصمیم‌گیری در این قسمت بصورت استفاده از کتابخانهٔ آماده پایتونی مشابه قبل صورت می‌گیرد.

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y, clf=LR_model)
```

Code 18: Train by SGD



شکل ۱۶: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

۲ عنوان سوال دوم

۱.۲ با مراجعه به صفحهٔ دیتاست CWRU Bearing با یک دیتاست مربوط به حوزهٔ «تشخیص عیب» آشنا شوید. با جستجوی آن در اینترنت و مقالات، توضیحاتی از اهداف، ویژگی ها و حالت های مختلف این دیتاست ارائه کنید.

داده‌ها برای یاتاقان‌های معمولی و عیب‌های در درایو تک‌نقطه‌ای و در انتهای فن جمع‌آوری شد. داده‌ها در ۱۲۰۰۰ نمونه در ثانیه و ۴۸۰۰۰ نمونه در ثانیه برای آزمایش‌های یاتاقان انتهایی درایو جمع‌آوری شد. تمام داده های بلبرینگ انتهای فن در ۱۲۰۰۰ نمونه در ثانیه جمع آوری شد

هدف استفاده از این نوع دیتاست، تعمیر و نگهداری پیش‌بینی‌کننده در ماشین‌ها پیش‌بینی خرابی‌ها است. در ماشین های دوار، قطعه ای که بیشترین آسیب را متحمل می شود، بلبرینگ ها هستند. هدف اصلی این تحقیق تشخیص خرابی بلبرینگ با استفاده از حداقل مجموعه مشاهدات و انتخاب حداقل تعداد ویژگی است. مجموعه آزمایشی برای جمع آوری داده ها شامل یک موتور القایی الکتریکی ۰.۳.۲ اسب بخار در یک سر، یک مبدل گشتاور در وسط، و یک دینامومتر در انتهای دیگر است که بار را شبیه سازی می کند. سنسورها، به‌ویژه شتاب‌سنج‌ها، روی یاتاقان‌های انتهای شفت موتور و روی فن داخل محفظه موتور قرار گرفتند و ۱۲۰۰۰ و ۴۸۰۰۰ نمونه در ثانیه از عیوب را جمع‌آوری کردند. سرعت و قدرت از طریق مبدل گشتاور بدست آمد و به صورت دستی ثبت شد.

نقص هایی با استفاده از تخلیه الکتریکی به بلبرینگ های SKF اضافه شد که باعث خرابی در تاج داخلی، توپ ها و مسیر بیرونی با قطرهای مختلف از ۰.۰۰۷ اینچ تا ۰.۰۴ اینچ شد. علاوه بر این، این آزمایش ها با تغییر سرعت چرخش و بار انجام شد. مجموعه داده شامل ۱۶۱ رکورد است که به چهار گروه تقسیم می شوند: ۴۸k خط پایه معمولی (داده های بدون خطا)، ۴۸k خطای انتهای درایو، ۱۲k خطای انتهای درایو و ۱۲k خطای انتهای فن. آنها حاوی اطلاعاتی بر اساس بارها و سرعت موتور هستند. در مورد نام فایل ها، حرف اول نشان دهنده موقعیت نقص، سه عدد بعدی قطر خرابی و آخرین عدد نشان دهنده بار است. به عنوان مثال، فایل IR۰۰۷-۰ دارای اطلاعات خرابی تاج داخلی است، با قطر شکست ۰.۰۰۷ اینچ برای بار موتور ۰ اسب بخار. هر فایل داده در مجموعه داده CWRU از داده هایی با طول های مختلف تشکیل شده است و مضرب ۲ نیست، علاوه بر این مجموعه ای بزرگ، متنوع و پیچیده است. [b۴]

۲.۲ قسمت دوم

۱.۲.۲ از هر کلاس نمونه با طول N جدا کنید (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 × از داده های هر کلاس استخراج کنید.

ابتدا مجموعه‌داده‌های مورد خواسته مسئله و کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم. سپس دو دیتاست سالم و خطا را در متغیرهای مورد نظر وارد می‌کنیم. با استفاده از ویژگی key می‌توان به سرستون‌های هر کدام از دستاست‌ها دسترسی پیدا کرد.

```
!gdown 14JXJeFLn-V1KSM5yNAiwokaX1F9P3haS
!gdown 1oqLyBpuEfXOpYIZ-aY_gzznCPsBz1pGp

from scipy.io import loadmat
import numpy as np
```

```
from scipy.stats import skew, kurtosis
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

# Load the .mat file
normal_data = loadmat('/content/97.mat')
fault_data = loadmat('/content/105.mat')

print(normal_data.keys())
print(fault_data.keys())
```

Code 19: import dataset and library

خروجی سرستون‌های دیتاست‌ها بصورت زیر است.

```
dict_keys(['__header__', '__version__', '__globals__', 'X097_DE_time', 'X097_FE_time', 'X097RPM'])
dict_keys(['__header__', '__version__', '__globals__', 'X105_DE_time', 'X105_FE_time', 'X105_BA_time', 'X105RPM'])
```

شکل ۱۷: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

سپس از مجموعه‌دادهٔ سالم ستون X۰۹۷-DE-time و از مجموعه‌دادهٔ خطا ستون X۱۰۵-DE-time را به عنوان نمونه با توجه به خواسته مسئله انتخاب می‌کنیم و در متغیرهای مناسب می‌ریزیم. برای داشتن دید مناسب از ابعاد مجموعه‌داده ابتدا نوع دیتای ستون‌های مورد نظر و همچنین ابعاد آن‌ها را خروجی می‌گیریم.

```
normal_data_variable = normal_data['X097_DE_time']
print(type(normal_data_variable))  # Print the type of the variable
print(normal_data_variable.shape)  # Print the shape of the variable (if it's
    a numpy array)

fault_data_variable = fault_data['X105_DE_time']
print(type(fault_data_variable))  # Print the type of the variable
print(fault_data_variable.shape)  # Print the shape of the variable (if it's a
     numpy array)
```

Code 20: import dataset and library

خروجی مورد نظر به صورت زیر است.

در این با انتخاب تعداد نمونه برابر ۱۲۰ و اندازه طول ۲۲۰ از هر کلاس جداسازی شده است. این ۱۲۰ نمونه با طول ۲۲۰ به صورت پشت سر هم قرار گرفته اند. در نهایت ابعاد هر کدام برابر (۱،۲۶۴۰۰) می‌شود.

```
<class 'numpy.ndarray'>
(243938, 1)
<class 'numpy.ndarray'>
(121265, 1)
```

شکل ۱۸: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

```python
# Extract 10 samples, each containing 5 data points
n_samples = 120
n_data = 220
ext_normal_data = normal_data_variable[:n_samples * n_data, 0]
ext_fault_data = fault_data_variable[:n_samples * n_data, 0]

print(ext_normal_data.shape)
print(ext_fault_data.shape)
```

Code 21: import dataset and library

سپس برا تکمیل ابعاد مجموعه‌داده با استفاده از دستور ،reshape دیتاها به صورتی درمی‌آیند که سطرهای آن دربرگیرندهٔ ۱۲۰ نمونهٔ مورد خواسته مسئله باشد.

```python
# Reshape the extracted data to have 10 rows and 5 columns
ext_normal_data = ext_normal_data.reshape(n_samples, n_data)
ext_fault_data = ext_fault_data.reshape(n_samples, n_data)

print(ext_normal_data.shape)
print(ext_fault_data.shape)
```

Code 22: import dataset and library

خروجی این کد نیز بصورت زیر است.

```
(120, 220)
(120, 220)
```

شکل ۱۹: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

۲.۲.۲ در مورد اهمیت استخراج ویژگی در یادگیری ماشین توضیحاتی بنویسید. سپس، با استفاده از حداقل ۸ عدد از روش های ذکرشده در جدول ۱، ویژگی های دیتاست قسمت «۲ آ» را استخراج کنید و یک دیتاست جدید تشکیل دهید.

استخراج ویژگی فرآیند شناسایی و انتخاب مهم ترین اطلاعات یا ویژگی ها از یک مجموعه داده است. این مانند تقطیر عناصر ضروری است، کمک به ساده‌سازی و برجسته کردن جنبه‌های کلیدی و در عین حال فیلتر کردن جزئیات کمتر مهم. این راهی برای تمرکز بر آنچه واقعاً در داده ها مهم است.

استخراج ویژگی مهم است زیرا اطلاعات پیچیده را ساده تر می کند. در مواردی مانند یادگیری کامپیوتری، به یافتن مهم‌ترین الگوها یا جزئیات کمک می‌کند و با تمرکز بر آنچه در داده‌ها اهمیت دارد، رایانه‌ها را در پیش‌بینی یا تصمیم‌گیری بهتر می‌کند.

اهمیت استخراج ویژگی در یادگیری ماشین را می‌توان بصورت خلاصه اینطور بررسی کرد:

کاهش ابعاد: این کار باعث ساده‌تر شدن داده‌های با ابعاد بالا می‌شود، که محاسبات سریع‌تری را فراهم می‌کند و از بیش‌برازش (overfitting) جلوگیری می‌کند.

کاهش نویز: این کار با حذف ویژگی‌های بی‌اهمیت یا نویزی، عملکرد مدل را بهبود می‌بخشد.

بهبود عملکرد: با تمرکز بر ویژگی‌های مرتبط، مدل‌ها می‌توانند الگوهای داده را بهتری تشخیص دهند که منجر به دقت و عمومیت بهتر می‌شود.

قابل فهم بودن: استخراج ویژگی می‌تواند مدل‌ها را قابل فهم‌تر کند با تبدیل داده به یک فرمت قابل فهم‌تر.

کار با داده‌های غیر عددی: این اجازه می‌دهد از انواع داده‌های غیر عددی (مانند متن یا تصاویر) با تبدیل آن‌ها به یک فرمت مناسب برای الگوریتم‌های یادگیری ماشین استفاده شود.

ادغام دانش حوزه‌ای: استخراج ویژگی اغلب شامل دانش خاص حوزه است که می‌تواند اطلاعاتی را تقویت کند.

با توجه به جدول مسئله تعداد هشت عدد از روش‌های ذکرشده انتخاب شد. با استفاده از این این روش‌ها ویژگی‌های دیتاست مورد استخراج می‌شود. کد زیر نحوه محاسبه هر کدام از این روش‌ها و اعمال بر روی دیتاست نرمال را نشان می‌دهد. برای محاسبهٔ هر کدام از این روش‌ها از کتابخانهٔ numpy استفاده شده است. قسمت axis=۱ باعث می‌شود که الگوریتم مورد نظر بر روی سطرها که نشان‌دهندهٔ نمونه‌های دیتاست هستند اعمال شود. خروجی ابعاد هر کدام از الگوریتم در پایان پرینت شده است. در قسمت بعدی خروجی ویژگی‌های استخراج‌شده با دستور column-stack بصورت ستونی به یکدیگر متصل شدند تا دیتاستی با ابعاد (۸، ۱۲۰) ساخته شود. دی قسمت پایانی نیز یک ستون صفر به دیتاست اضافه شده است که نشان‌دهنده برچسب کلاس نرمال است.

