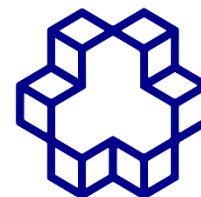


به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق



دانشگاه صنعتی خواجه نصیرالدین طوسی

مبانی سیستم های هوشمند

گزارش مینی پروژه شماره یک - بخش اول

[ایمان فکری اسکی]

[۹۹۲۹۰۸۳]

استاد : آقای دکتر مهدی علیاری

آذرماه ۱۴۰۲

فهرست مطالب

عنوان	شماره صفحه
بخش ۱: چکیده	۴
مقدمه	۵
سوال اول	۶
مورد اول	۶
مورد دوم	۷
مورد سوم	۱۰
مورد چهارم	۱۴
مورد پنجم	۱۹
سوال دوم	۲۳
مورد اول	۲۳
مورد دوم	۲۵
مورد سوم	۲۶
مورد چهارم	۳۰
مورد پنجم	۳۳
مورد ششم	۳۴
مورد هفتم	۳۷
سوال سوم	۳۸
مورد اول	۳۸
مورد دوم	۴۰

مورد سوم ۴۲

مورد چهارم ۴۶

مورد پنجم ۴۹

مراجع ۵۰



چکیده:

در این پروژه، سه سوال در مورد یادگیری ماشین مطرح شده است. سوال اول و سوم مربوط به طبقه بندی داده ها با استفاده از الگوریتم های مختلف هستند. سوال دوم مربوط به چالش های موجود در فرآیند طبقه بندی داده ها مانند عدم تعادل کلاس ها و نرمال سازی داده ها است.

در مجموع، نتایج این پروژه نشان می دهد که الگوریتم های یادگیری ماشین می توانند دقت بالایی در طبقه بندی داده ها داشته باشند. با این حال، انتخاب الگوریتم مناسب و تنظیم فرامترهای آن می تواند بر عملکرد مدل تأثیر بگذارد. همچنین، چالش های موجود در فرآیند طبقه بندی داده ها مانند عدم تعادل کلاس ها و نرمال سازی داده ها می توانند بر عملکرد مدل تأثیر منفی بگذارند.

مقدمه :

یادگیری ماشین یک زمینه هوش مصنوعی است که با توسعه الگوریتم‌هایی برای یادگیری از داده‌ها سروکار دارد. این الگوریتم‌ها می‌توانند برای حل طیف گسترده‌ای از مسائل، از جمله طبقه‌بندی، رگرسیون، تشخیص الگو و یادگیری تقویتی استفاده شوند.

طبقه‌بندی یکی از متداول‌ترین مسائل در یادگیری ماشین است. در این مسئله، هدف این است که داده‌های جدید را به یکی از چندین دسته از پیش تعریف‌شده تقسیم کنیم. به عنوان مثال، می‌توان از طبقه‌بندی برای شناسایی تصاویر، طبقه‌بندی متن یا تشخیص بیماری استفاده کرد.

در این پروژه، ما سه سوال در مورد طبقه‌بندی داده‌ها با استفاده از الگوریتم‌های یادگیری ماشین بررسی خواهیم کرد. سوال اول مربوط به انتخاب الگوریتم مناسب و تنظیم فرآیندهای آن است. سوال دوم مربوط به چالش عدم تعادل کلاس‌ها است. سوال سوم مربوط به استفاده از شاخص‌های ارزیابی مختلف است.

به طور کلی از هدف ما در انجام این پروژه می‌توان به موارد زیر اشاره کرد :

- بررسی عملکرد الگوریتم‌های مختلف طبقه‌بندی در شرایط مختلف
- بررسی تأثیر چالش عدم تعادل کلاس‌ها بر عملکرد طبقه‌بندان
- مقایسه عملکرد شاخص‌های ارزیابی مختلف

بخش ۱: سوالات تحلیلی

۱ سوال اول)

۱ با استفاده از `sklearn.datasets`، یک دیتاست با ۱۰۰۰ نمونه، ۲ کلاس و ۲ ویژگی تولید کنید.

در این قسمت به ما گفته شده است که با استفاده از ماژول `sklearn.datasets` یک دیتاست با ۱۰۰۰ نمونه دیتا و دارای ۲ کلاس و با ۲ ویژگی در پایتون تولید کنیم. پس باید در ابتدا کتابخانه های مورد نیاز خودمان را در فضای کولب `import` کنیم. بنابراین به صورت زیر عمل می کنیم :

```
# The first case is the first question :  
  
from sklearn.datasets import make_classification  
import matplotlib.pyplot as plt  
import numpy as np
```

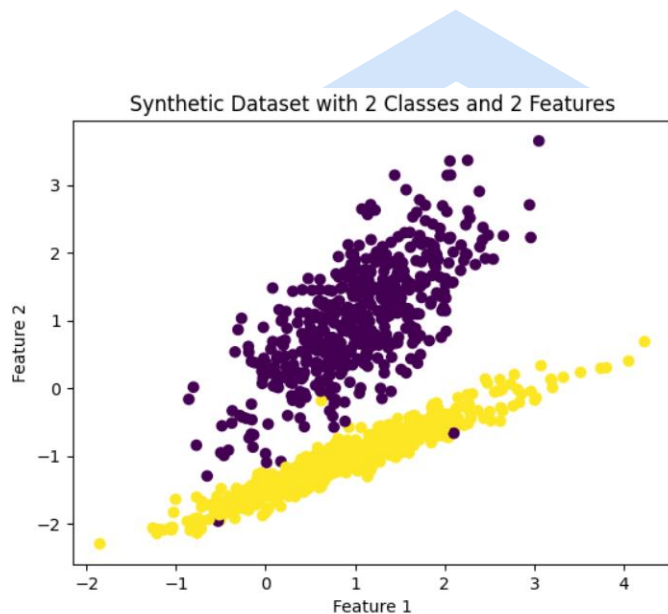
سپس با استفاده از دستور `make_classification` اطلاعات داده های مورد نظرم را وارد می کنیم تا با توجه به ویژگی های داده شده دیتاست ما را تولید کند :

```
# Generating synthetic dataset  
X, y = make_classification(n_samples=1000, n_features=2, n_classes=2,  
n_clusters_per_class=1, n_redundant=0, random_state=83)
```

در کد بالا دقت کنید که `n_sample` همان تعداد کل دیتاست های ما می باشد که طبق صورت سوال آن را برابر با ۱۰۰۰ قرار داده ایم. در ادامه `n_feature` نیز تعداد ویژگی های ما می باشد که آن را هم مطابق چیزی گفته شده است باید برابر با ۲ قرار دهیم. `n_classes` تعداد کلاس های ما می باشد که آن را نیز برابر با ۲ می گذاریم. `n_clusters_per_class` تعداد خوشه ها در هر کلاس را نشان می دهد که مقدار آن را برابر با ۱ قرار می دهیم. این بدین معنا می باشد که خوشه ها با یکدیگر همپوشانی ندارند. `n_redundant` تعداد ویژگی های اضافی را بیان می کند که با توجه به عدم خواسته سوال مقدار آن را برابر با صفر قرار می دهیم. و در نهایت نیز مقدار `random_state` را برابر با دو رقم آخر شماره دانشجویی قرار می دهیم. مقدار این مورد خیلی اهمیت ندارد اما ثابت بود آن باعث تکرار پذیری دیتاهای ما می شود.

