به نام خداوند رنگین کمان

مینی پروژه اول

مبانی سیستم های هوشند

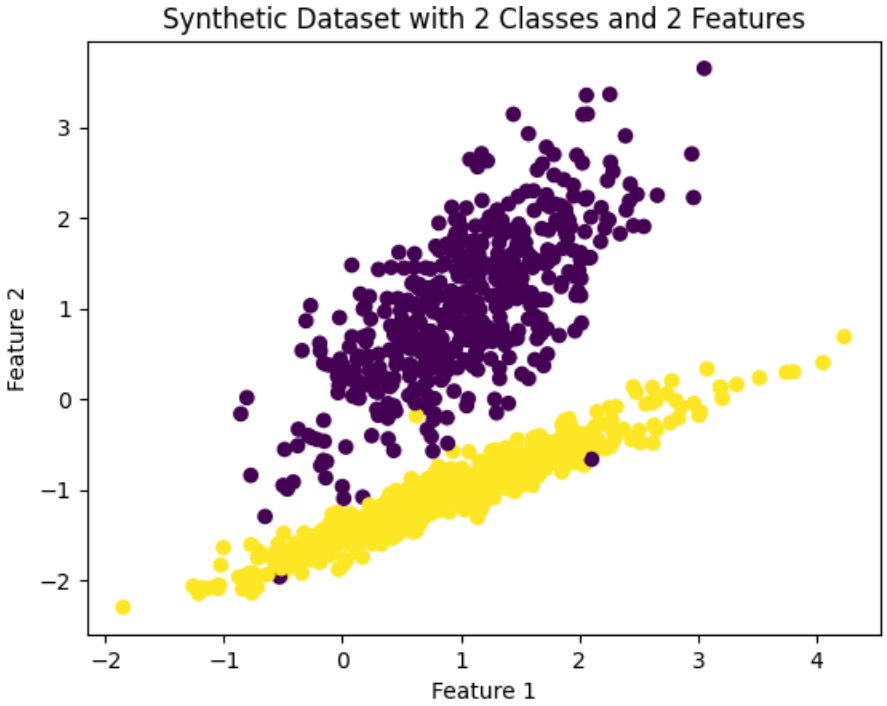
شیرین مهدی حاتم

پاییز ۱۴۰۲

سوال اول)

قسمت اول)

```
##1-1
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification

# Generating synthetic dataset
num_samples = 1000
num_features = 2
num_classes = 2
num_clusters_per_class = 1
num_redundant = 0
random_seed = 83

X, y = make_classification(
    n_samples=num_samples,
    n_features=num_features,
    n_classes=num_classes,
    n_clusters_per_class=num_clusters_per_class,
    n_redundant=num_redundant,
    random_state=random_seed
)

# Plotting the generated dataset
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Synthetic Dataset with {} Classes and {}
Features'.format(num_classes, num_features))
plt.show()
```



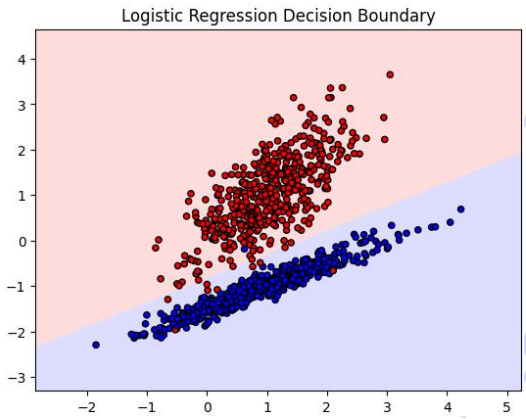
قسمت دوم)

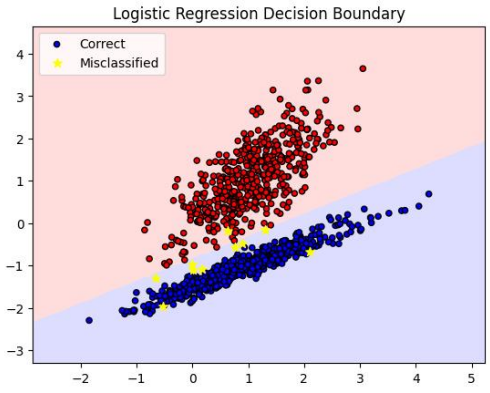
```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=83)
```

نتایج

```
Logistic Regression Accuracy: 99.00%
Random Forest Accuracy: 99.00%
```

قسمت سوم)

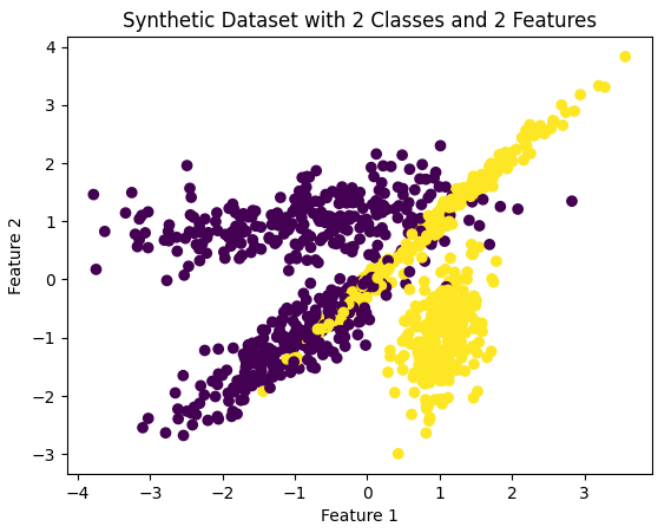




قسمت چهارم)

یکی از مهم ترین روش هایی که می تواند شرایط را برای ما در طبقه بندی کردن داده ها به مشکل بیاندازد این است که مقدار عددی پارامتر n_clusters_per_classرا افزایش دهیم. این مورد باعث می شود تا داده های دو کلاس بیشتر در هم قاطی شوند و اختلاط بیش از حد آن ها می تواند کار classificationرا سخت تر کند

```
##1-4-1

# Generating synthetic dataset
X, y = make_classification(n_samples=1000, n_features=2, n_classes=2,
n_clusters_per_class=2, n_redundant=0, random_state=83)

# Plotting the generated dataset
plt.scatter(X[:, 0], X[:, 1] ,c=y)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Synthetic Dataset with 2 Classes and 2 Features')
plt.show()
```