```
#Normal data feature extraction
normal_standard_deviations = np.std(ext_normal_data, axis=1)
normal_skewnesses = skew(ext_normal_data, axis=1)
normal_kurtoses = kurtosis(ext_normal_data, axis=1)
normal_peak_to_peaks = np.ptp(ext_normal_data, axis=1)
normal_root_mean_squares = np.sqrt(np.mean(np.square(ext_normal_data), axis=1)
    )
normal_means = np.mean(ext_normal_data, axis=1)
normal_absolute_means = np.mean(np.abs(ext_normal_data), axis=1)
normal_peaks = np.max(ext_normal_data, axis=1)

print("Standard Deviations:", normal_standard_deviations.shape)
```

```
print("Skewnesses:", normal_skewnesses.shape)
print("Kurtoses:", normal_kurtoses.shape)
print("Peak to Peaks:", normal_peak_to_peaks.shape)
print("Root Mean Squares:", normal_root_mean_squares.shape)
print("Means:", normal_means.shape)
print("Absolute Means:", normal_absolute_means.shape)
print("Peaks:", normal_peaks.shape)

normal_ext_feature_dataset = np.column_stack((normal_standard_deviations,
    normal_skewnesses, normal_kurtoses, normal_peak_to_peaks,
                                normal_root_mean_squares, normal_means,
    normal_absolute_means, normal_peaks))


normal_ext_feature_dataset = np.hstack((normal_ext_feature_dataset, np.zeros((
    len(normal_ext_feature_dataset), 1))))
print("final normal dataset: ", normal_ext_feature_dataset.shape)
```

Code 23: import dataset and library

خروجی کد مورد استفاده بصورت زیر است. در نهایت دیتاست مربوط به داده‌های نرمال با ابعاد (۹، ۱۲۰) که هم دارای ویژگی‌ها است و هم دارای برچسب در ستون اخر خود، قابل بهره‌برداری است.

```
Standard Deviations: (120,)
Skewnesses: (120,)
Kurtoses: (120,)
Peak to Peaks: (120,)
Root Mean Squares: (120,)
Means: (120,)
Absolute Means: (120,)
Peaks: (120,)
final normal dataset:  (120, 9)
```

شکل ۲۰: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

برای دیتاست خطا نیز همین مسیر مورد استفاده قرار گرفته است که کد و خروجی آن بصورت زیر قابل مشاهده است.

```
#fault data feature extraction
fault_standard_deviations = np.std(ext_fault_data, axis=1)
fault_skewnesses = skew(ext_fault_data, axis=1)
fault_kurtoses = kurtosis(ext_fault_data, axis=1)
fault_peak_to_peaks = np.ptp(ext_fault_data, axis=1)
```

```
fault_root_mean_squares = np.sqrt(np.mean(np.square(ext_fault_data), axis=1))
fault_means = np.mean(ext_fault_data, axis=1)
fault_absolute_means = np.mean(np.abs(ext_fault_data), axis=1)
fault_peaks = np.max(ext_fault_data, axis=1)

print("Standard Deviations:", fault_standard_deviations.shape)
print("Skewnesses:", fault_skewnesses.shape)
print("Kurtoses:", fault_kurtoses.shape)
print("Peak to Peaks:", fault_peak_to_peaks.shape)
print("Root Mean Squares:", fault_root_mean_squares.shape)
print("Means:", fault_means.shape)
print("Absolute Means:", fault_absolute_means.shape)
print("Peaks:", fault_peaks.shape)

fault_ext_feature_dataset = np.column_stack((fault_standard_deviations,
    fault_skewnesses, fault_kurtoses, fault_peak_to_peaks,
                            fault_root_mean_squares, fault_means,
    fault_absolute_means, fault_peaks))

fault_ext_feature_dataset = np.hstack((fault_ext_feature_dataset, np.ones((len
    (fault_ext_feature_dataset), 1))))
print("final fault dataset: ", fault_ext_feature_dataset.shape)
```

Code 24: import dataset and library

```
Standard Deviations: (120,)
Skewnesses: (120,)
Kurtoses: (120,)
Peak to Peaks: (120,)
Root Mean Squares: (120,)
Means: (120,)
Absolute Means: (120,)
Peaks: (120,)
final fault dataset:  (120, 9)
```

شکل ۲۱: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

در نهایت برای این که دیتاست تشکیل شده مانند یک دیتاستی که دارای کلاس‌های مختلف است به صورت استاندارد دربیاید، دو ماتریس تشکیل شده با هم ادغام شده است. کد و خروجی آن بصورت زیر قابل بررسی است.

```
final_data = np.concatenate((normal_ext_feature_dataset,
    fault_ext_feature_dataset), axis=0)
final_data.shape
```

Code 25: import dataset and library

```
(240, 9)
```

شکل ۲۲: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

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

مخلوط کردن داده ها می تواند منجر به بهبود دقت مدل های پیش بینی و یادگیری ماشین شود. این فرآیند اغلب منجر به داده هایی با تنوع و اطلاعات بیشتری می شود که باعث افزایش دقت و قابلیت پیش بینی مدل های آموزش داده شده می شود. ادغام داده ها اغلب منجر به افزایش تنوع و کمیت داده های موجود می شود که این امر می تواند منجر به بهتر شدن تحلیل ها، کاهش انحرافات غیرمنتظره و افزایش اطمینان در تصمیم گیری ها شود.مخلوط کردن داده ها ممکن است اطلاعات جدید و مفیدی که از یک ترکیب منابع به دست می آید را ارائه دهد که ممکن است در انجام تحلیل های پیشرفته و تصمیم گیری های مهم مورد استفاده قرار گیرد.

بنابراین مخلوط‌کردن دیتاها دارای اهمیت فراوانی است. در این قسمت پس از مخلوط‌کردن دیتاها با دستور shuffle، دیتاهای مربوط به ویژگی‌های دیتاست (همه ستون‌ها به‌جز ستون اخر) به متغیر X و دیتای برچسب به متغیر y تعلق پیدا می‌کند. در نهایت دیتاها به دو دستهٔ آموزش و ارزیابی تقسیم می‌شوند. نسبت داده‌های آموزش به ارزیابی در این قسمت با توجه به اندازه کلی دیتاست برابر ۲۰ درصد در نظر گرفته شده است. کد مورد نظر و خروجی نهایی بصورت زیر است.

```
shuffled_final_data = shuffle(final_data)

X = shuffled_final_data[:, :-1]
y = shuffled_final_data[:, -1]

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Code 26: import dataset and library

اما این ابعاد برای برچسب‌ها در ادامه مشکل‌ساز خواهد بود بنابراین نیاز است که با تغییر ابعاد، آن را به صورتی در بیاوریم که بعد ۱ در قسمت ستون وارد شود. با استفاده از دستور reshape این کار را انجام می‌دهیم.

```
((216, 8), (24, 8), (216,), (24,))
```

شکل ۲۳: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

```
# Reshape y_train and y_test
y_train = np.reshape(y_train, (-1, 1))
y_test = np.reshape(y_test, (-1, 1))
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Code 27: import dataset and library

در نهایت ابعاد دیتاهای مورد نظر به صورت زیر در می‌آید.

```
((216, 8), (24, 8), (216, 1), (24, 1))
```

شکل ۲۴: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

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

نرمال کردن داده ها برای منطقی کردن مقایسه پذیری و بهبوذ عملکرد الگوریتم های یادگیری ماشین و تسهیل فرآیند بهینه سازی و کمک به جلوگیری از overfitting صورت می گیرد. واحدها و مقیاس های مختلف برای ویژگی ها مکن است باعث کاهش دقت و عدم قابلیت مقایسه و تفسیر شود. نرمال کردن داده ها باعث می شود تمام ویژگی ها به یک مقیاس یا بازه مشابه تبدیل شوند که قابل مقایسه تر و تفسیر پذیرتر باشند. برای مدل های با پارامتر زیاد، اگر داده ها نرمال نشوند، احتمال overfitting بیشتر می شود؛ زیرا مدل ممکن است به اندازه ی زیادی به داده های با ویژگی های بزرگتر (به دلیل مقیاس بزرگتر) وابستگی پیدا کند و از یادگیری الگوهای کلی دور شود.

روش اول Scaling Max-Min : در این روش، داده‌ها به گونه‌ای تغییر می‌کنند که حداقل و حداکثر آنها به ترتیب به یک مقدار نگاشته می‌شوند. فرمول نرمال‌سازی Max-Min برای یک داده X به صورت زیر است:

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \quad (۱)$$

روش دوم Normalization score-Z : این روش با توجه به میانگین و انحراف معیاری داده ها، آن ها را به صورتی نرمال می کند که میانگین آن ها صفر و انحراف معیاری یک باشد.

$$X_{\text{normalized}} = \frac{X - \mu}{\sigma} \quad (۲)$$

نیاز است که فرایند نرمال‌سازی پس از تقسیم داده‌ها به آموزش و تست صورت بگیرد چرا که داده‌های تست نباید توسط مدل دیده شوند و با اعمال ویژگی آن ها در فرایند نرمال سازی، برخی ویژگی‌ها وارد مدل می‌شود و ارزیابی صورت گرفته درست نخواهد بود. درنتیجه فرآیند نرمال‌سازی با استفاده از اطلاعات داده‌ها آموزش بر روی داده‌های آموزش و ارزیابی صورت می‌پذیرد.

برای نرمال‌سازی داده‌ها در این سوال از روش اول استفاده می‌کنیم. کد مورد استفاده بصورت زیر است.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit the scaler on the training data
scaler.fit(x_train)

# Transform the training and testing data
x_train = scaler.transform(x_train)
x_test = scaler.traansform(x_test)
```