در ادامه کار نیز دو ستون را که دارای دو ویژگی می باشند در پایتون با استفاده از دستور `scatter` رسم می کنیم.

```
# 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()
```



نکته : در کد بالا $c = y$ برای تمایز میان داده های دو کلاس نوشته شده است. در غیر این صورت داده های دو کلاس با یک رنگ نمایش داده می شد که قابل تشخیص از یکدیگر نبوده است. در این مورد اطلاعات کافی در مورد هر یک از کلاس ها به طور دقیق گفته نشده است و تنها تعداد آن ها را مشخص کرده ایم. حال در ادامه به جزئیات بیشتری در این زمینه می پردازیم. دیتاست ما به صورت روبرو خواهد بود :

۲ با استفاده از حداقل دو طبقه بند آماده پایتون و در نظر گرفتن فرآپارامترهای مناسب، دو کلاس موجود در دیتاست قسمت قبلی را از هم تفکیک کنید. ضمن توضیح روند انتخاب فرآپارامترها (مانند تعداد دوره آموزش و نرخ یادگیری)، نتیجه دقت آموزش و ارزیابی را نمایش دهید. برای بهبود نتیجه از چه تکنیک هایی استفاده کردید؟

در این سوال از ما خواسته شده است تا داده های تولید شده را در قسمت قبلی دسته بندی کنیم. این کار را با استفاده از دو طبقه بند آماده در پایتون `LogisticRegression` و `RandomForestClassifier` انجام می دهیم. قبل از تقسیم بندی داده ها می توانیم برای بررسی صحت داده ها آن ها را به دو قسمت `train` و `test` تقسیم بندی کنیم. در نهایت هم برای بررسی می زان `accuracy` داده ها از دو روش می توانیم از کتابخانه آماده در پایتون استفاده کنیم. برای همین موضوع کتابخانه های مورد نیاز خودمان را در پایتون `import` می کنیم :

```
# The second case is the first question :
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
```

برای همین موضوع مطابق بالا کتابخانه های `LogisticRegression` و `train_test_split` و `RandomForestClassifier` و `accuracy_score` را از `sklearn` در پایتون `import` می کنیم. از کتابخانه `train_test_split` برای تقسیم بندی داده ها به دو دسته `train` و `test` استفاده می کنیم. از دو کتابخانه `LogisticRegression` و `RandomForestClassifier` نیز برای تقسیم بندی خطی داده ها در پایتون استفاده می کنیم. این دو کتابخانه به صورت آماده در سایت وجود دارند و نیاز به محاسبات دستی مانند روش های گرادیان نزولی ندارد. در نهایت نیز پس از انتخاب پارامتر های خواسته شده، باید میزان `accuracy` داده ها را نیز مشخص کنیم. برای این موضوع نیز از دو روش می توانیم استفاده کنیم. یا می توانیم از فرمول و روش دستی که در جزوه گفته شده بود استفاده کنیم و هم می توانیم از کتابخانه آماده در `sklearn` استفاده کنیم. ما نیز از همان کتابخانه استفاده می کنیم :

```
# Generating synthetic dataset
X, y = make_classification(n_samples=1000, n_features=2, n_classes=2,
n_clusters_per_class=1, n_redundant=0, random_state=83)
```

مطابق با قسمت قبل می توانیم به همین صورت داده ها را تولید کنیم. تعداد داده ها را برابر با ۱۰۰۰ قرار می دهیم و تعداد کلاس ها و ویژگی های آن ها را نیز برابر با ۲ قرار می دهیم. تعداد خوشه های در هر کلاس را برابر با یک قرار می دهیم و از طرفی ویژگی های اضافی را نیز در نظر نمی گیریم. برای اینکه بعد از هر ران کردن دادهای ما تغییر نکند و ثابت باقی بماند و بتوانیم آن را تحلیل کنیم از پارامتر `random_state` استفاده می کنیم و آن را برابر با دو رقم آخر شماره دانشجویی خود قرار می دهیم.

```
# 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)
```

داده ها را به دو دسته `train` و `test` دسته بندی می کنیم. به طور دلخواه می توانیم مشخص کنیم که ماشین چه تعداد از داده های اصلی را برای `train` و چه تعداد از آن ها را برای `test` در نظر بگیرد. که در این قسمت

مطابق با فیلم حل تمرین ما ۲۰ درصد داده ها را برای test و ۸۰ درصد آن ها را مشخصاً برای train در نظر می گیریم. از دستور random_state نیز استفاده می کنیم تا shape داده ها تغییری نکند.

```
# Logistic Regression
logreg_model = LogisticRegression(random_state=83)
logreg_model.fit(X_train, y_train)
logreg_predictions = logreg_model.predict(X_test)
logreg_accuracy = accuracy_score(y_test, logreg_predictions)
```

بنابراین از کتابخانه LogisticRegression استفاده می کنیم و random_state را هم مطابق با قبل قرار می دهیم. مدل رگرسیون لجستیک را بر روی داده های آموزشی (x_train , y_train) آموزش می دهد. روش برازش پارامترهای مدل را برای به حداقل رساندن تفاوت بین برچسب های پیش بینی شده و برچسب های واقعی تنظیم می کند. از مدل رگرسیون لجستیک آموزش دیده برای پیش بینی داده های آزمون (x_train) استفاده می کند. روش پیش بینی برچسب های پیش بینی شده را بر اساس ویژگی های ورودی تولید می کند. دقت مدل رگرسیون لجستیک را با مقایسه پیش بینی های آن (logreg_predictions) با برچسب های واقعی مجموعه آزمایشی (y_train) محاسبه می کند. برای این منظور از تابع accuracy_score از scikit-learn استفاده می شود. حال به سراغ روش دوم می رویم :

```
# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=83)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
```

خط اول یک نمونه از کلاس RandomForestClassifier با ۱۰۰ درخت (n_estimators=100) ایجاد می کند. همانند رگرسیون لجستیک، random_state=42 برای تکرارپذیری تنظیم شده است. خط بعدی طبقه بندی کننده تصادفی را روی داده های آموزشی (x_train , y_train) آموزش می دهد. خط بعدی از طبقه بندی کننده تصادفی آموزش دیده برای پیش بینی داده های آزمون (x_train) استفاده می کند. در نهایت، خط آخر دقت طبقه بندی کننده تصادفی را با مقایسه پیش بینی های آن (rf_predictions) با برچسب های واقعی مجموعه آزمایشی (y_test) محاسبه می کند. برای این منظور مجدداً از تابع accuracy_score استفاده می شود. در نهایت نیز باید دقت محاسبه شده در دو روش را نمایش دهیم برای همین منظور به صورت زیر عمل می کنیم :

```
# Display the results
print("Logistic Regression Accuracy:", logreg_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
```

دقت آموزش و ارزیابی در نهایت به صورت زیر خواهد بود :

```
Logistic Regression Accuracy: 0.99
Random Forest Accuracy: 0.99
```

برای بهبود نتایج، می توانیم تکنیک های زیر را در نظر بگیریم:

Hyperparameter Tuning: با مقادیر مختلف هایپر پارامترها آزمایش می کنیم تا ترکیب بهینه را پیدا کنیم. می توانیم از تکنیک هایی مانند جستجوی شبکه ای یا جستجوی تصادفی استفاده کنیم.

Feature Engineering: اگر مجموعه داده اجازه می دهد، سعی می کنیم ویژگی های جدید ایجاد کنیم یا ویژگی های موجود را تغییر دهیم تا اطلاعات بیشتری به مدل ارائه دهیم.

Ensemble Methods: چندین مدل را برای ایجاد یک مجموعه ترکیب می کنیم. این اغلب می تواند عملکرد را با کاهش بیش از حد برازش یا گرفتن الگوهای مختلف در داده ها بهبود بخشد.

Cross-Validation: از اعتبارسنجی متقاطع برای به دست آوردن تخمین بهتری از عملکرد مدل استفاده می کنیم. این کمک می کند تا اطمینان حاصل شود که مدل به خوبی به داده های دیده نشده تعمیم می یابد.