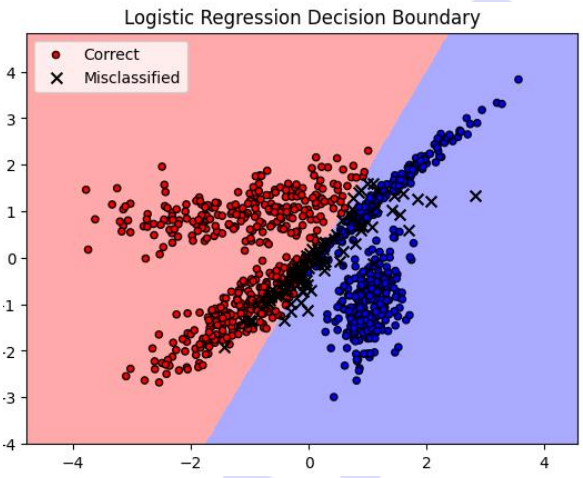
قسمت دوم قسمت چهار)

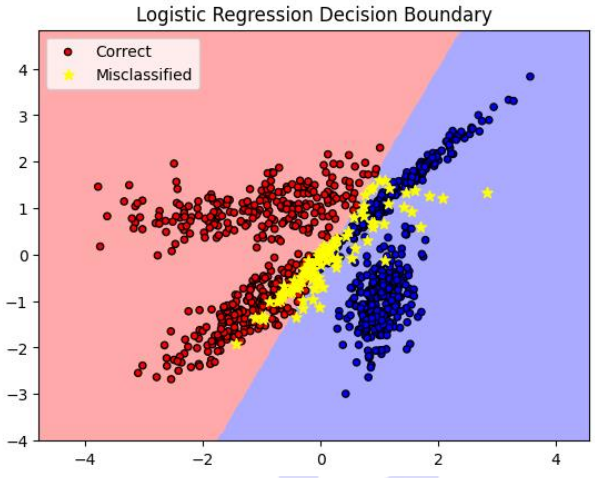
```
##1-4-2
# Generating synthetic dataset
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score

def generate_synthetic_dataset():
    # Generating synthetic dataset
    X, y = make_classification(n_samples=1000, n_features=2, n_classes=2,
n_clusters_per_class=2, n_redundant=0, random_state=83)
    return X, y

def split_dataset(X, y):
    # Splitting 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=83)
    return X_train, X_test, y_train, y_test

def train_logistic_regression(X_train, y_train):
    # Logistic Regression
    logreg_model = LogisticRegression(random_state=83)
    logreg_model.fit(X_train, y_train)
    return logreg_model

def train_random_forest(X_train, y_train):
    # Random Forest Classifier
    rf_model = RandomForestClassifier(n_estimators=100, random_state=83)
    rf_model.fit(X_train, y_train)
    return rf_model

def evaluate_model(model, X_test, y_test, model_name):
    # Evaluating the model
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print(f"{model_name} Accuracy: {accuracy}")

# Part 1 - the first question
X, y = generate_synthetic_dataset()
X_train, X_test, y_train, y_test = split_dataset(X, y)

# Train and evaluate Logistic Regression
logreg_model = train_logistic_regression(X_train, y_train)
evaluate_model(logreg_model, X_test, y_test, "Logistic Regression")

# Train and evaluate Random Forest
rf_model = train_random_forest(X_train, y_train)
evaluate_model(rf_model, X_test, y_test, "Random Forest")
```

```
Logistic Regression Accuracy: 0.925
Random Forest Accuracy: 0.915
```

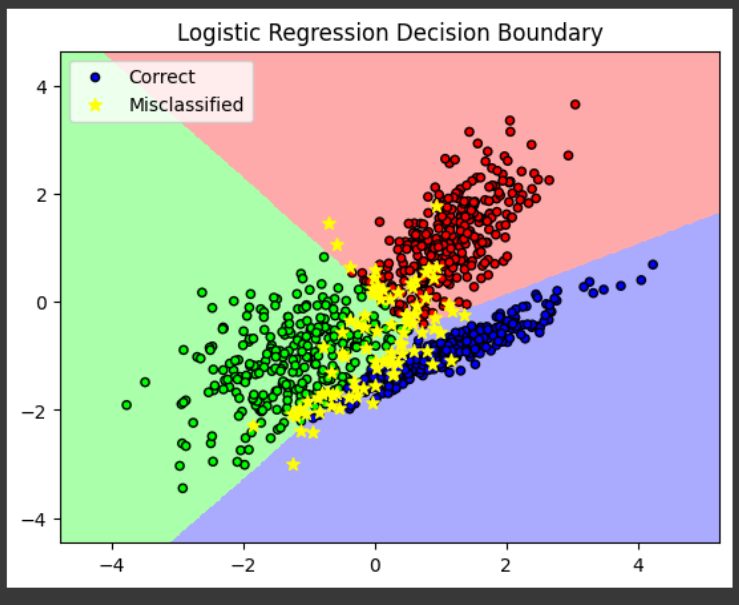
قسمت سوم قسمت چهارم)





قسمت پنج

```
Accuracy: 0.92
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93        71
           1       0.93      0.98      0.95        63
           2       0.92      0.83      0.87        66

    accuracy                           0.92       200
   macro avg       0.92      0.92      0.92       200
weighted avg       0.92      0.92      0.92       200
```



قسمت پنجم)