Code 28: import dataset and library

۳.۲ بدون استفاده از کتابخانه های آمادهٔ 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 نتیجهٔ 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 مرحلهٔ ارزیابی با قطعیت در مورد عمل کرد مدل نظر داد؟ چرا و اگر نمی توان، راه حل چیست؟

طبق تابع های موجود برای ایجاد hat-y با استفاده از تابع sigmoid به شکل زیر عمل می کنیم و ابتدا تابع sigmoid را تعریف می کنیم و سپس hat-y را ایجاد می کنیم. با استفاده از hat-y تابع logisticregression که ضرب هاx و هاw است را تشکیل می دهیم و خروجی آن hat-y می شود که همان ایy است که ما در شبکه خود ایجاد کرده ایم. سپس تابع اتلاف را با loss-log معرفی کرده و گرادیان آن را محاسبه می کنیم و eta که همان ضریب یادگیری است را در grads که حاصل ضرب x در اختلافy اصلی ما با y بدست آمده است، ضرب کرده و w را هر سری با آن آپدیت می کنیم تا به w درست برسیم

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

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

```

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

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

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

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

Code 29: import dataset and library

در ادامه برای در نظر گرفتن بایاس یک ستون تماما یک به ان اضافه می‌کنیم. در اینجا خروجی گرفتن ابعاد دیتاست دارای اهمیت است چون با هر بار اجرای این سلول سک ستون به دیتا اضافه می‌شود. کد و خروجی مورد نظر بصورت زیر است.

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

Code 30: import dataset and library

```
(216, 9)
```

شکل ۲۵: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

در این قسمت نیاز است تا پیش از شروع آموزش برخی پارامترهای مورد نیاز تعریف شوند. مقداردهی اولیه وزن‌ها با توجه به تعداد ویژگی‌ها دیتاست و با درنظر گرفتن بایاس صورت می‌گیرد. پارامترهای مورد نیاز تعریف و یک لیست خالی برای ذخیره کردن مقدار خطا در هر مرحله ایجاد می‌شود. سپس در یک حلقهٔ تکرار فرآیند آموزش اجرا می‌شود.

```
m = 8 #num of features
```

```
w = np.random.randn(m+1, 1)

num_epoch = 1000
eta = 0.01

error_hist = []

for epoch in range(num_epoch):
  y_hat = logestic_regression(x_train, w)
  loss = bce(y_train, y_hat)
  error_hist.append(loss)
  grads = gradient(x_train, y_train, y_hat)
  w = gradient_descent(w, eta, grads)
  if epoch % 50 == 0:
    print(f'Epoch = {epoch+1}, \t loss = {loss:.4},\t w={w.T[0, 0]}')
```

Code 31: import dataset and library

خروجی آموزش به صورت زیر گزارش شده است. کد طوری نوشته شده است که هر ۵۰ ایپاک یکبار خروجی تعیین‌شده نمایان شود. همانطور که قابل مشاهده است مقدار خطا در هر مرحله دارای شیب کاهشی است که نشان‌دهندهٔ فرآیند مناسب آموزش است. وزن‌ها نیز در هر مرحله دارای تغییر و حرکت به سمت مقادیر بهینه‌اند.

```
Epoch = 1,      loss = 1.705,   w=-0.8901391594909194
Epoch = 51,     loss = 1.268,   w=-0.7314287370227337
Epoch = 101,    loss = 0.9389,  w=-0.6116008682208355
Epoch = 151,    loss = 0.7309,  w=-0.5384661548822829
Epoch = 201,    loss = 0.6119,  w=-0.5077540513480393
Epoch = 251,    loss = 0.5418,  w=-0.5078634612413715
Epoch = 301,    loss = 0.4957,  w=-0.5279823807819595
Epoch = 351,    loss = 0.4616,  w=-0.5605434166192694
Epoch = 401,    loss = 0.4342,  w=-0.6006668044127286
Epoch = 451,    loss = 0.4107,  w=-0.6452400506262735
Epoch = 501,    loss = 0.39,    w=-0.6922583629886134
Epoch = 551,    loss = 0.3714,  w=-0.7404144533029205
Epoch = 601,    loss = 0.3544,  w=-0.7888486819568431
Epoch = 651,    loss = 0.3387,  w=-0.8369946942048667
Epoch = 701,    loss = 0.3242,  w=-0.884481770148694
Epoch = 751,    loss = 0.3108,  w=-0.9310715404646281
Epoch = 801,    loss = 0.2983,  w=-0.9766160796975121
Epoch = 851,    loss = 0.2867,  w=-1.0210296489878985
Epoch = 901,    loss = 0.2758,  w=-1.0642693611264002
Epoch = 951,    loss = 0.2657,  w=-1.106321796993851
```

شکل ۲۶: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

برای رسم نمودار تابع اتلاف از دستور plt-plot استفاده می‌کنیم و نتیجهٔ آموزش نیز با استفاده از تابع تعریف شده در قسمت قبل

دارای محاسبه است.

```
import matplotlib.pyplot as plt
acc_train = accuracy(y_train, y_hat)
plt.plot(error_hist)
```

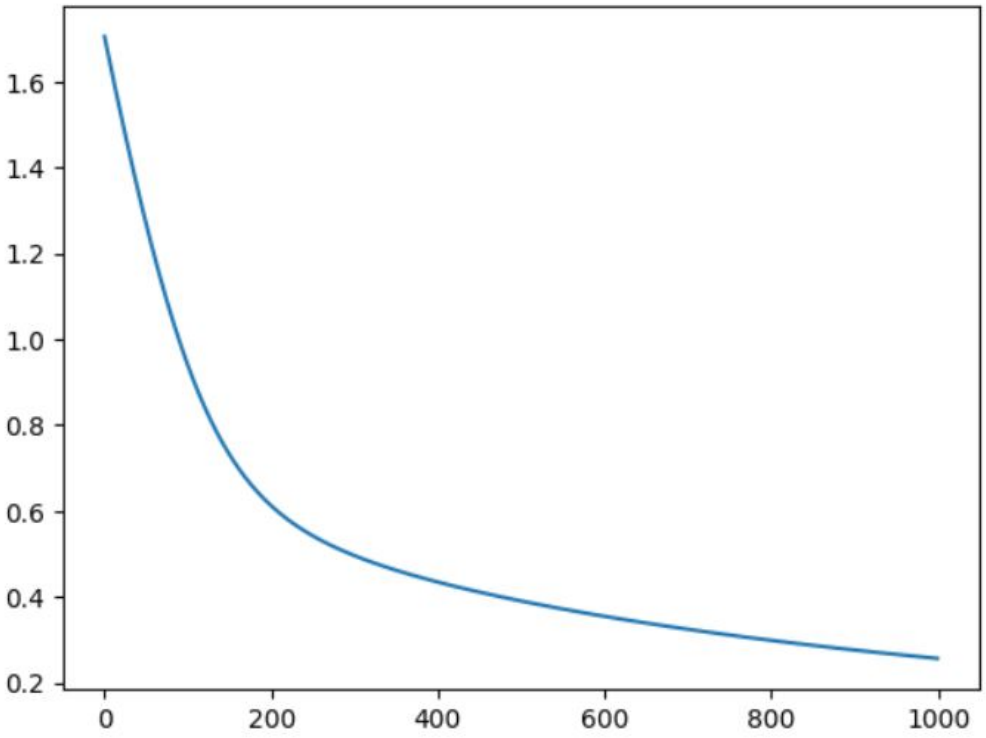
Code 32: import dataset and library

مقدار acc برای داده‌های آموزش بصورت زیر است.

```
train accuracy =  1.0
```

شکل ۲۷: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

همچنین نمودار تابع اتلاف بصورت زیر قابل مشاهده است.



شکل ۲۸: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

همانطور که در تابع اتلاف مشاهده می شود، با هر بار آموزش مقدار اختلاف hat-y که همان خروجی مد نظر ما است از train-y که خروجی شبکه عصبی ما است، کمتر می شود و مقدار خطای ما کمتر می شود و به حدود صفر می رسد و نشان دهنده ی آن است که آموزش شبکه ما به درستی کار کرده است و رو به بهتر شدن می رود. هدف اصلی در این مسئله کمینه کردن مقدار تابع اتلاف است، به طوری که مدل توانایی خود را در پیش بینی یا تطبیق با داده های ورودی بهینه کند.در طول فرآیند آموزش، مدل بهبود می یابد و توانایی

پیش بینی بهتری را نسبت به داده های ورودی پیدا می کند. این به معنای این است که تابع اتلاف کاهش می یابد و مدل به سمت کمینه کردن خطا یا اختلاف بین خروجی مدل و مقادیر واقعی هدایت می شود. البته با توجه به ظاهر تابع اتلاف می‌شود این برداشت را کرد که شبکه امکان آموزش‌دیدن بیشتر را با افزایش تعداد ایپاک دارد اما به دلیل همگرا شدن و accuracy بالا نیازی به آموزش بیشتر وجود ندارد.

تمام مراجل بالا را برای داده های ارزیابی نیز تکرار می کنیم با این تفاوت که این بار هاw آپدیت نمی شوند و مقدار آخرین w بدست آمده در فرآیند آموزش را به عنوان ورودی برای داده های ارزیابی لحاظ می کنیم. برای محاسبهٔ نتیجهٔ ارزیابی از دو شاخصهٔ accuracy که تابع محاسباتی آن را قبلا تعریف کردیم و همچنین f۱-score از کتابخانهٔ sklearn-metrics استفاده می‌کنیم.