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

```
# Part 3 - the first question :

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

در ابتدا تمامی کتابخانه هایی که نیاز داریم را در پایتون و در کولب import می کنیم. در ادامه باید خط تقسیم کننده داده ها به دو کلاس را با توجه به متدی که در مورد قبل train کرده بودیم رسم کنیم.

بنابراین به صورت زیر عمل می کنیم :

```
# Function to plot decision boundaries and areas
def plot_decision_boundary(model, X, y, title):
```

این خط تابعی به نام `plot_decision_boundary` را تعریف می‌کند که چهار پارامتر را می‌گیرد: مدل (طبقه‌بندی‌کننده آموزش‌دیده)، (ویژگی‌های ورودی)، (برچسب‌های مربوطه)، و عنوان (عنوان نمودار).

```
h = .02 # Step size in the mesh
```

این خط اندازه گام را برای شبکه مشبک تعیین می‌کند که برای ترسیم مرزهای تصمیم‌گیری و مناطق استفاده می‌شود. مقادیر کوچکتر `h` منجر به نمودار دقیق‌تری می‌شود، اما ممکن است محاسبه آن بیشتر طول بکشد.

```
# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#0000FF'])
```

این خطوط با استفاده از `ListedColormap` دو نقشه رنگی را تعریف می‌کنند. `cmap_light` برای مناطق تصمیم‌گیری استفاده می‌شود، در حالی که `cmap_bold` برای ترسیم نقاط آموزشی استفاده می‌شود.

```
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

این خطوط یک شبکه مشبک `(xx, yy)` ایجاد می‌کنند که محدوده ویژگی‌های ورودی را پوشش می‌دهد. پیش‌بینی‌های مدل برای هر نقطه در شبکه مشبک محاسبه شده و برای مطابقت با ابعاد شبکه تغییر شکل می‌دهند. این به ما این امکان را می‌دهد که مرزها و مناطق تصمیم‌گیری را تجسم کنیم. (مطابق با چیزی که در فیلم آموزشی گفته شده بود)

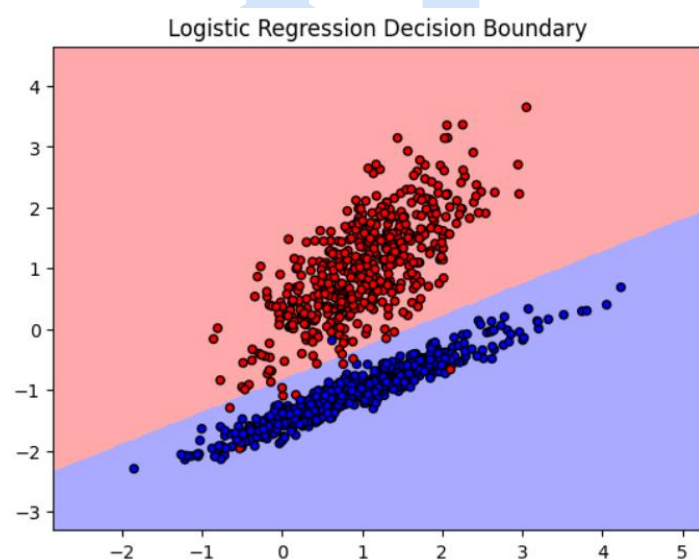
```
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

کد بالا یک شکل ایجاد می کند و از `pcolormesh` برای پر کردن مناطق تصمیم با رنگ ها مطابق با `cmap_light` استفاده می کند.

```
# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k',
s=20)
plt.title(title)

plt.show()

# Plot decision boundary for Logistic Regression
plot_decision_boundary(logreg_model, X, y, 'Logistic Regression Decision
Boundary')
```



در ادامه نیز باید شکل طبقه بندی شده را ترسیم کنیم که به صورت بالا عمل می کنیم. تمامی این مراحل را نیز می توانستیم با استفاده از یک کتابخانه آماده ای در `sklearn` ایجاد کنیم و خیلی سریع تر به جواب برسیم. اما برای نمایش متفاوتی نسبت به سایر این کار را انجام داده ایم. شکل نهایی به صورت روبرو خواهد بود :

همان طور که خواسته شده بود داده های هر کلاس را به صورت متفاوت نمایش داده ایم و با

استفاده از مدلی که در مورد قبل `train` کرده بودیم (`logreg_model`) خط مقسم کننده را ترسیم کرده ایم. این در صورتی است که برخی از داده های هر کلاس در کلاس دیگری قرار دارد و با نمایش متفاوتی ترسیم شده است.

یا اگر منظور سوال این است که به طور کلی داده هایی که به اشتباه طبقه بندی شده اند به صورت کاملاً متفاوت با سایر داده ها باشند ، می توانیم از کد پایین استفاده کنیم و خروجی را نمایش دهیم :

```

# Part 3 - the first question :

def plot_decision_boundary(model, X, y, title):
    h = .02 # Step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

    # Plot the decision boundary
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot the training points
    correct_predictions = model.predict(X) == y
    incorrect_predictions = ~correct_predictions

    plt.scatter(X[correct_predictions, 0], X[correct_predictions, 1],
c=y[correct_predictions], cmap=cmap_bold, marker='o', edgecolor='k', s=20,
label='Correct')
    plt.scatter(X[incorrect_predictions, 0], X[incorrect_predictions, 1],
marker='*', color='yellow', s=50, label='Misclassified')

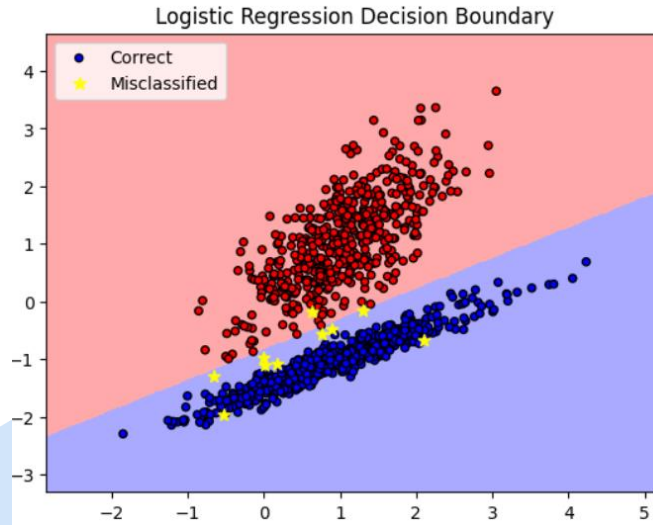
    plt.title(title)
    plt.legend()
    plt.show()