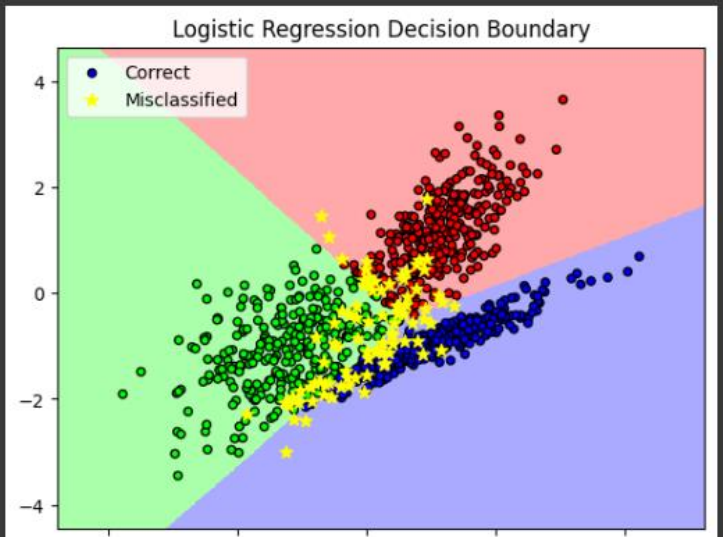
هنگام تولید مجموعه داده، باید تعداد کلاس ها n_classes را مشخص کنیم و مطمئن شویم که در این مورد روی 3تنظیم شده است. تعریف مدل:
اگر از طبقه بندی کنندهای استفاده می کنیم که از طبقه بندی چند کلاسه پشتیبانی می کند (به عنوان مثال، رگرسیون لجستیک و غیره)، ممکن است نیازی به ایجاد تغییرات مهم نداشته باشیم. با این حال، اگر از یک طبقه بندیکننده باینری استفاده می کنیم، باید به یک طبقه بندی کننده 'class='multinomial_با پارامتر LogisticRegression

آموزش: هنگام تقسیم مجموعه داده و آموزش مدل، اطمینان حاصل کنیم که مدل با طبقه بندی چند کلاسه سازگار است. اگر از scikit-learnاستفاده می کنیم، بسیاری از طبقه بندی کنندهها بهطور خودکار دسته بندی چند کلاسه را انجام میدهند.

ارزیابی: برای در نظر گرفتن کلاس جدید، معیارهای ارزیابی را به روز می کنیم. به عنوان مثال، دقت، برای هر سه کلاس محاسبه می شود. ادامه کار دقیقا به همان صورتی است که در قسمت های دیده شد. فقط در اینجا با سه کلاس سر و کار داریم و باید از الگوریتم هایی استفاده کنیم که توانایی طبقه بندی سه کلاس را داشته باشند.

```
def generate_synthetic_dataset(n_samples=1000, n_features=2, n_classes=3, n_clusters_per_class=1, n_redundant=0, random_state=83):
    # Generating synthetic dataset with specified parameters
    X, y = make_classification(n_samples=n_samples, n_features=n_features, n_classes=n_classes,
                               n_clusters_per_class=n_clusters_per_class, n_redundant=n_redundant,
                               random_state=random_state)
```

```
Accuracy: 0.92
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93        71
           1       0.93      0.98      0.95        63
           2       0.92      0.83      0.87        66

    accuracy                           0.92       200
   macro avg       0.92      0.92      0.92       200
weighted avg       0.92      0.92      0.92       200
```



سوال دوم)

قسمت اول:

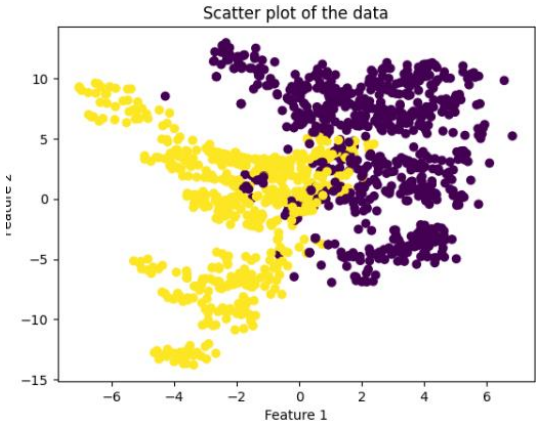
```
! pip install --upgrade --no-cache-dir gdown
! gdown 1EqYX552b90gRE6h19xOKLYZgNkPXaxS_
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.7.3)
Collecting gdown
  Downloading gdown-5.1.0-py3-none-any.whl (17 kB)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.11.17)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.7.3
    Uninstalling gdown-4.7.3:
      Successfully uninstalled gdown-4.7.3
Successfully installed gdown-5.1.0
Downloading...
From: https://drive.google.com/uc?id=1EqYX552b90gRE6h19xOKLYZgNkPXaxS_
To: /content/data_banknote_authentication.txt
100% 46.4k/46.4k [00:00<00:00, 56.2MB/s]
```

```
[2] import pandas as pd
    import numpy as np
    from sklearn.utils import shuffle

    file = pd.read_csv (r'/content/data_banknote_authentication.txt')
    #file = read_file.to_csv (r'/content/data_banknote_authentication.csv')
    headerlist = ['feature1' , 'feature2','feature3','feature4','feature5']
    file.to_csv("/content/data_banknote_authentication.csv" ,header = headerlist)
    df = pd.read_csv("/content/data_banknote_authentication.csv")
    df = shuffle(df)
```

قسمت دوم:

داده هارا مخلوط میکنیم و در ادامه با نسبت ۲۰ درصد به ۸۰ درصد فرایند اموزش را شروع میکنیم.