```
x_test = np.hstack((np.ones((len(x_test), 1)), x_test))

y_hat_test = logestic_regression(x_test, w)

acc_test = accuracy(y_test, y_hat_test)

from sklearn.metrics import f1_score
f1score = f1_score(y_test, np.round(y_hat_test))

print("test accuracy:", acc_test)
print("test f1score:", f1score)
```

Code 33: import dataset and library

نمودار تابع اتلاف که در طول زمان آموزش شبکه عصبی رسم می شود، اطلاعات مفیدی را ارائه می دهد اما تنها از روی آن نمی توان به طور کامل و با قطعیت ظر دقیقی در مورد عملکرد نهایی مدل ارائه داد. دلایل آن هم می تواند نوسان داشتن نمودار تابع اتلاف در مراحل ابتدایی آموزش که به دلیل فرآیند آموزش و تنظیم پارامترهای مدل است و ممکن است در ادامه بهبود یابند، باشد. همچنین نمودار تابع اتلاف معمولا فقط نمایانگر عملکرد مدل بر روی داده های آموزشی است و اطلاعاتی درباره ی عملکرد واقعی مدل بر روی داده های جدید یا داده هایی که مدل آن ها را ندیده است، فراهم نمی کند.

اگر بخواهیم راه حلی ارائه دهیم برای اینکه بتوانیم پیش از مرحلهٔ ارزیابی در مورد عملکرد شبکه اظهار نظر کنیم، می‌توانیم داده‌ها آموزشی را به دو قسمت آموزش و اعتبارسنجی تقسیم کنیم. نحوهٔ استفاده از این دستهٔ جدید از داده هم به این صورت است که در هر ایپاک آموزشی که روی داده‌های اموزشی شبکه در حال یادگیری است، داده‌های اعتبارسنجی با وزن‌های اموزش‌دیده در همان ایپاک مورد ارزیابی قرار می‌گیرد و این فرآیند در کل طول همه ایپاک‌ها انجام می‌شود. مقایسه نمودار آموزش و اعتبارسنجی می‌تواند اطلاعات بسیار مفیدی از روند آموزش و نحوهٔ یادگیری شبکه بدهد.

۴.۲ فرآیند آموزش و ارزیابی را با استفاده از یک طبقه بند خطی آمادهٔ پایتون (در model-linear.sklearn) (انجام داده و نتایج را مقایسه کنید. در حالت استفاده از دستورات آمادهٔ سایکیت لرن، آیا راهی برای نمایش نمودار تابع اتلاف وجود دارد؟ پیاده سازی کنید.

در این حالت از از طبقه بند آمادهٔ LogisticRegression استفاده می‌کنیم. با توجه به نتایج مناسبی که در مرحلهٔ عدم استفاده از کتابخانه‌های پایتونی به دست آمده بود، از پارامترهای پیش‌فرض استفاده می‌کنیم. کد این قسمت بصورت زیر است.

```
from sklearn.linear_model import LogisticRegression
y_train = y_train.ravel()
LR_model = LogisticRegression(penalty='l2',
                              dual=False,
                              tol = 1e-05,
                              solver='sag',
                              max_iter=1000,
                              C=1,
                              random_state=4)
LR_model.fit(x_train, y_train)
LR_model.predict(x_test)
print("LR_model train score = ", LR_model.score(x_train, y_train))
print("LR_model test score = ", LR_model.score(x_test, y_test))
```

Code 34: import dataset and library

خروجی مورد نظر به این صورت قابل مشاهده است.

```
LR_model train score =  1.0
LR_model test score =  1.0
```

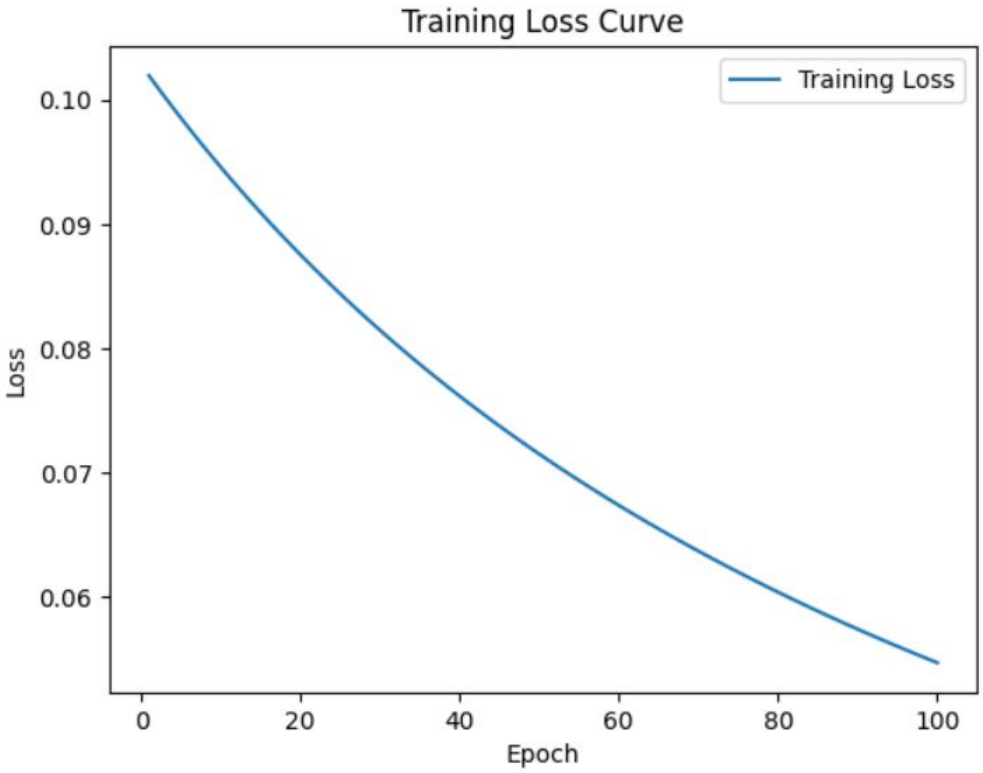
شکل ۲۹: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

بله. امکان نمایش نمودار تابع اتلاف در حالت استفاده از دستورات آماده وجود دارد. در این حالت در ابتدا تمامی کتابخانه های مورد نیاز را در محیط import می کنیم و با استفاده از کلاس SGDCeslassifier مدل مورد نظرمان را روی داده ها پیاده می کنیم و در داخل model می ریزیم. در ادامه تابعی با نام losses تعریف می کنیم که در ادامه مقادیر اتالفی را در داخل آن بریزیم. در این قسمت برای بهبودی عملکرد مقدار اپاک ها را زیاد و روی ۱۰۰۰ قرار می دهیم اما این مدل برای اپاک های کمتر از این هم جوابگو خواهد بود. در ادامه اپاک ها را به مدل مورد نظرمان اعمال می کنیم و تابع losses را append می کنیم.

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

```python
# Train the classifier and collect loss values
epochs = 100
for epoch in range(epochs):
    SGD_model.partial_fit(x_train, y_train, classes=np.unique(y_train))
    loss = log_loss(y_train, SGD_model.predict_proba(x_train))
    train_loss.append(loss)

# Plot the training loss curve
plt.plot(range(1, epochs + 1), train_loss, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.show()
```

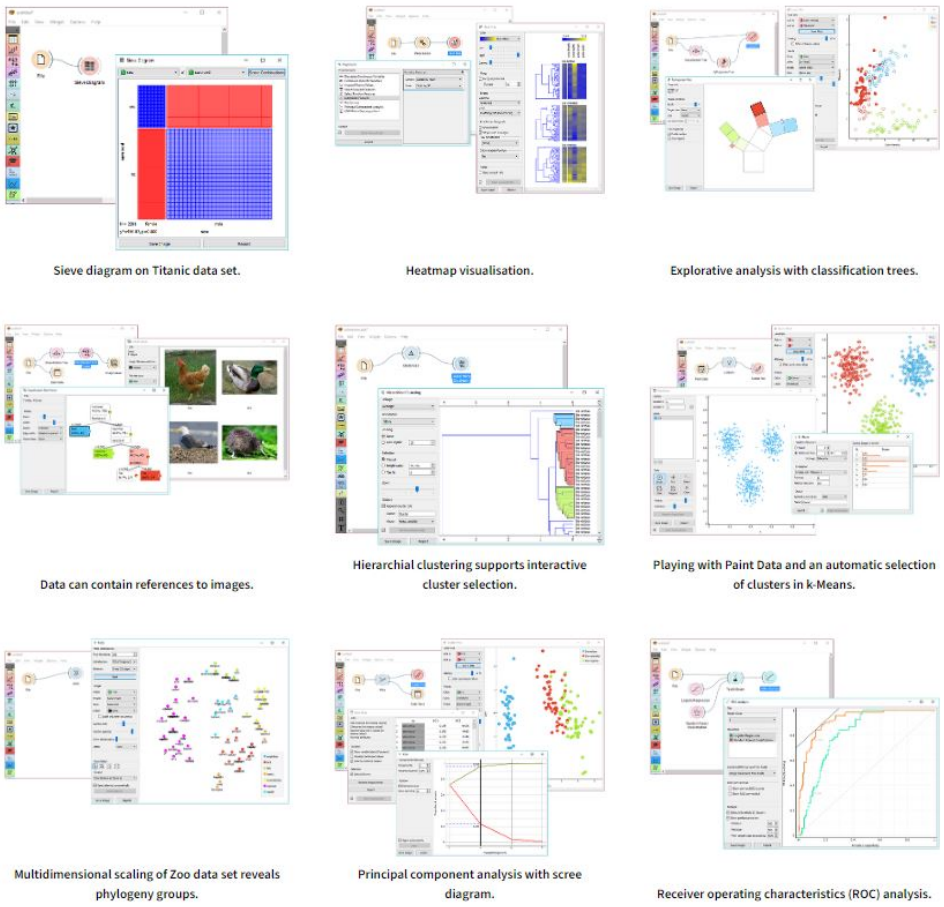
Code 35: import dataset and library



شکل ۳۰: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

۵.۲ در مورد نرم افزار داده کاوی Orange و قابلیت های آن تحقیق کنید و سعی کنید این سوال یا یک مثال ساده تر را با استفاده از این نرم افزار پیاده سازی کنید (راهنمایی: می توانید از پیوندهای ۱، ۲ و ۳ کمک بگیرید). پاسخ به این قسمت از سوال اختیاری و امتیازی است. می توانید عملکرد خود را به صورت تصویری و یا ویدیویی هم نشان دهید. مقدار نمرهٔ امتیازی، وابسته به جامعیت مثال بررسی شده و استفاده از ویژگی های مختلف این ابزار است.

این نرم‌افزار یک پلت فرم قدرتمند برای انجام تجزیه و تحلیل و تجسم داده ها، دیدن جریان داده ها و بهره وری بیشتر است. این یک پلت فرم منبع باز و تمیز و امکان افزودن قابلیت های بیشتر برای همه زمینه های علم را فراهم می کند. از جمله قابلیت‌های آن می‌توان به Heatmap visualisation, Receiver operating characteristics (ROC) analysis, Model-based feature scoring, CN2 rule induction اشاره کرد. در تصویر زیر به برخی دیگر از قابلیت‌های این نرم‌افزار اشاره شده است.



شکل ۳۱: نرم‌افزار Orange

ویدیوی عملکرد این نرم‌افزار در لینک زیر قابل دسترسی است.

https://drive.google.com/drive/folders/12pFGGgxIkyYdCL5pybK_CpyKH4jNExdw?usp=sharing

۳ سوال سوم

یک دیتاست در زمینهٔ آب و هوا با نام Weather in Szeged ۲۰۰۶-۲۰۱۶ را در نظر بگیرید. در این دیتاست هدف آن است که ارتباط بین Humidity با Temperature و هم چنین ارتباط بین Humidity و Apparent Temperature پیدا شده و با کمک داده های Humidity و Temperature تخمین انجام شود.

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

ابتدا کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
```