# Plot decision boundary for Logistic Regression
plot_decision_boundary(logreg_model, X, y, 'Logistic Regression Decision
Boundary')

```

آرایه های `correct_predictions` و `incorrect_predictions` برای جداسازی نقاط صحیح و اشتباه طبقه بندی شده استفاده می شوند.

`marker='o'` برای نقاط صحیح استفاده می شود و `marker='x'` برای نقاط اشتباه طبقه بندی شده در تابع `scatter` استفاده می شود. در نهایت شکل نهایی به صورت زیر خواهد بود :



۴ از چه طریقی می توان دیتاست تولیدشده در قسمت «۱» را چالش برانگیزتر و سخت تر کرد؟ این کار را انجام داده و قسمت های «۲» و «۳» را برای این داده های جدید تکرار و نتایج را مقایسه کنید.

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_class` را 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 `classification` را سخت تر کند. برای این که این موضوع را بهتر متوجه شویم می توانیم آن را به صورت زیر نمایش دهیم :

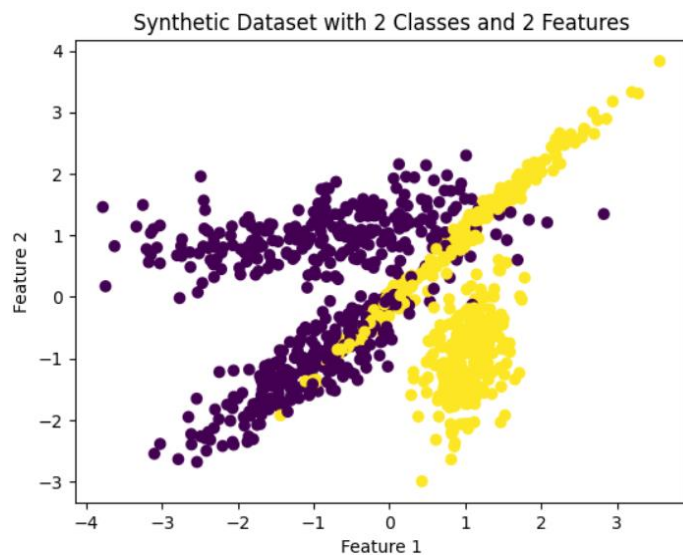
```
# Part 4 - the first question :

# Part 1 - the first question :

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

# 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()
```



همان طور که مشخص است در این مورد مقدار `n_clusters_per_class` را بجای یک برابر با دو قرار داده ایم و شکل نهایی با وجود تغییر کوچکی که انجام داده ایم کاملاً متفاوت شده است :

حال می خواهیم تمامی مراحل ۲ و ۳ را با این نوع داده ها و شرایط انجام دهیم پس به صورت زیر عمل می کنیم :

بررسی مورد ۲ در دیتا های جدید :

```
# Part 4 - the first question :

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

# 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)

# 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)

# Logistic Regression
logreg_model = LogisticRegression(random_state=83)
logreg_model.fit(X_train, y_train)
logreg_predictions = logreg_model.predict(X_test)
logreg_accuracy = accuracy_score(y_test, logreg_predictions)

# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=83)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
```

```
# Display the results
print("Logistic Regression Accuracy:", logreg_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
```

دقیقا مانند قبل تکرار می کنیم. (بنابراین دیگر نیاز به توضیح نیست.) در این حالت همان طور که انتظار داریم

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

دقت آموزش و ارزیابی کمتر از حالت قبل شده است :

بررسی مورد ۳ در دیتاهای جدید :

```
# Part 4 - the first question :

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

# 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)

# 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)

# Training a logistic regression model
logreg_model = LogisticRegression(random_state=83)
logreg_model.fit(X_train, y_train)

# Making predictions on the test set
logreg_predictions = logreg_model.predict(X_test)

# Function to plot decision boundaries and areas
def plot_decision_boundary(model, X, y, title):
    h = .02 # Step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
```



```

cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

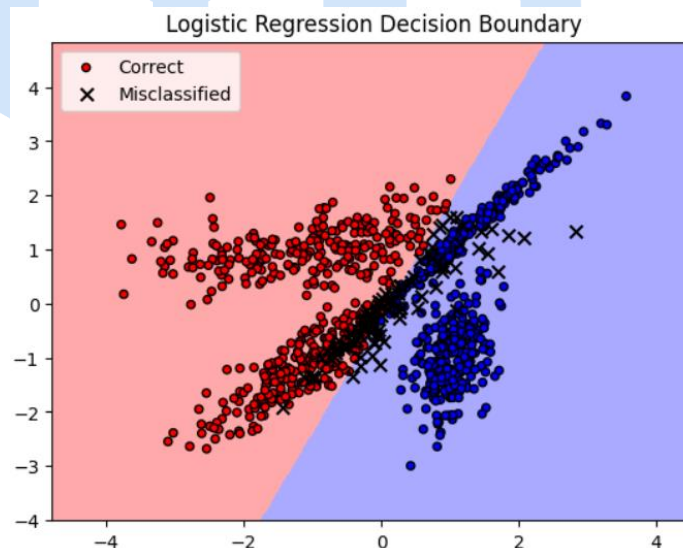
# Plot the training points
correct_predictions = model.predict(X) == y
incorrect_predictions = ~correct_predictions

plt.scatter(X[correct_predictions, 0], X[correct_predictions, 1],
c=y[correct_predictions], cmap=cmap_bold, marker='o', edgecolor='k', s=20,
label='Correct')
plt.scatter(X[incorrect_predictions, 0], X[incorrect_predictions, 1],
marker='x', color='black', s=50, label='Misclassified')

plt.title(title)
plt.legend()
plt.show()

# Plot decision boundary for Logistic Regression
plot_decision_boundary(logreg_model, X, y, 'Logistic Regression Decision
Boundary')

```



```

# Part 4 - the first question :

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

# 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)

# 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)

# Training a logistic regression model
logreg_model = LogisticRegression(random_state=83)
logreg_model.fit(X_train, y_train)

# Making predictions on the test set
logreg_predictions = logreg_model.predict(X_test)

# Function to plot decision boundaries and areas
def plot_decision_boundary(model, X, y, title):
    h = .02 # Step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

    # Plot the decision boundary
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

```

```

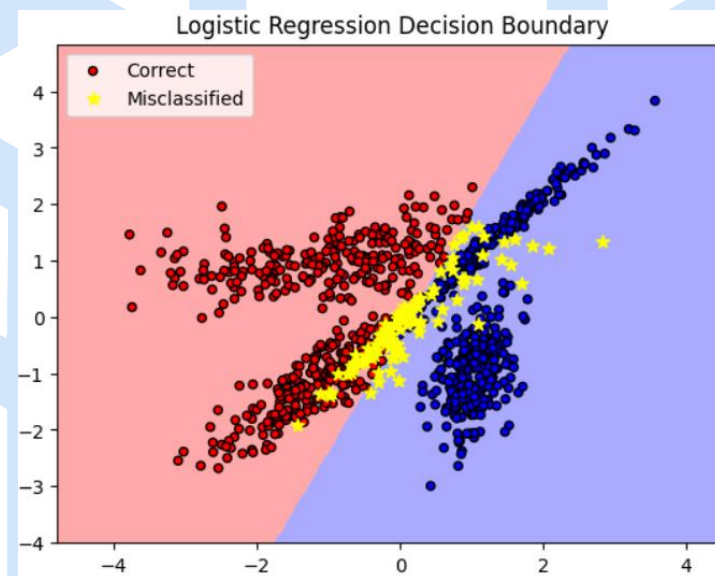
# Plot the training points
correct_predictions = model.predict(X) == y
incorrect_predictions = ~correct_predictions

plt.scatter(X[correct_predictions, 0], X[correct_predictions, 1],
c=y[correct_predictions], cmap=cmap_bold, marker='o', edgecolor='k', s=20,
label='Correct')
plt.scatter(X[incorrect_predictions, 0], X[incorrect_predictions, 1],
marker='*', color='yellow', s=50, label='Misclassified')

plt.title(title)
plt.legend()
plt.show()