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

x_train.shape , x_test.shape , y_train.shape , y_test.shape
```

قسمت سوم)

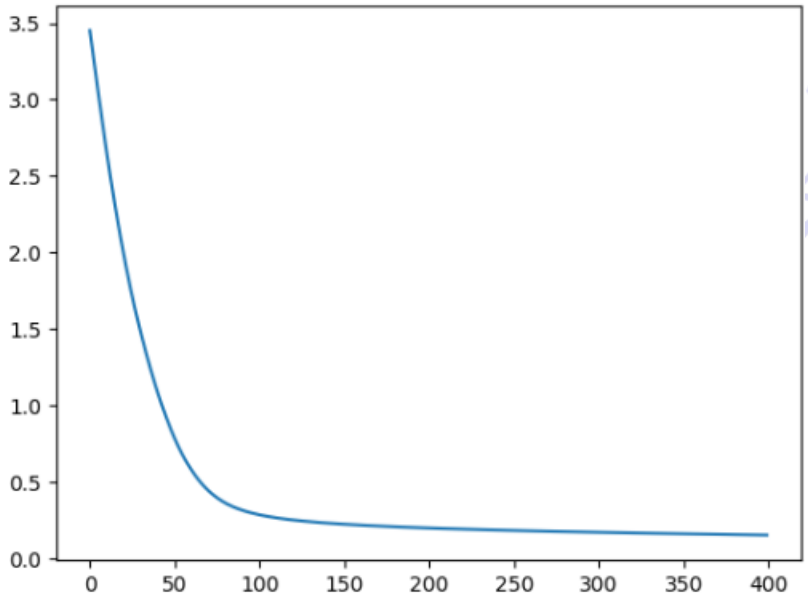
نوشتن توابع فعالسازی و اتلاف و ارزیابی بدون استفاده از کتابخانه آماده:

```
import numpy as np

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

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

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

def compute_gradient(x, y, y_hat):
    return (x.T @ (y_hat - y)) / len(y)

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

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

# Assuming x_train, y_train are defined elsewhere in your code
m = 4
w = np.random.randn(m + 1, 1)
print("Initial weights:", w.T[0])

eta = 0.01
n_epochs = 400
error_history = []

for epoch in range(n_epochs):
```

```
    y_hat = logistic_regression(x_train, w)

    error = binary_cross_entropy(y_train, y_hat)
    error_history.append(error)

    grads = compute_gradient(x_train, y_train, y_hat)
    w = gradient_descent(w, eta, grads)

    if (epoch + 1) % 100 == 0:
        accuracy = compute_accuracy(y_train, y_hat)
        print(f"Epoch = {epoch + 1}, Error = {error:.4f}, Accuracy =
{accuracy:.2%}")

print("Final weights:", w.T[0])
```

```
Initial weights: [ 0.04007028 -0.22481541 -1.5496606  -0.72844437  0.77128753]
Epoch = 100, Error = 0.2877, Accuracy = 103021.72%
Epoch = 200, Error = 0.1991, Accuracy = 104025.18%
Epoch = 300, Error = 0.1710, Accuracy = 104359.67%
Epoch = 400, Error = 0.1526, Accuracy = 104526.92%
Final weights: [-0.55191193 -0.25023734 -0.83923084 -1.23007239  0.81183893]
```

نمودار اتلاف:



آیا نمودار اتلاف با تابع اتلاف به تنهایی کافی است؟ خیر!

تحلیل نمودار تابع اتلاف: نمودار تابع اتلاف که در بالا رسم شد، نشان دهنده تغییرات مقدار تابع هزینه (تابع اتلاف) در طول فرآیند آموزش است. در زیر تحلیلی از این نمودار ارائه شده است:

- کاهش تدریجی تابع اتلاف: همانطور که انتظار میرود، با پیشرفت آموزش، تابع اتلاف به تدریج کاهش پیدا میکند. این نشان دهنده بهینه سازی و بهبود وزنها در جهتی که تابع هزینه کمینه شود، است.
- همگرایی: نمودار نشان میدهد که تابع اتلاف همگراست و به یک مقدار پایدار همگرا میشود. این نشاندهنده این است که فرآیند آموزش به یک وزن بهینه رسیده است.
- تغییرات تابع اتلاف در مراحل آخر آموزش: مراحل آخر آموزش، تغییرات در تابع اتلاف کمتر میشود و نمودار به یک خط میل میکند. این نشان دهنده استقرار مدل در یک وضعیت که دیگر بهبود قابل توجهی ندارد.

تحلیل نمودار تابع اتلاف مهم است تا مشاهده کنیم که آیا مدل به اندازه کافی بهینه شده است یا نیاز به افزایش تعداد ایپاک هاepochsداریم. از آنجایی که تابع اتلاف به سرعت کاهش پیدا کرده و سپس به یک مقدار ثابت رسیده، نشاندهنده برازش خوب مدل است.

نظر در مورد عملکرد مدل قبل از ارزیابی از روی تابع اتلاف: نمودار تابع اتلاف میتواند به ما اطلاعات مهمی درباره عملکرد مدل بدهد، اما این تا حدی است که به علتی به نام "برازش بر روی دادههای آموزش"، overfittingممکن است مدل به دادههای آموزش بسیار خوب برازش یافته باشد ولی بر روی دادههای تست یا دادههای جدید عملکرد مناسبی نداشته باشد. به عبارت دیگر، این ممکن است نشاندهنده این باشد که مدل به داده های آموزش "خاص "شده و توانمندی عمومی برای تفکیک موارد جدید را نداشته باشد.

به همین دلیل، نمودار تابع اتلاف به تنهایی کافی نیست و نیاز به ارزیابی روی دادههای جدید (تست) داریم. اگر تابع اتلاف بر روی داده های آموزش به خوبی کاهش پیدا کرده باشد اما عملکرد مدل بر روی داده های تست به اندازه مطلوب نباشد، ممکن است مدل ما با مشکلاتی مثل برازش بر روی دادههای آموزش روبرو شده باشد.

برای ارزیابی بهتر عملکرد مدل، میتوان از معیارهایی مانند صحت accuracy(بازخوانی) ،recall(دقت) precision(و اندازه F1 (F1-score (بر روی داده های تست استفاده کرد. این معیارها به ما اطلاعاتی درباره توانمندی مدل در تشخیص مثبتها و منفیها ارائه میدهند و میتوانند بهتر از تابع اتلاف به ما کمک کنند تا عملکرد مدل را به صورت جامعتر ارزیابی کنیم.