Code 36: import dataset and library

سپس فایل CSV دانلودی از مرجع مورد ذکر سوال را با دستور gdown فراخوانی می‌کنیم و آن را بصورت یک دیتافریم درمی‌آوریم.

```
%cd /content
!gdown 1fXFPECCGZ7-Kc8wXw8VhfGrsASlAOdAz

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

Code 37: import dataset and library

دیتافریم تشکیل‌شده بصورت زیر است.

هیت‌مپ ماتریس همبستگی که نشان‌دهندهٔ میزان ارتباط داده‌های هر کدام از ویژگی‌ها است به صورت زیر قابل پیاده‌سازی است. در دیتاست مورد نظر با توجه به اینکه داده‌های برخی از ستون‌ها دارای مقادیر مورد نظر برای اعمال برای رسم این ماتریس بودند، ابتدا ستون‌های مورد نظر شناسایی شدند و برای اعمال انتخاب شدند.

```
# Exclude the last column from the DataFrame
```

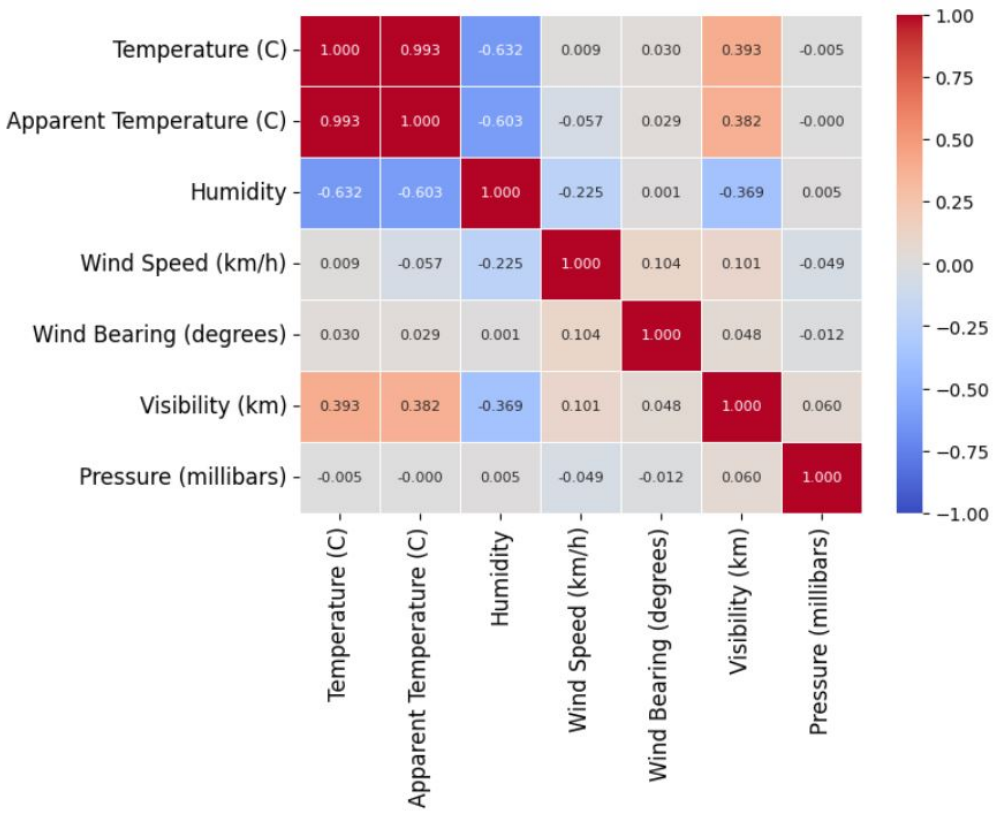
	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +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 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +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.
...	...	...	...	...	...	...	...	...	...	...	...	...

شکل ۳۲: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

```
feature_df = df.iloc[:, [3,4,5,6,7,8,10]]

# Calculate correlation matrix
corr_matrix = feature_df.corr()

# Create heatmap using seaborn with a colormap from -1 to 1
plt.figure(figsize=(10, 10))  # Adjust the figure size as needed
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5,
    annot_kws={"size": 8}, fmt='.3f',
            yticklabels=corr_matrix.columns, vmin=-1, vmax=1)  # Set vmin and
    vmax

# Adjust font size of annotations
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Adjust margins of PDF file
plt.savefig('PIcS1.pdf', bbox_inches='tight')
```

Code 38: import dataset and library

ماتریس همبستگی به صورت زیر است. همانطور که قابل مشاهده است، ویژگی‌های Temperature ، Apparent-Temperature دارای بیشتری همبستگی با یکدیگرند. به این معنی که داده‌های این دو وِیژگی ارتباط زیادی با یکدیگر دارند. همچنین دو ویژگی Hu-midity و Visibility نیز تا حدودی ارتباط بیشتری نسبت به سایر ویژگی‌ها با این دو ویژگی دارند.



شکل ۳۳: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

همچنین نمودار پراکندگی ویژگی‌های مختلف نسبت به یکدیگر بصورت زیر قابل مشاهده است.

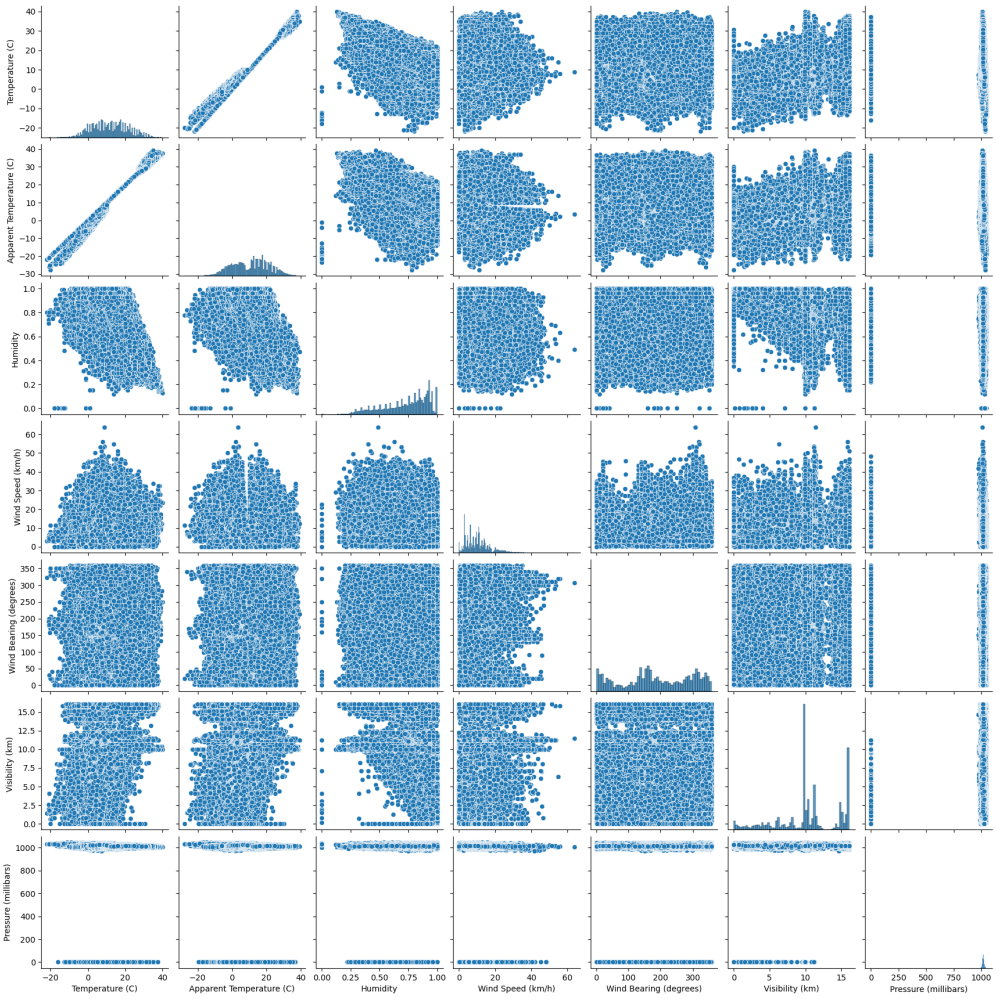
در صورت سوال تاکید شده است که از داده‌های مربوط به ویژگی‌های Temperature Humidity، و Apparent-Temperature استفاده شود اما با توجه به این که ویژگی Visibility نیز تا حدودی همبستگی مناسبی دارد، فعلا در ادامه کار روی این ویژگی نیز بررسی‌هایی را صورت می‌دهیم تا نتیجهٔ مناسبی رقم بخورد. با استفاده از این کد این ویژگی‌ها را بصورت جداگانه استخراج می‌کنیم.