# Plot decision boundary for Logistic Regression
plot_decision_boundary(logreg_model, X, y, 'Logistic Regression Decision
Boundary')

```



۵ اگر یک کلاس به داده های تولیدشده در قسمت «۱» اضافه شود، در کدام قسمت ها از بلوک دیاگرام آموزش و ارزیابی تغییراتی ایجاد می شود؟ در مورد این تغییرات توضیح دهید. آیا می توانید در این حالت پیاده سازی را به راحتی و با استفاده از کتابخانه ها و کدهای آماده پایتونی انجام دهید؟ پیاده سازی کنید.

همان طور که می دانیم الگوریتم آموزش طراحی مشخصی دارد و می توانیم هر قسمت آن را تغییر دهیم و تغییر در هر قسمت آن می تواند تغییراتی در ارزیابی سیستم ما ایجاد کند. بلوک دیاگرام آن به صورت زیر خواهد بود :

تولید داده:

هنگام تولید مجموعه داده، باید تعداد کلاس ها ($n_classes$) را مشخص کنیم و مطمئن شویم که در این مورد روی ۳ تنظیم شده است.

تعریف مدل:

اگر از طبقه‌بندی‌کننده‌ای استفاده می‌کنیم که از طبقه‌بندی چند کلاسه پشتیبانی می‌کند (به عنوان مثال، رگرسیون لجستیک و غیره)، ممکن است نیازی به ایجاد تغییرات مهم نداشته باشیم. با این حال، اگر از یک طبقه‌بندی‌کننده باینری استفاده می‌کنیم، باید به یک طبقه‌بندی‌کننده

چند کلاسه مانند LogisticRegression با پارامتر `multi_class='multinomial'` برویم.

آموزش:

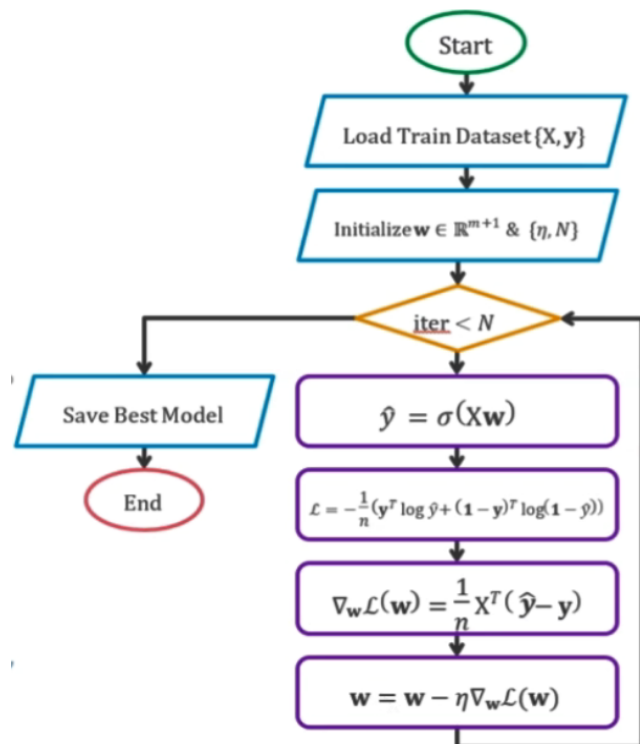
هنگام تقسیم مجموعه داده و آموزش مدل، اطمینان حاصل کنیم که مدل با طبقه‌بندی چند کلاسه سازگار است. اگر از `scikit-learn` استفاده می‌کنیم، بسیاری از طبقه‌بندی‌کننده‌ها به‌طور خودکار دسته‌بندی چند کلاسه را انجام می‌دهند.

ارزیابی:

برای در نظر گرفتن کلاس جدید، معیارهای ارزیابی را به روز می‌کنیم. به عنوان مثال، دقت، برای هر سه کلاس محاسبه می‌شود.

ادامه کار دقیقاً به همان صورتی است که در قسمت‌های دیده شد. فقط در اینجا با سه کلاس سر و کار داریم و باید از الگوریتم‌هایی استفاده کنیم که توانایی طبقه‌بندی سه کلاس را داشته باشند.

در اینجا یک نمونه پیاده‌سازی با استفاده از کتابخانه `scikit-learn` آورده شده است. در این مورد، من از LogisticRegression با پارامتر `multi_class='multinomial'` استفاده می‌کنم:



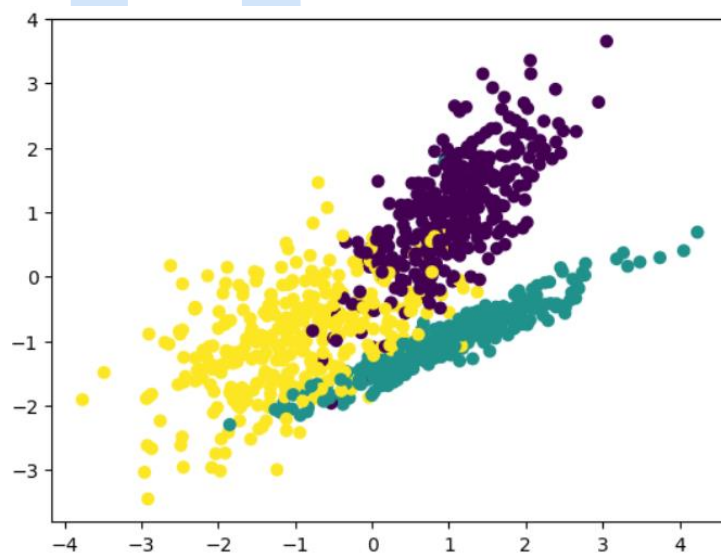
در ابتدا باید دیتا ها را با همان ساختاری که در قسمت یک خواسته شده بود اما این بار با سه کلاس تولید کنیم.
پس به صورت زیر پیاده سازی می کنیم :

```
# Part 5 - the first question :

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

# Generating synthetic dataset with 3 classes
X, y = make_classification(n_samples=1000, n_features=2, n_classes=3,
n_clusters_per_class=1, n_redundant=0, random_state=83)
plt.scatter(X[:, 0], X[:, 1], c=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)
```



همان طور که مشخص است دیتا ها همان دیتاهای مورد یک می باشند با این تفاوت که دارای سه کلاس هستند. حال باید برای طبقه بندی این دیتا ها در پایتون از کتابخانه های آماده در پایتون استفاده کنیم. باید این کتابخانه هایی که انتخاب می کنیم دارای قابلیت تبدیل به کلاس های بیشتری نسبت به حالت باینری را داشته باشند. در غیر این صورت باید از دستور بالا برای این حالت استفاده کنیم. بنابراین در این قسمت از همان کتابخانه ای که در قسمت قبل استفاده کردیم استفاده می کنیم و به صورت زیر پیش میرویم :

```
# Training a logistic regression model for multi-class classification
logreg_model = LogisticRegression(multi_class='multinomial',
solver='lbfgs', random_state=83)
logreg_model.fit(X_train, y_train)

# Making predictions on the test set
logreg_predictions = logreg_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, logreg_predictions)
print("Accuracy:", accuracy)
```

در این حالت نیز دقت جدید را محاسبه می کنیم و با حالت های قبل مقایسه می کنیم :

Accuracy: 0.92

که دقت مطابق با بالا از حالت های پیش خیلی کمتر شده است.

حال می خواهیم مرز تصمیم گیری را نیز مشخص کنیم. برای این منظور می توانیم از دستوراتی که از قبل استفاده کرده بودیم استفاده کنیم :

```
# Function to plot decision boundaries and areas
def plot_decision_boundary(model, X, y, title):
    h = .02 # Step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF', '#AAFFAA'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF', '#00FF00'])