```
test_predictions = evaluate(X_test, trained_weights)
accuracy = accuracy_score(y_test, test_predictions)
precision = precision_score(y_test, test_predictions)
recall = recall_score(y_test, test_predictions)
f1 = f1_score(y_test, test_predictions)
```

```
Accuracy: 0.9709090909090909
Precision: 0.9734513274336283
Recall: 0.9565217391304348
F1 Score: 0.9649122807017544
```

قسمت چهارم)

به طور کلی روش های زیادی برای نرمال سازی داده ها داریم که در این قسمت می خواهیم به دو روش از آن اشاره کنیم:

نرمالسازی داده ها یک مرحله مهم در پردازش و تحلیل داده هاست که هدف آن ایجاد یک دسته بندی یکنواخت از داده ها برای اجتناب از مشکلات ناشی از مقیاس ها و واحدهای مختلف در داده ها است. دو روش متداول برای نرمالسازی داده ها عبارتند از:

Min-Max Scaling :در این روش، دادهها به گونهای تغییر میکنند که حداقل و حداکثر آنها به ترتیب به یک مقدار نگاشته میشوند.

Z-score Standardization :

در این روش، داده ها به گونه ای تغییر میکنند که میانگین آنها صفر و انحراف معیاری آنها یک شود.
در فرآیند نرمالسازی به تنهایی معمولاً استفاده نمیشود. بخش ارزیابی معمولاً برای "ارزیابی"اطلاعات بخش ارزیابی عملکرد مدل یا سیستم پس از اعمال تغییرات (مانند نرمالسازی) استفاده میشود. انتخاب یک روش نرمالسازی باید بر اساس نیازها و خصوصیات دادهها انجام شود. اگر توزیع دادهها نسبت به هم مهم است، ممکن است از Z-score Standardization استفاده کنید. اگر میخواهید دادهها را به یک بازه خاص نگاشت کنید، Min-Max Scaling مناسب تر است.
در مورد بخش "ارزیابی "در فرآیند عادی سازی، بستگی به زمینه دارد. اگر بخش ارزیابی حاوی اطلاعاتی در مورد دامنه و توزیع ویژگی ها باشد، می تواند در انتخاب روش نرمال سازی مناسب سودمند باشد. به عنوان مثال، اگر ویژگی ها دارای نقاط پرت باشند، عادی سازی امتیاز Z ممکن است قوی تر باشد. اگر ویژگیها محدوده مشخصی دارند، ممکن است مقیاسبندی Min-Max ترجیح داده شود. درک ویژگی های داده ها و انتخاب روش عادی سازی بر این اساس ضروری است.

```python
# Assuming df is your DataFrame and features are 'feature1', 'feature2', 'feature3', 'feature4'

# Calculate the maximum and minimum values for each feature
max_values = df[['feature1', 'feature2', 'feature3', 'feature4']].max()
min_values = df[['feature1', 'feature2', 'feature3', 'feature4']].min()

# Normalize each feature
for i in range(4):
    feature_name = f'feature{i+1}'
    df[feature_name] = (df[feature_name] - min_values[i]) / (max_values[i] - min_values[i])
    print(df[feature_name])
```

```
Name: feature1, Length: 1371, dtype: float64
704     0.399226
692     0.858925
297     0.546573
989     0.644520
491     0.920635
          ...
509     0.605189
124     0.784825
1268    0.257298
956     0.506542
977     0.526287
```

داده های را به دو دسته تست و آموزش تقسیم میکنیم:

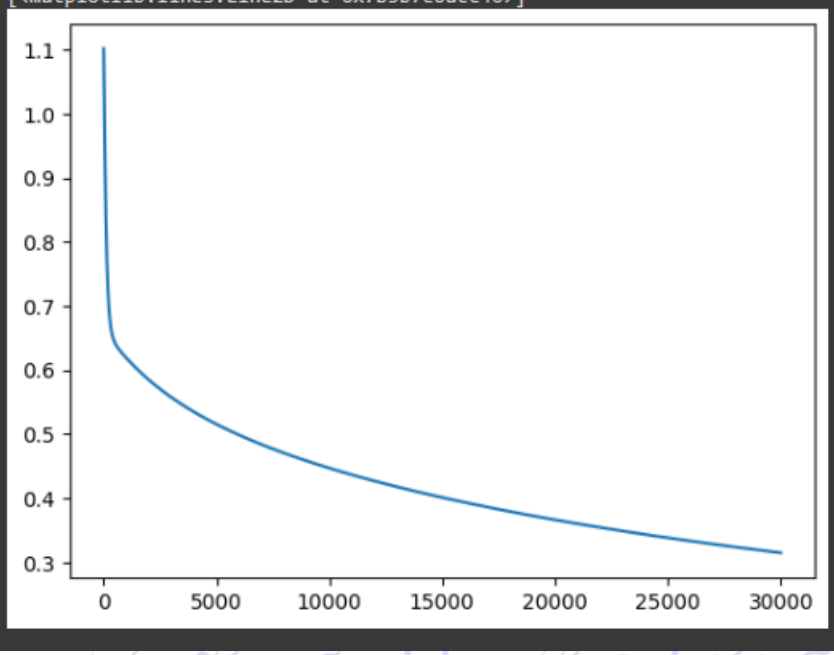
```
X = df[["feature1" ,"feature2" , "feature3" , "feature4"]].values

y = df[["feature5"]].values
X , y
```

به سراغ آموزش با استفاده از توابع نوشته شده بدون کتابخانه پایتون میرویم:

```
Test Accuracy: 91.2727%
```

```
plt.plot(error_history)
```

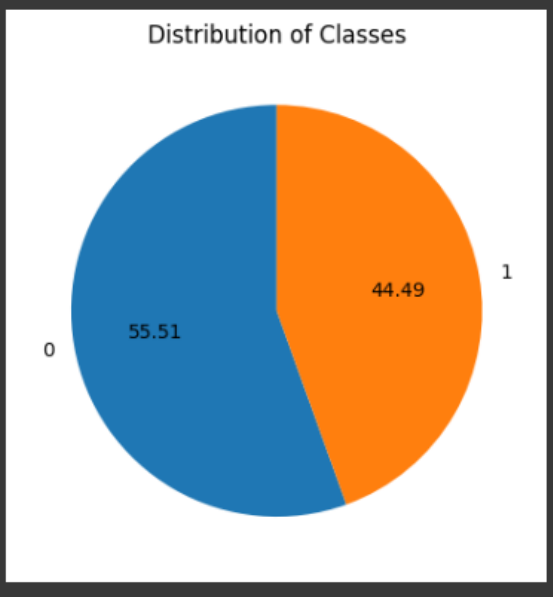
```
[<matplotlib.lines.Line2D at 0x7b3b7c6acc40>]
```



قسمت پنجم)

ابتدا باید وضعیت بالانس بودن در کلاس هارا مشاهده کنیم:

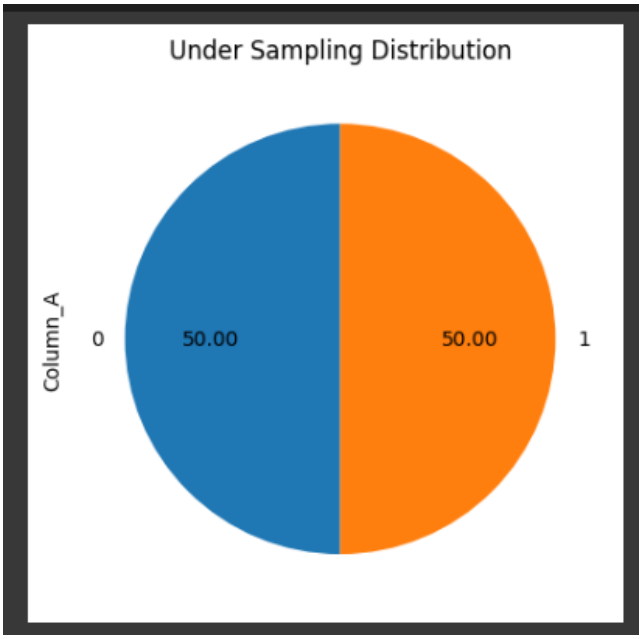
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Assuming 'y' is a pandas Series
new_y = pd.DataFrame(y, columns=['Column_A'])
value_counts = new_y['Column_A'].value_counts()

# Plotting a pie chart
plt.pie(value_counts, labels=value_counts.index, autopct="%.2f", startangle=90)
plt.title("Distribution of Classes")
plt.show()
```



مشاهده میکنیم کمی آنبالانسی در دو کلاس وجود دارد:

```
[13] ! pip install -U imbalanced-learn

     Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.10.1)
     Collecting imbalanced-learn
       Downloading imbalanced_learn-0.12.0-py3-none-any.whl (257 kB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 257.7/257.7 kB 2.4 MB/s eta 0:00:00
     Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.23.5)
     Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.11.4)
     Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.2.2)
     Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.3.2)
     Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.2.0)
     Installing collected packages: imbalanced-learn
       Attempting uninstall: imbalanced-learn
         Found existing installation: imbalanced-learn 0.10.1
         Uninstalling imbalanced-learn-0.10.1:
           Successfully uninstalled imbalanced-learn-0.10.1
     Successfully installed imbalanced-learn-0.12.0
```

```python
import pandas as pd
import matplotlib.pyplot as plt
from imblearn.under_sampling import RandomUnderSampler

# Assuming 'y' is a pandas Series
y = pd.DataFrame(y, columns=['Column_A'])

rus = RandomUnderSampler(sampling_strategy=1)
x_res_undersampling, y_res_undersampling = rus.fit_resample(X, y)

# Plotting a pie chart for the undersampled data
ax = y_res_undersampling['Column_A'].value_counts().plot.pie(autopct='%.2f', startangle=90)
ax.set_title("Under Sampling Distribution")
plt.show()
```



قسمت هفتم)

```
from sklearn.metrics import accuracy_score

# Assuming 'model' is a trained model and 'x_test', 'y_test' are your test
data
y_hat = model.predict(x_test)

# Using model.score() to calculate accuracy
accuracy_from_score = model.score(x_test, y_test)
print(f"Accuracy from model.score(): {accuracy_from_score:.4%}")

# Reshape y_hat if needed
y_hat = y_hat.reshape(len(y_test), 1)

# Checking shapes
print("Shapes - y_test:", y_test.shape, ", y_hat:", y_hat.shape)

# Using accuracy_score from sklearn.metrics
accuracy_from_metrics = accuracy_score(y_test, y_hat)
print(f"Accuracy from accuracy_score: {accuracy_from_metrics:.4%}")
```