```
Humidity = df['Humidity'].values
Temperature = df["Temperature (C)"].values
Apparent_Temperature = df["Apparent Temperature (C)"].values
Visibility = df["Visibility (km)"].values
```

Code 39: import dataset and library

در قسمت بعد ویژگی‌هایی که همبستگی بیشتری داشتند را از نظر هیستوگرام پراکندگی ویژگی‌ها بررسی می‌کنیم. کد زیر این هیستوگرام‌ها را رسم می‌کند. قابل مشاهده است که ویژگی‌های Temperature و Apparent-Temperature دارای پراکندگی ویژگی بسیار مشابه یکدیگراند. ویژگی Humidity دارای پراکندگی بیشتر در مقادیر نزدیک به یک است و ویژگی Visibility دارای پراکندگی بیشتری در حد متوسط آن است.

```
# Plot histogram for 'Temperature (C)'
```

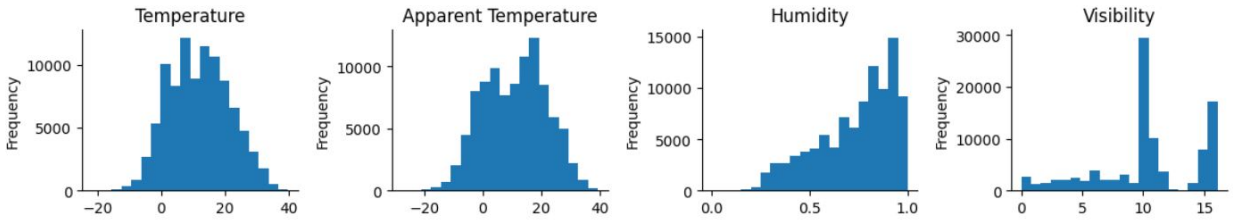


شکل ۳۴: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

```
plt.figure(figsize=(12, 2.3))
plt.subplot(1, 4, 1)
feature_df['Temperature (C)'].plot(kind='hist', bins=20, title='Temperature')
plt.gca().spines[['top', 'right']].set_visible(False)

# Plot histogram for 'Apparent Temperature (C)'
plt.subplot(1, 4, 2)
feature_df['Apparent Temperature (C)'].plot(kind='hist', bins=20, title='
    Apparent Temperature')
plt.gca().spines[['top', 'right']].set_visible(False)

# Plot histogram for 'Humidity'
```

```
plt.subplot(1, 4, 3)
feature_df['Humidity'].plot(kind='hist', bins=20, title='Humidity')
plt.gca().spines[['top', 'right']].set_visible(False)

# Plot histogram for 'Visibility'
plt.subplot(1, 4, 4)
feature_df['Visibility (km)'].plot(kind='hist', bins=20, title='Visibility')
plt.gca().spines[['top', 'right']].set_visible(False)

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()
```

Code 40: import dataset and library



شکل ۳۵: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

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

در صورت مسئله عنوان شده است که با کمک داده‌های Humidity ، Temperature تخمین صورت بپذیرد. بنابراین در این قسمت از این دو ویزگی به عنوان ویژگی‌هایی که شبکهٔ RL به عنوان ورودی می‌پذیرد استفاده می‌کنیم و ویژگی Apparent-Temperature را به عنوان خروجی که شبکه سعی در تخمین‌زدن آن دارد انتخاب می‌کنیم. ابتدا شبکهٔ RL مورد نظر را به صورت زیر فراخوانی می‌کنیم. در این قسمت از کد مورد استفاده در کلاس حل تمرین در این قسمت استفاده شده است.

```
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)
```

Code 41: import dataset and library

از ویژگی‌های Humidity و Temperature و Visbility مطابق صورت مسئله استفاده شده است و سعی می‌شود مدل بتواند تخمین مناسبی از Apparent-Temperature پیش‌بینی شود. سپس داده‌های فراهم شده به دو قسمت آموزش و ارزیابی تقسیم می‌شوند و از مدلی که پیش‌تر توضیح داده شد استفاده شده و پیش‌بینی و مقدار خطای MSE محاسبه می‌شود.

```
X = np.concatenate((Humidity, Temperature, Visibility), axis=1)
y = Apparent_Temperature

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=4)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print()


# Initialize and fit the linear regression model using least squares
model = LinearRegressionLS()
model.fit(X_train, y_train)

# Make predictions on the testing set
```

```
y_pred = model.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

Code 42: import dataset and library

خروجی نهایی به صورت زیر است. مقدار خطای MSE نشان‌دهندهٔ میزان نزدیکی مقادیر پیش‌بینی شده و مقدار ویژگی هدف است که هر چه این خطا کمتر باشد، یعنی مدل عملکرد بهتری در تخمین داشته است.

```
(77162, 3) (19291, 3) (77162, 1) (19291, 1)

Mean Squared Error: 1.5746506582916415
```

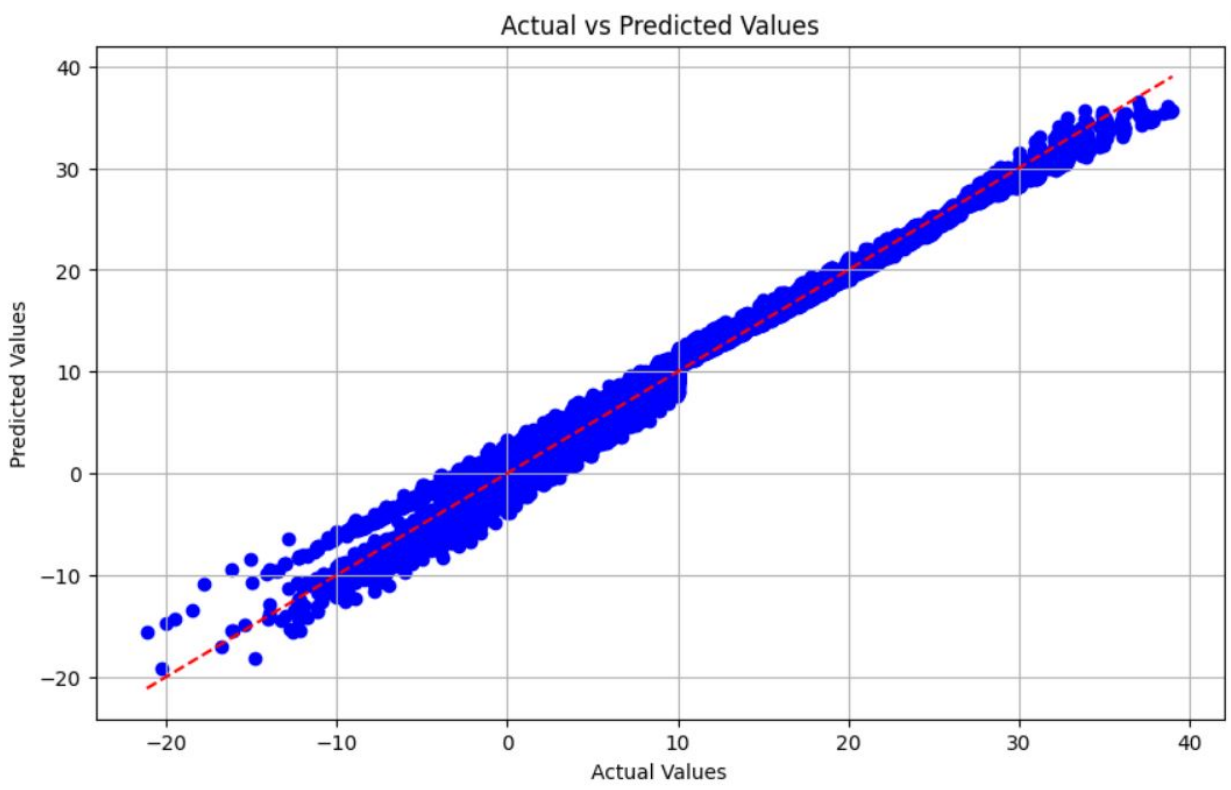
شکل ۳۶: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

به صورت زیر می‌توان نمودار مقادیر تخمینی شبکه را بر حسب خروجی هدف که قصد تخمین زدن آن را داشتیم رسم کرد. این نمودار حاوی مقادیر تمامی داده‌های مقادیر واقعی و تخمینی و خطی به عنوان خط تخمینی است.

```
# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='
    red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.grid(True)
plt.show()
```

Code 43: import dataset and library

برای روش RLS از کد زیر بهره می‌بریم. در این حالت یک کلاس تشکیل شده است که یک ورودی با عنوان forgetting-factor می‌گیرد. این ورودی با میزان به یادآوری مقادیر قبلی ارتباط دارد که هرچی این مقدار بیشتر باشد، به مقادیر جدیدتری که اخیرا شبکه تولید کرده است وابسته‌تر می‌شود. بهترین مقدار این پارامتر مطابق آزمایش‌ها برابر مقدار ۰.۹۹ یا ۰.۹ است که ما هم در این قسمت از همین مقدارهای اسفاده کرده‌ایم اما در ادامه صحت این ادعا را بررسی می‌کنیم.