    # Plot the decision boundary
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

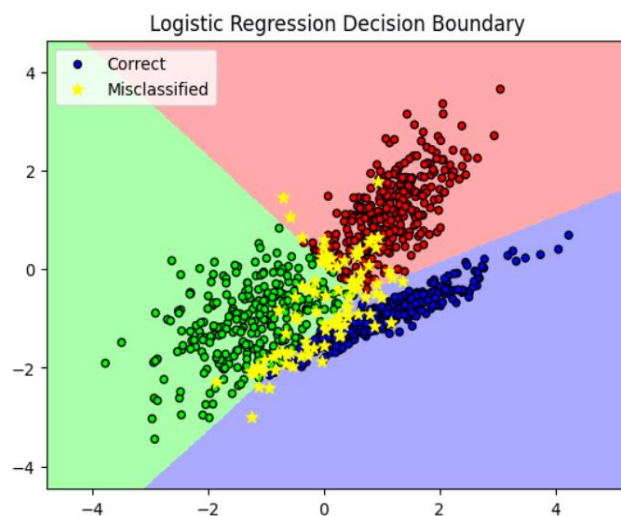
    # Plot the training points
    correct_predictions = model.predict(X) == y
    incorrect_predictions = ~correct_predictions
```

```
plt.scatter(X[correct_predictions, 0], X[correct_predictions, 1],
c=y[correct_predictions], cmap=cmap_bold, marker='o', edgecolor='k', s=20,
label='Correct')

plt.scatter(X[incorrect_predictions, 0], X[incorrect_predictions, 1],
marker='*', color='yellow', s=50, label='Misclassified')

plt.title(title)
plt.legend()
plt.show()

# Plot decision boundary for Logistic Regression
plot_decision_boundary(logreg_model, X, y, 'Logistic Regression Decision
Boundary')
```



در این حالت دیگر نمی توانیم سه کلاس را با هر تعداد دیتایی از یکدیگر جدا کنیم. بنابراین این کار را کتابخانه های آماده پایتون به صورت خودکار انجام می دهند. شکل طبقه بندی شده به صورت روبرو نمایش داده می شود و مانند قبل باز داده هایی که از کلاس های دیگر در طبقه های دیگری به اشتباه قرار گرفته اند و دارای خطا می باشند را با علامت متفاوت نمایش می دهیم. در شکل هر رنگ نشان دهنده یک کلاس در دیتاست بالا می باشد :

۲ سوال دوم)

۱ با مراجعه به این پیوند با یک دیتاست مربوط به حوزه «بانکی» آشنا شوید و ضمن توضیح کوتاه اهداف و ویژگی هایش، فایل آن را دانلود کرده و پس از بارگذاری در گوگل درایو خود، آن را با دستور `gdown` در محیط گوگل کولب قرار دهید. اگر تغییر فرمتی برای فایل این دیتاست نیاز می بینید، این کار را با دستورهای پایتونی انجام دهید.

داده ها از تصاویری که از نمونه های واقعی و جعلی شبیه اسکناس گرفته شده بودند استخراج شد. برای دیجیتالی کردن، از یک دوربین صنعتی که معمولاً برای بازرسی چاپ استفاده می شود استفاده می شود. تصاویر نهایی دارای ۴۰۰ در ۴۰۰ پیکسل هستند. با توجه به لنز شی و فاصله تا جسم مورد بررسی، تصاویری در مقیاس خاکستری با وضوح حدود ۶۶۰ نقطه در اینچ به دست آمد. ابزار تبدیل مویک برای استخراج ویژگی ها از تصاویر استفاده شد.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Computer Science	Classification
Feature Type	# Instances	# Features
Real	1372	-

حال می خواهیم پس از دانلود فایل دیتاست آن را در محیط کولب import کنیم.

```

Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.6.6)
collecting gdown
  Downloading gdown-4.7.1-py3-none-any.whl (15 kB)
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: six in /usr/local/lib/python3.10/dist-packages (from gdown) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: BeautifulSoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
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) (3.4)
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) (2023.7.22)
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 4.6.6
    Uninstalling gdown-4.6.6:
      Successfully uninstalled gdown-4.6.6
  Successfully installed gdown-4.7.1
Downloading...
From: https://drive.google.com/uc?id=1eqYX552b0qR6Eh19XOKLYZghKPKXa5...
To: /content/data_banknote_authentication.txt
100% 46.4k/46.4k [00:00<00:00, 87.7MB/s]
  
```

تا الان توانستیم داده ها را در فضای کولب آپلود کنیم. حال می خواهیم تغییر فرمتی در آن ایجاد کنیم :

```

import pandas as pd
import numpy as np

read_file = pd.read_csv (r'/content/data_banknote_authentication.txt')
read_file.to_csv (r'/content/data_banknote_authentication.csv')
df = pd.read_csv("/content/data_banknote_authentication.csv")
df
  
```

برای نمایش داده ها در فضای کولب فرمت داده ها را از txt به CSV تبدیل می کنیم. در این صورت داده های صورت سوال به صورت زیر خواهند بود :

	Unnamed: 0	3.6216	8.6661	-2.8073	-0.44699	0
0	0	4.54590	8.16740	-2.4586	-1.46210	0
1	1	3.86600	-2.63830	1.9242	0.10645	0
2	2	3.45660	9.52280	-4.0112	-3.59440	0
3	3	0.32924	-4.45520	4.5718	-0.98880	0
4	4	4.36840	9.67180	-3.9606	-3.16250	0
...
1366	1366	0.40614	1.34920	-1.4501	-0.55949	1
1367	1367	-1.38870	-4.87730	6.4774	0.34179	1
1368	1368	-3.75030	-13.45860	17.5932	-2.77710	1
1369	1369	-3.56370	-8.38270	12.3930	-1.28230	1
1370	1370	-2.54190	-0.65804	2.6842	1.19520	1

1371 rows × 6 columns

مطابق چیزی که از قبل نیز می دانستیم فایل دارای ۱۳۷۲ داده می باشد و به طور کلی دارای ۶ ستون می باشد. ستون آخر نیز تارکت های ما خواهد بود و سایر ستون ها می تواند در واقع همان ویژگی های ما باشد. حال به سراغ سایر قسمت های سوال و موارد خواسته شده می رویم :

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

اهمیت شافل کردن دیتا ها در یادگیری ماشین به شرح زیر است:

- جلوگیری از بروز الگوهای کاذب : شافل کردن دیتا ها باعث می شود که الگوریتم یادگیری ماشین الگوهای کاذبی را در دیتا ها تشخیص ندهد. این الگوهای کاذب می توانند ناشی از ترتیب خاصی از داده ها در مجموعه داده باشند.
- بهبود عملکرد الگوریتم : شافل کردن دیتا ها می تواند به بهبود عملکرد الگوریتم یادگیری ماشین کمک کند. این به این دلیل است که شافل کردن باعث می شود که الگوریتم به طور مساوی از تمام داده ها استفاده کند و از تأثیر داده های نادرست یا غیرعادی جلوگیری کند.
- افزایش سرعت یادگیری : شافل کردن دیتا ها می تواند به افزایش سرعت یادگیری الگوریتم یادگیری ماشین کمک کند. این به این دلیل است که شافل کردن باعث می شود که الگوریتم به طور مساوی از تمام داده ها استفاده کند و از تکرار داده ها جلوگیری کند.