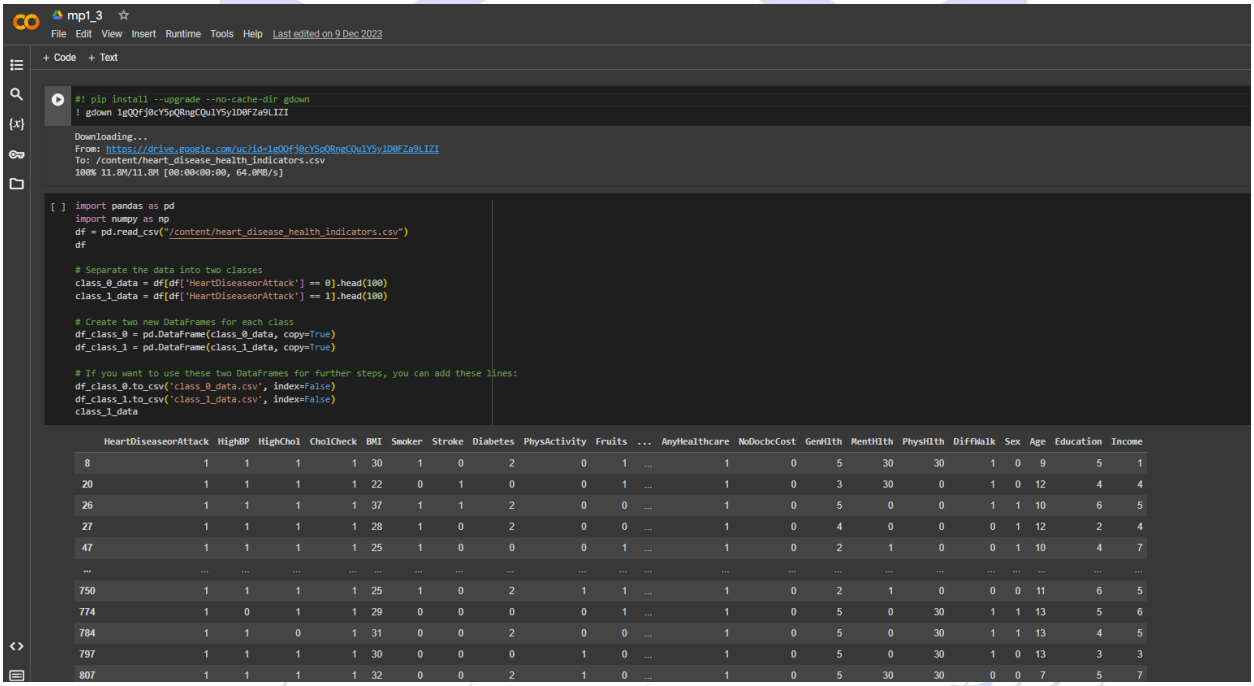
```
Accuracy from model.score(): 97.5410%
Shapes - y_test: (244, 1) , y_hat: (244, 1)
Accuracy from accuracy_score: 97.5410%
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
X = df[["feature1", "feature2", "feature3", "feature4"]].values
y = df["feature5"].values  # Assuming "feature5" is your target column

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=93)

# Initialize and train the Logistic Regression model
model = LogisticRegression(random_state=93, solver='sag', max_iter=200)
model.fit(x_train, y_train)

# Make predictions on the test set
y_hat = model.predict(x_test)
```

```
Accuracy from model.score(): 97.0909%
Accuracy from accuracy_score: 97.0909%
```

سوال سوم)

قسمت اول و دوم:



mp1_3

File Edit View Insert Runtime Tools Help Last edited on 9 Dec 2023

+ Code + Text

```
#! pip install --upgrade --no-cache-dir gdown
! gdown 1gQQfj0cY5pQRngCQu1Y5y1D0FZa9LIZI
```

```
Downloading...
From: https://drive.google.com/uc?id=1gQQfj0cY5pQRngCQu1Y5y1D0FZa9LIZI
To: /content/heart_disease_health_indicators.csv
100% 11.8M/11.8M [00:00<00:00, 64.0MB/s]
```

```
import pandas as pd
import numpy as np
df = pd.read_csv("/content/heart_disease_health_indicators.csv")
df

# Separate the data into two classes
class_0_data = df[df['HeartDiseaseorAttack'] == 0].head(100)
class_1_data = df[df['HeartDiseaseorAttack'] == 1].head(100)

# Create two new DataFrames for each class
df_class_0 = pd.DataFrame(class_0_data, copy=True)
df_class_1 = pd.DataFrame(class_1_data, copy=True)

# If you want to use these two DataFrames for further steps, you can add these lines:
df_class_0.to_csv('class_0_data.csv', index=False)
df_class_1.to_csv('class_1_data.csv', index=False)
class_1_data
```

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income
8	1	1	1	1	30	1	0	2	0	1	...	1	0	5	30	30	1	0	9	5	1
20	1	1	1	1	22	0	1	0	0	1	...	1	0	3	30	0	1	0	12	4	4
26	1	1	1	1	37	1	1	2	0	0	...	1	0	5	0	0	1	1	10	6	5
27	1	1	1	1	28	1	0	2	0	0	...	1	0	4	0	0	0	1	12	2	4
47	1	1	1	1	25	1	0	0	0	1	...	1	0	2	1	0	0	1	10	4	7
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
750	1	1	1	1	25	1	0	2	1	1	...	1	0	2	1	0	0	0	11	6	5
774	1	0	1	1	29	0	0	0	0	1	...	1	0	5	0	30	1	1	13	5	6
784	1	1	0	1	31	0	0	2	0	0	...	1	0	5	0	30	1	1	13	4	5
797	1	1	1	1	30	0	0	0	1	0	...	1	0	5	0	30	1	0	13	3	3
807	1	1	1	1	32	0	0	2	1	0	...	1	0	5	30	30	0	0	7	5	7

قسمت سوم:

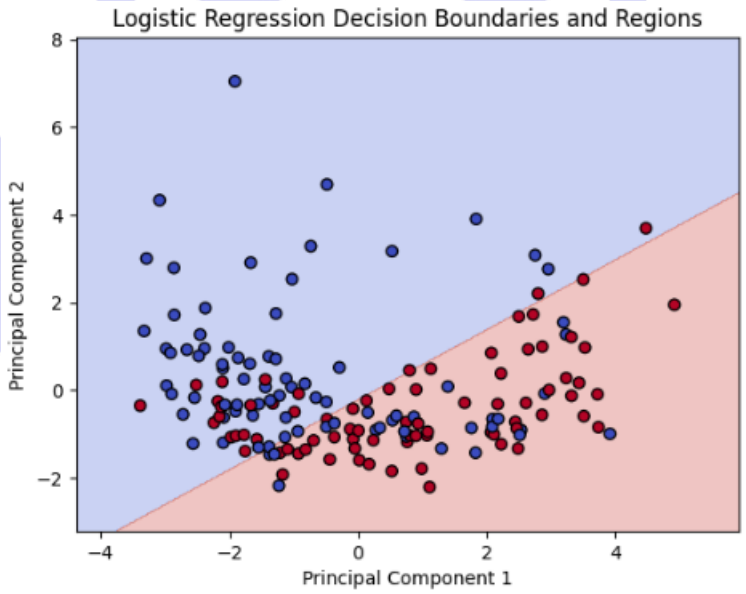
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.utils import shuffle

# Assuming you have df_class_0 and df_class_1 from the previous code
```

```
# Combine the two classes into a new DataFrame
combined_df = pd.concat([df_class_0, df_class_1], ignore_index=True)
combined_df = shuffle(combined_df)