شکل ۳۷: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

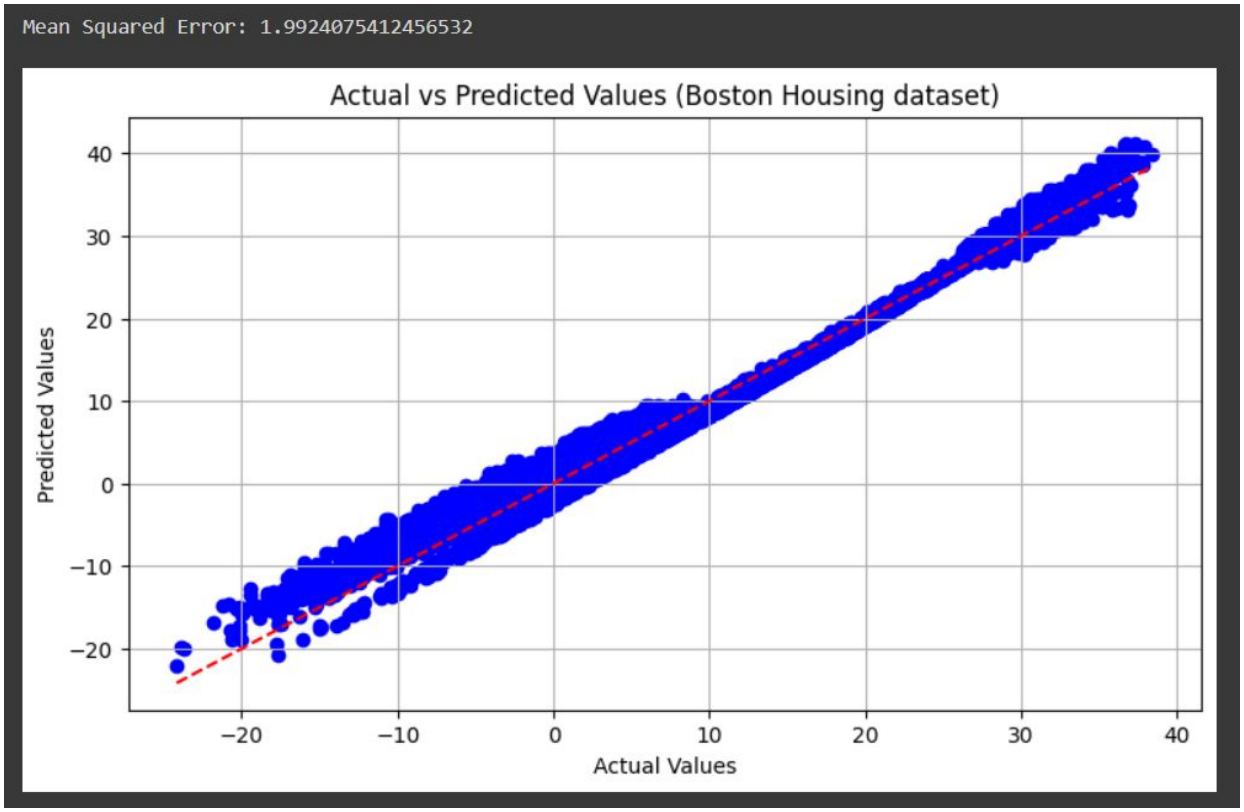
```python
class RecursiveLeastSquares:
    def __init__(self, n_features, forgetting_factor=0.99):
        self.n_features = n_features
        self.forgetting_factor = forgetting_factor
        self.theta = np.zeros((n_features, 1))  # Initialize model parameters
        self.P = np.eye(n_features)  # 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)

X = np.concatenate((Humidity, Visibility, Temperature), axis=1)
y = Apparent_Temperature

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the RLS model
rls = RecursiveLeastSquares(n_features=X_train.shape[1], forgetting_factor=0.99)
errors = rls.fit(X_train, y_train)

# Make predictions
y_pred = rls.predict(X_test)

# Calculate Mean Squared Error
mse = np.mean(np.array(errors)**2)
print("Mean Squared Error:", mse)

# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='
```

```
    red', linestyle='--')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values (Boston Housing dataset)')
plt.grid(True)
plt.show()
```

Code 44: import dataset and library

مقدار خطای MSE و نمودار مقدار تخمینی توسط شبکه بر حسب مقادیر واقعی در تصویر زیر قابل مشاهده است. مقدار خطا به مقادیر کم رسیده است و شبکه توانسته است خطی که به طور مناسبی بر مقادیر تخمینی و واقعی قرار می‌گیرد را شناسایی و رسم کند.



شکل ۳۸: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

اما برای بررسی صحت ادعای بهترین مقدار forgetting-factor در یک حلقهٔ یکرار مقادیر مختلف این پارامتر را مورد آزمایش قرار می‌دهیم.

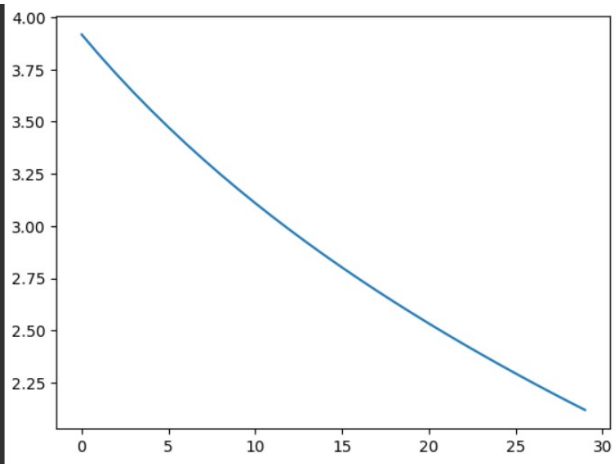
```
mse_list = []
for ff in np.linspace(0.5, 0.99, 30):
    # Initialize and fit the RLS model
```

```
    rls = RecursiveLeastSquares(n_features=X_train.shape[1], forgetting_factor
    = ff)
    errors = rls.fit(X_train, y_train)

    # Make predictions
    y_pred = rls.predict(X_test)

    # Calculate Mean Squared Error
    mse = np.mean(np.array(errors)**2)
    mse_list.append(mse)
    # print("Mean Squared Error:", mse, "           Forgetting Factor:", ff)
    print("Mean Squared Error: {:.10f}      Forgetting Factor: {:.10f}".format
   (mse, ff))
```

Code 45: import dataset and library

نتیجه به صورت زیر قابل مشاهده است که هر چه به سمت مقادیر نزدیک ۱ می‌رویم نتیجه بهبود می‌یابد.

```
Mean Squared Error: 4.7292146825      Forgetting Factor: 0.5000000000
Mean Squared Error: 4.5707946946      Forgetting Factor: 0.5168965517
Mean Squared Error: 4.4209272142      Forgetting Factor: 0.5337931034
Mean Squared Error: 4.2788022404      Forgetting Factor: 0.5506896552
Mean Squared Error: 4.1437247651      Forgetting Factor: 0.5675862069
Mean Squared Error: 4.0150943500      Forgetting Factor: 0.5844827586
Mean Squared Error: 3.8923863773      Forgetting Factor: 0.6013793103
Mean Squared Error: 3.7751353399      Forgetting Factor: 0.6182758621
Mean Squared Error: 3.6629209647      Forgetting Factor: 0.6351724138
Mean Squared Error: 3.5553578762      Forgetting Factor: 0.6520689655
Mean Squared Error: 3.4520890250      Forgetting Factor: 0.6689655172
Mean Squared Error: 3.3527824761      Forgetting Factor: 0.6858620690
Mean Squared Error: 3.2571305976      Forgetting Factor: 0.7027586207
Mean Squared Error: 3.1648504035      Forgetting Factor: 0.7196551724
Mean Squared Error: 3.0756838271      Forgetting Factor: 0.7365517241
Mean Squared Error: 2.9893970324      Forgetting Factor: 0.7534482759
Mean Squared Error: 2.9057784126      Forgetting Factor: 0.7703448276
Mean Squared Error: 2.8246355153      Forgetting Factor: 0.7872413793
Mean Squared Error: 2.7457915370      Forgetting Factor: 0.8041379310
Mean Squared Error: 2.6690820827      Forgetting Factor: 0.8210344828
Mean Squared Error: 2.5943526261      Forgetting Factor: 0.8379310345
Mean Squared Error: 2.5214568863      Forgetting Factor: 0.8548275862
Mean Squared Error: 2.4502563261      Forgetting Factor: 0.8717241379
Mean Squared Error: 2.3806207731      Forgetting Factor: 0.8886206897
Mean Squared Error: 2.3124298304      Forgetting Factor: 0.9055172414
Mean Squared Error: 2.2455765993      Forgetting Factor: 0.9224137931
Mean Squared Error: 2.1799824040      Forgetting Factor: 0.9393103448
Mean Squared Error: 2.1156476365      Forgetting Factor: 0.9562068966
Mean Squared Error: 2.0528209908      Forgetting Factor: 0.9731034483
Mean Squared Error: 1.9924075412      Forgetting Factor: 0.9900000000
```



شکل ۳۹: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

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

حداقل مربعات وزن دار یک تکنیک رگرسیونی است که با تخصیص وزن به مشاهدات بر اساس واریانس های تخمین زده شده، به بررسی ناهمسانی می پردازد و امکان برآورد پارامتر قوی تر و عملکرد مدل را بهبود می بخشد. به عبارتی دیگر حداقل مربعات وزنی (WLS) نوعی تغییر از روش حداقل مربعات معمولی (OLS) است که در تحلیل رگرسیون خطی استفاده می شود. در OLS، هر نقطه داده از نظر تأثیر آن بر تخمین ضرایب رگرسیون به طور مساوی رفتار می شود. با این حال، در برخی موارد، تغییرپذیری نقاط داده ممکن است برابر نباشد و برخی از مشاهدات ممکن است قابل اعتمادتر باشند یا واریانس کمتری نسبت به سایرین داشته باشند.