```
# Part 2 of question two :
import pandas as pd
from sklearn.model_selection import train_test_split

# Shuffle the data to ensure randomness
df = df.sample(frac=1).reset_index(drop=True)
```

```
# Define the ratio for splitting (e.g., 80% for training, 20% for
evaluation)
train_ratio = 0.8

# Split the data into training and evaluation sets
train_data, eval_data = train_test_split(df, test_size=1 - train_ratio,
random_state=83)

# Save the training and evaluation sets to separate CSV files
train_data.to_csv('train_data.csv', index=False)
eval_data.to_csv('eval_data.csv', index=False)
df
```

	Unnamed: 0	3.6216	8.6661	-2.8073	-0.44699	0
0	834	-0.94255	0.039307	-0.24192	0.31593	1
1	1113	-1.05550	0.794590	-1.69680	-0.46768	1
2	717	4.09320	5.413200	-1.82190	0.23576	0
3	389	-0.36279	8.289500	-1.92130	-3.33320	0
4	1071	-1.36600	0.184160	0.90539	1.58060	1
...
1366	660	-0.11996	6.874100	0.91995	-0.66940	0
1367	743	4.98520	8.351600	-2.54250	-1.28230	0
1368	622	3.77580	7.178300	-1.51950	0.40128	0
1369	1177	-2.07540	1.276700	-0.64206	-1.26420	1
1370	339	4.99230	7.865300	-2.35150	-0.71984	0

1371 rows x 6 columns

بنابراین به این شکل داده را با هم مخلوط کرده ایم و برای قسمت داده های آموزش و داده های تست آن ها را به صورت ۸۰ درصد به ۲۰ درصد تقسیم بندی می کنیم. سپس می توانیم برای راحتی کار خودمان داده های آموزش و ارزیابی را به صورت مجزا در فایل های جداگانه ذخیره کنیم. (مانند کاری که در قسمت بالا انجام داده ایم.)

۳ بدون استفاده از کتابخانه های آماده پایتون، مدل، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کدنویسی کنید تا دو کلاس موجود در دیتاست به خوبی از یکدیگر تفکیک شوند. نمودار تابع اتلاف را رسم کنید و نتیجه دقت ارزیابی روی داده های تست را محاسبه کنید. نمودار تابع اتلاف را تحلیل کنید. آیا می توان از روی نمودار تابع اتلاف و قبل از مرحله ارزیابی با قطعیت در مورد عمل کرد مدل نظر داد؟ چرا و اگر نمی توان، راه حل چیست؟ به طور مثال برای پیدا کردن مدل برای دیتا های بالا می توانیم به طور دستی با کد های پایتون از روش گرادینان و گرادینان نزولی استفاده کنیم: (الگوریتم آن را در صفحات قبل آورده ایم و به آن اشاره شده است.)

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression , SGDClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

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

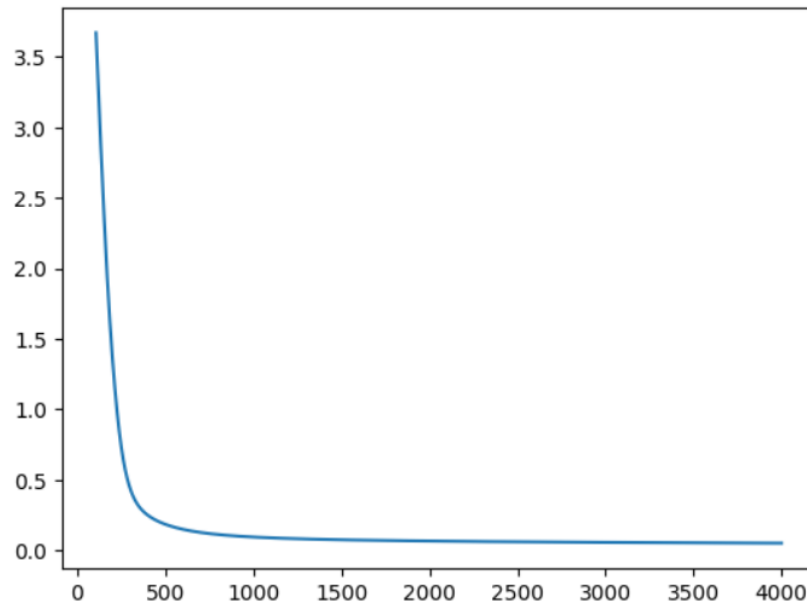
```

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
def sigmoid(x):
    return 1/(1 + np.exp(-x))
def logistic_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
x_train = np.hstack((np.ones((len(x_train) , 1)) , x_train))
x_train.shape
m = 4
w = np.random.randn(m+1 , 1)
print(w.shape)
eta = 0.01
n_epochs = 4000
error_hist = []
for epoch in range(n_epochs):
    y_hat = logistic_regression(x_train , w)

    e = bce(y_train , y_hat)
    error_hist.append(e)
    grads = gradient(x_train , y_train , y_hat)
    w = gradient_descent(w , eta , grads)
    if(epoch + 1) % 100 == 0:
        print(f"Epoch = {epoch} , \t E = {e:.4} \t w={w.T[0]}")
plt.plot(error_hist)

```

Epoch = 99 ,	E = nan	w=[0.88336218 -0.06945917 0.01794809 2.25273392 1.52866013]
Epoch = 199 ,	E = 1.368	w=[0.76713119 -0.8703222 0.2216453 0.99228995 1.10861861]
Epoch = 299 ,	E = 0.4412	w=[0.71171588 -1.13063672 0.09164248 0.13137879 0.91259714]
Epoch = 399 ,	E = 0.2502	w=[0.73935807 -1.21568969 -0.06715699 -0.20424751 0.73433995]
Epoch = 499 ,	E = 0.1849	w=[0.79223139 -1.27172421 -0.18251404 -0.34327062 0.57483598]
Epoch = 599 ,	E = 0.1492	w=[0.84623396 -1.30931286 -0.26561181 -0.42969835 0.44582421]
Epoch = 699 ,	E = 0.127	w=[0.89656444 -1.33622883 -0.33183491 -0.49339649 0.34387704]
Epoch = 799 ,	E = 0.1122	w=[0.94258868 -1.35706639 -0.38757009 -0.54435479 0.26358921]
Epoch = 899 ,	E = 0.1019	w=[0.98469661 -1.37440852 -0.43573453 -0.58721114 0.19999901]
Epoch = 999 ,	E = 0.09433	w=[1.02349157 -1.38973954 -0.47796248 -0.62445434 0.14907052]
Epoch = 1099 ,	E = 0.08861	w=[1.05953659 -1.40392187 -0.51535975 -0.65754316 0.1077399]
Epoch = 1199 ,	E = 0.08411	w=[1.09329264 -1.41745461 -0.54875946 -0.68740075 0.073756]
Epoch = 1299 ,	E = 0.08047	w=[1.12512153 -1.43062206 -0.57881837 -0.71465369 0.04547686]
Epoch = 1399 ,	E = 0.07745	w=[1.15530499 -1.44358153 -0.60606348 -0.73975266 0.02169659]
Epoch = 1499 ,	E = 0.07489	w=[1.18406383 -1.4564158 -0.63092114 -0.76303649 0.00151753]
Epoch = 1599 ,	E = 0.07267	w=[1.21157339 -1.46916441 -0.65373831 -0.78476824 -0.0157393]



همان طور که می توان فهمید با زیاد شدن ایپاک ها میزان خطای الگوریتم استفاده شده در این قسمت کاهش می یابد. می توانیم تعداد ایپاک ها را زیاد یا کم کنیم ولی ممکن است الگوریتمی که در این قسمت استفاده کرده ایم دچار **under modeling** و یا **over modeling** شود. تا به اینجا داده ها را در پایتون **import** کردیم و با توجه به فیلم درس الگوریتم را به صورت دستی وارد کرده ایم. در این قسمت در ابتدا داده ها را مخلوط کردیم و پس از مخلوط کردن داده ها را به بخش های ارزیابی و آموزش تقسیم بندی کرده ایم. سپس با استفاده از الگوریتم گرادیان نزولی داده ها را تفکیک کرده ایم و دو کلاس را از یکدیگر جدا کرده ایم. در نهایت هم نمودار تابع اتلاف را نمایش داده ایم. حال می خواهیم میزان دقت را برای ارزیابی داده ها بررسی کنیم :