# Separate features (X) and target (y)
X = combined_df.drop('HeartDiseaseorAttack', axis=1)  # Assuming
'HeartDiseaseorAttack' is the target column
y = combined_df['HeartDiseaseorAttack']

# 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)

# Train Logistic Regression model
logistic_model = LogisticRegression(solver='sag', max_iter=10000,
random_state=83)
logistic_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_logistic = logistic_model.predict(X_test)

# Evaluate the accuracy of the Logistic Regression model
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
print(f'Logistic Regression Accuracy: {accuracy_logistic:.2f}')

# Train Random Forest model
random_forest_model = RandomForestClassifier(n_estimators=10000,
random_state=83)
random_forest_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = random_forest_model.predict(X_test)

# Evaluate the accuracy of the Random Forest model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Random Forest Accuracy: {accuracy_rf:.2f}')

# Print shapes for X and predicted values
print("Shapes - X:", X.shape, ", y_pred_rf:", y_pred_rf.shape)
```

```
Logistic Regression Accuracy: 0.68
Random Forest Accuracy: 0.53
Shapes - X: (200, 21) , y_pred_rf: (40,)
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Assuming you have X_train and y_train from the previous code
# If you don't have a 2D dataset, you might need to use PCA or other
dimensionality reduction techniques

# Apply PCA to reduce the data to 2D for visualization
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)

# Train Logistic Regression model
logistic_model = LogisticRegression()
logistic_model.fit(X_train_pca, y_train)

# Create a meshgrid to plot the decision boundaries
h = 0.02
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,
h))

# Predict the labels for each point in the meshgrid
Z = logistic_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundaries and regions
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)
```

```
# Plot the examples
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train,
cmap=plt.cm.coolwarm, edgecolors='k', marker='o')
plt.title('Logistic Regression Decision Boundaries and Regions')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```




```
      # Combine the two classes into a new DataFrame
combined_df = pd.concat([df_class_0, df_class_1], ignore_index=True)

# Separate features (X) and target (y)
X = combined_df.drop('HeartDiseaseorAttack', axis=1)  # Assuming
'HeartDiseaseorAttack' is the target column
y = combined_df['HeartDiseaseorAttack']

# 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=83)

# Train Support Vector Machine (SVM) model
```

```
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_svm = svm_model.predict(X_test)

# Evaluate the accuracy of the SVM model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'SVM Accuracy: {accuracy_svm:.2f}')

# Train K-Nearest Neighbors (KNN) model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_knn = knn_model.predict(X_test)

# Evaluate the accuracy of the KNN model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'KNN Accuracy: {accuracy_knn:.2f}')
```

```
SVM Accuracy: 0.65
KNN Accuracy: 0.60
```

باز هم تقریبا به دقت مشابه در آموزش و ارزیابی رسیدیم.

قسمت چهارم:

بله. !

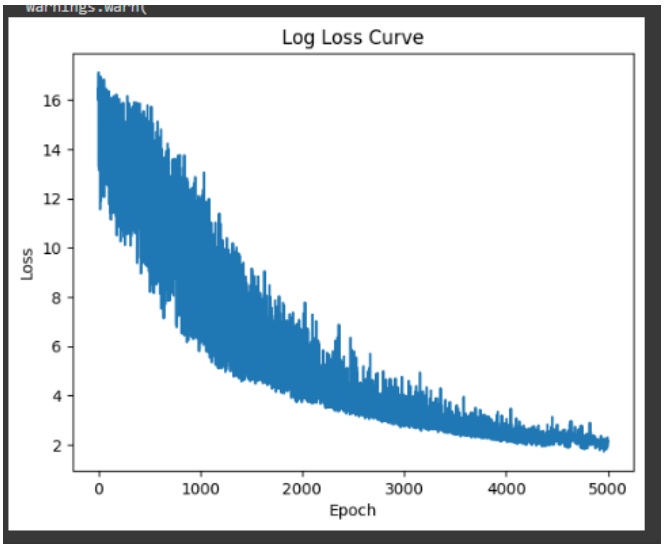
در واقع در این سوال می خواهیم با استفاده از کتابخانه های آماده در پایتون که در قسمت قبل داده ها را طبقه بندی کرده ایم در این حالت تابع اتلاف را بیابیم و رسم کنیم. طبق توضیحات گفته شده و خواسته شده در سوال باید مدل مورد استفاده در این سوال با استفاده از کتابخانه های آماده در پایتون زده شود و سپس به هر روشی که خواستیم و توانستیم تابع اتلاف را بیابیم و رسم کنیم. در این قسمت من سعی کردم از همان مدل هایی که در قسمت قبل استفاده کرده ایم در این قسمت نیز استفاده کنیم اما به طور مثال برخی از روش ها و مدل ها مانند LogisticRegressionاز دستورات مربوط به محاسبه تابع اتلاف به صورت عادی پیروی نمی کردند و باید به صورت نقطه ای و iterationآن ها را مشخص می کردیم. برای همین منظور در این قسمت من از یک مدل آماده در سایکیت لرن استفاده کرده ام که دستورات اماده آن به صورت زیر می باشد:

```
```

```python
import numpy as np

import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss

# Assuming you have X_train and y_train from the previous code
# If not, make sure to replace them with your actual training data

# 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=83)

model = SGDClassifier(loss='log', random_state=93)
losses = []
epochs = 5000

for epoch in range(epochs):
    model.partial_fit(X_train, y_train, [0, 1])
    loss = log_loss(y_train, model.predict_proba(X_train))
    losses.append(loss)

plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Log Loss Curve')
plt.show()
```

رندوم استیت دو رقم

اخرشماره دانشجویی



قسمت پنجم)

```
from sklearn.metrics import confusion_matrix, f1_score,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming you have y_test and y_pred_logistic from the previous code

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred_logistic)

# Calculate F1 score
F1 = f1_score(y_test, y_pred_logistic, average=None)

# Display F1 scores for each class
print("F1 Score for each class:", F1)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('Confusion Matrix')
plt.show()
```