حداقل مربعات وزنی با تخصیص وزن به هر نقطه داده بر اساس واریانس یا پایایی آنها، به این موضوع می‌پردازد. ایده اصلی این است که به مشاهداتی که قابل اعتمادتر در نظر گرفته می شوند یا دارای تنوع کمتری هستند و وزن کمتری به مشاهداتی که کمتر قابل اعتماد تلقی می شوند یا دارای تنوع بالاتری هستند، وزن بیشتری بدهیم. این کار با اعمال ماتریس W به فرمول محاسباتی LS صورت می‌پذیرد که به شکل تصویر زیر اعمال می‌شود.

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_n \end{pmatrix}$$

$$\begin{aligned} \hat{\beta}_{WLS} &= \arg\min_{\beta} \sum_{i=1}^{n} \epsilon_i^{*2} \\ &= (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{Y} \end{aligned}$$

شکل ۴۰: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

برای اعمال این روش بر روی مجموعه‌دادهٔ مورد استفاده در این سوال از کد زیر استفاده می‌کنیم. در این حالت با تعریف ماتریس وزن از حالت LS به حالت WLS وارد می‌شویم و به این طریق وزن‌های مورد نظر را اعمال می‌کنیم.

```
X = np.concatenate((Humidity, Temperature, Visibility), axis=1)
y = Apparent_Temperature

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=4)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print()

```

```python
error_variance = 2  # 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_with_intercept = sm.add_constant(X_train)  # Add intercept term
model = sm.WLS(y_train, X_with_intercept, weights=weights)
result = model.fit()

# Print the model summary
print(result.summary())
```

Code 46: import dataset and library

خلاصه‌ای از پارامترهای شبکهٔ آموزش‌داده شده به صورت زیر استو همانطور که قابل مشاهده است مقادیر مانند R-squared و .Adj R-squared که به مقادیر نزدیک ۱ رسیده‌اند نشان‌دهندهٔ همگرا شدن الگوریتم مورد نظر به تخمین مناسب است.

```
(77162, 3) (19291, 3) (77162, 1) (19291, 1)
                            WLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.986
Model:                            WLS   Adj. R-squared:                  0.986
Method:                 Least Squares   F-statistic:                 1.866e+06
Date:                Fri, 12 Apr 2024   Prob (F-statistic):               0.00
Time:                        23:26:51   Log-Likelihood:            -1.2655e+05
No. Observations:               77162   AIC:                         2.531e+05
Df Residuals:                   77158   BIC:                         2.531e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -4.3176      0.031   -139.838      0.000      -4.378      -4.257
x1             2.2417      0.030     74.512      0.000       2.183       2.301
x2             1.1423      0.001   1833.235      0.000       1.141       1.143
x3            -0.0103      0.001     -8.686      0.000      -0.013      -0.008
==============================================================================
Omnibus:                     1603.053   Durbin-Watson:                   1.981
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2739.820
Skew:                          -0.183   Prob(JB):                         0.00
Kurtosis:                       3.847   Cond. No.                         171.
==============================================================================

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

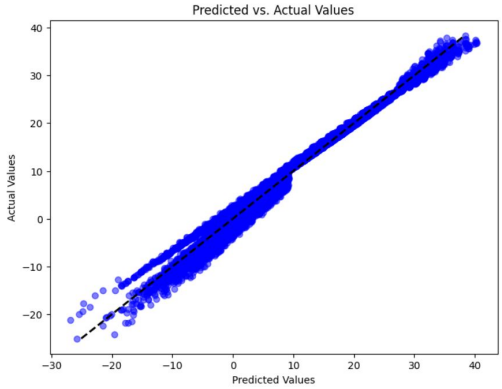
شکل ۴۱: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

برای اطمینان از صحت روش مورد استفاده، نیاز است که مشابه موارد قبلی نمودار تخمین بر اساس مقادیر واقعی رسم شود. قابل مشاهده است که تخمین مانند موارد قبلی همگرا شده و توانسته به طور مناسبی خطی منطبق و تخمینی بر داده‌های تخمینی بر اساس مقادیر واقعی رسم کند.

```python
X_test_with_intercept = sm.add_constant(X_test)
y_pred = result.predict(X_test_with_intercept)

# Plot Predicted vs. Actual Values
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, y_test, color='blue', alpha=0.5)
```

```
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw
    =2)  # Diagonal line for reference
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Predicted vs. Actual Values')
plt.show()
```

Code 47: import dataset and library



شکل ۴۲: نتیجهٔ آموزش و ارزیابی با استفاده از SGD

۴.۳ در مورد الگوریتم **QR-Decomposition-Based RLS** تحقیق کنید. پاسخ به این قسمت از سوال اختیاری و امتیازی است.

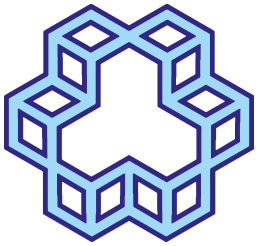
استفاده از تجزیه QR برای مثلثی کردن ماتریس داده های ورودی منجر به یک روش جایگزین برای اجرای روش حداقل مربعات بازگشتی (RLS) می‌شود. مزایای اصلی ناشی از الگوریتم حداقل مربعات بازگشتی مبتنی بر تجزیه QR، اجرای احتمالی آن در آرایه‌های سیستولیک و بهبود رفتار عددی آن با در نظر گرفتن اثرات کوانتیزاسیون است.

به عبارت دیگر حداقل مربعات بازگشتی مبتنی بر تجزیه روشی است که برای تخمین پارامتر در مسائل فیلتر تطبیقی و حداقل مربعات بازگشتی استفاده می شود. این تکنیک تجزیه QR را با به‌روزرسانی بازگشتی ترکیب می‌کند تا با در دسترس قرار گرفتن داده‌های جدید، راه‌حل حداقل مربعات را به طور مؤثر محاسبه کند.

الگوریتم RLS بر پایه Decomposition QR از ترکیب دو تکنیک استفاده می‌کند: Decomposition QR و Least Recursive Squares. این الگوریتم برای پردازش سیگنال‌های پویا و تخمین پارامترها استفاده می‌شود. در این الگوریتم، ابتدا ماتریس مورد نظر را با استفاده از روش Decomposition QR به دو ماتریس قراری (یا orthogonal) و ماتریس سه‌گانه بالایی (یا upper triangular) تجزیه می‌کنیم. سپس از روش Squares Least Recursive برای به‌روزرسانی پاسخ به صورت بازگشتی با هر آمدن داده جدید استفاده می‌کنیم، بدون نیاز به محاسبه مجدد کل پاسخ از ابتدا. این ترکیب از دو روش، باعث افزایش کارایی و کاهش پیچیدگی محاسباتی مسئله می‌شود، به‌ویژه زمانی که با مجموعه داده‌های بزرگ یا نیاز به پردازش به صورت زمان‌بندی شده روبرو هستیم.

۴ عنوان سوال چهارم

در این قسمت با نحوهٔ درج اشکال آشنا می‌شوید:



شکل ۴۳: شکل شماره ۱

۵ عنوان سوال پنجم

در این قسمت با نحوهٔ درج جداول آشنا می‌شوید:

جدول ۱: جدول شماره ۱

خانه شماره ۱	خانه شماره ۲	خانه شماره ۳
خانه شماره ۴	خانه شماره ۵	خانه شماره ۶
خانه شماره ۷	خانه شماره ۸	خانه شماره ۹

۶ عنوان سوال ششم

در این قسمت با نحوهٔ درج انواع لیست‌ها آشنا می‌شوید:

۱.۶ عنوان بخش اول سوال ششم

- مورد اول
- مورد دوم

۲.۶ عنوان بخش دوم سوال ششم

۱. مورد شماره ۱

۲. مورد شماره ۲

۷ عنوان سوال هفتم

در این قسمت با نحوه درج برنامه‌ها آشنا می‌شوید:

```python
# This program prints Hello, world!

print('Hello, world!')
```

Code 48: My Caption (Python)

```matlab
clc; clear all; close all;
disp('Hello world!')
```

Code 49: My Caption (MATLAB)

```cpp
// Your First C++ Program

#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Code 50: My Caption (C++)

```c
#include <stdio.h>
int main()
{
  printf("Hello world!\n");
  return 0;
}
```

Code 51: My Caption (C)

۸ عنوان سوال هشتم

در این قسمت با نحوهٔ ارجاع‌دادن آشنا می‌شوید.

۱.۸ عنوان بخش اول سوال هشتم

در این قسمت با نحوهٔ ارجاع به سایر منابع آشنا می‌شوید:
به صفحهٔ درس تشخیص و شناسایی عیب ارجاع داده می‌شود [b۱]. به این کتاب‌ها ارجاع داده می‌شود [b۲][b۳]. برای وارد کردن ارجاع می‌توانید از انتهای فایل `main.tex` استفاده کنید و یا با تغییر قالب مرجع‌نویسی، به فایل `bibliography.bib` مراجعه کرده و فرمت `bib.` را وارد کنید.

۲.۸ عنوان بخش دوم سوال هشتم

اگر می‌خواهید به یک شکل، جدول، یا بخش ارجاع دهید می‌توانید به دو صورتی که در ادامه آمده عمل کنید (حالت اول توصیه می‌شود):

۱. مورد شماره ۱: پاسخ سوال ۱، ؟؟، شکل ۴۳، جدول ۱، ۴.

۲. مورد شماره ۲: سوال اول.

۳.۸ عنوان بخش سوم سوال هشتم

اگر می‌خواهید به یک پایگاه اینترنتی ارجاع دهید، می‌توانید از این دستور هم استفاده کنید: گیتهاب (GitHub).

۹ ضمیمه

برای آشنایی بیشتر با LaTeX، با جست‌وجو در اینترنت منابع مفیدی خواهید یافت.

منابع

[۱] صفحهٔ درس یادگیری ماشین.

[2] Steven X. Ding, “Data-driven Design of Fault Diagnosis and Fault-tolerant Control System”, Springer, 2014.

[3] S. Theodoridis and K. Koutroumbas, ”Pattern recognition”, Fourth Edition, Academic Press, 2009.