```
# accuracy :
x_test = np.hstack((np.ones((len(x_test) , 1)), x_test))
x_test.shape
y_hat = logistic_regression(x_test , w)
accuracy(y_test , y_hat)
```

```
0.9818181818181818
```

همان طور که مشاهده می کنیم دقت این روش بسیار زیاد می باشد و این موضوع را می توانیم از روی مقایسه مقادیر تارگت ها و خروجی های محاسبه شده در روش گرادیان نزولی پی ببریم.

در حالی که نمودار تابع ضرر در طول آموزش می تواند بینش ارزشمندی در مورد همگرایی و پویایی یادگیری مدل شما ارائه دهد، اما همیشه برای نتیجه گیری قطعی در مورد عملکرد آن کافی نیست. در اینجا به چند دلیل اشاره میکنیم:

۱. ****تطبیق بیش از حد و تعمیم:**** یک مدل ممکن است در مجموعه آموزشی عملکرد خوبی داشته باشد (که منجر به کاهش تلفات آموزشی می شود)، اما آزمون واقعی آن با ارزیابی در یک اعتبارسنجی یا مجموعه تست جداگانه همراه است. تطبیق بیش از حد زمانی اتفاق می افتد که یک مدل داده های آموزشی را به خوبی یاد می گیرد، از جمله نویز و نقاط پرت آن، اما نمی تواند به داده های جدید و دیده نشده تعمیم یابد. بررسی ضرر در یک مجموعه اعتبار سنجی جداگانه برای ارزیابی عملکرد تعمیم بسیار مهم است.

۲. ****معیارهای ارزیابی:**** علاوه بر تابع ضرر، سایر معیارهای ارزیابی مانند دقت، غیره برای درک جامع عملکرد مدل ضروری هستند. تابع ضرر ممکن است تمام جنبه های رفتار مدل را در بر نگیرد، به ویژه زمانی که با مجموعه داده های نامتعادل یا الزامات خاص سروکار داریم.

۳. ****مشکلات نرخ یادگیری:**** نرخ یادگیری می تواند به طور قابل توجهی بر همگرایی مدل شما تأثیر بگذارد. گاهی اوقات، منحنی ضرر ممکن است به دلیل نرخ یادگیری خیلی بالا یا خیلی پایین، رفتار نامنظمی از خود نشان دهد. تنظیم نرخ یادگیری در طول آموزش یا استفاده از زمان بندی نرخ یادگیری می تواند کمک کننده باشد.

۴. ****دوران و توقف اولیه:**** منحنی ضرر ممکن است در چند دوره تثبیت نشود. نظارت بر منحنی در تعداد مناسبی از دوره ها ضروری است. علاوه بر این، استفاده از تکنیک هایی مانند توقف زود هنگام می تواند از تطبیق بیش از حد جلوگیری کرده و منابع محاسباتی را ذخیره کند.

۵. ****کیفیت داده و پیش پردازش:** ** کیفیت داده های شما و اثربخشی مراحل پیش پردازش می تواند بر عملکرد مدل تأثیر بگذارد. تجسم توزیع داده ها و درک تأثیر پیش پردازش بر روی منحنی ضرر می تواند آموزنده باشد.

برای ارزیابی آگاهانه تر از عملکرد مدل خود، توصیه می شود:

- ****ارزیابی بر روی یک مجموعه داده جداگانه:** ** از یک مجموعه اعتبارسنجی یا یک مجموعه آزمایشی استفاده کنید که مدل در طول آموزش ندیده است.
- **** نظارت بر معیارهای چندگانه:** ** بسته به ماهیت کار خود، معیارهای ارزیابی مختلفی را در نظر بگیرید.
- **** اجرای اعتبارسنجی متقابل:** ** در صورت امکان، از تکنیک هایی مانند اعتبارسنجی متقابل برای ارزیابی استحکام مدل در زیر مجموعه های مختلف داده ها استفاده کنید.

۴ 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؟

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

نرمال سازی داده ها یک مرحله مهم در پردازش و تحلیل داده ها است 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 نرمال سازی داده ها عبارتند از:

Min-Max Scaling: در این روش، داده ها به گونه ای تغییر می کنند که حداقل و حداکثر آن ها به ترتیب به یک مقدار نگاشته می شوند. فرمول نرمال سازی Min-Max برای یک داده X به صورت زیر است:

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

: Z-score Standardization

در این روش، داده ها به گونه ای تغییر می کنند که میانگین آن ها صفر و انحراف معیاری آن ها یک شود. فرمول نرمال سازی Z-score برای یک داده X به صورت زیر است:

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

اطلاعات بخش "ارزیابی" در فرآیند نرمال سازی به تنهایی معمولاً استفاده نمی شود. بخش ارزیابی معمولاً برای ارزیابی عملکرد مدل یا سیستم پس از اعمال تغییرات (مانند نرمال سازی) استفاده می شود. انتخاب یک روش نرمال سازی باید بر اساس نیازها و خصوصیات داده ها انجام شود. اگر توزیع داده ها نسبت به هم مهم است، ممکن است از Z-score Standardization استفاده کنید. اگر می خواهید داده ها 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 ویژگی ها محدوده مشخصی دارند، ممکن است مقیاس بندی Min-Max ترجیح داده شود. درک ویژگی های داده ها و انتخاب روش عادی سازی بر این اساس ضروری است.

در این قسمت می خواهیم با استفاده از روش اول نرمال سازی را انجام بدهیم. برای این کار باید تمامی داده های داخل یک ستون را که یک ویژگی ما را ایجاد می کنند، دریافت کرده و از میان این داده ها کمترین داده و بیشترین داده را دریافت کنیم و با استفاده از فرمول min and max scaler مقادیر دیتا ها را نرمال سازی و یا استاندارد سازی کنیم:

```
# normalized data :
maxx = df[['feature1', 'feature2', 'feature3', 'feature4']].max()

#print("Maximum value in column 'feature1', 'feature2', 'feature3',
'feature4': ")

minn = df[['feature1', 'feature2', 'feature3', 'feature4']].min()
#print("Minimum value in column 'feature1', 'feature2', 'feature3',
'feature4': ")

for i in range(4):
    df[f'feature{i+1}'] = (df[f'feature{i+1}']-minn[i])/(maxx[i]- minn[i])
    print(df[f'feature{i+1}'])
```

در کد بالا دقیقا همان چیزی را که از قبل گفته بودیم انجام داده ایم. در ابتدا از میان تمامی ستون هایی که جزوه ویژگی های ما بودند ، بیشترین داده و کمترین داده را استخراج کرده ایم. در این حالت دیگر ستون آخر که تارگت مورد نظرمون بود را در نظر نگرفته ایم. زیرا مقادیر داده های ما در این بخش به صورت باینری خواهد بود و نیازی به استفاده از فرمول های نرمال سازی نیست. پس از استخراج داده ها تا به اینجای کار از فرمول معروف زیر برای نرمال سازی داده ها استفاده می کنیم :

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

به طور مثال چند نمونه از داده ها را مشاهده می کنیم :

```
...
893    0.375210
357    0.701945
1119    0.277971
707    0.880889
699    0.790970
Name: feature1, Length: 1371, dtype: float64
549    0.554162
1165    0.435848
1293    0.863104
841    0.261324
760    0.649167
```

حال می توانیم داده ها را مانند قبل به دو دسته test و train تقسیم بندی کنیم که در ادامه سوالات به آن ها می پردازیم :

```
X = df[["feature1" , "feature2" , "feature3" , "feature4"]].values
y = df[["feature5"]].values
X , y
```

```
(array([[7.94265481e-01, 5.54161506e-01, 2.60444569e-01, 8.68317921e-01],
       [3.49335468e-01, 4.35847736e-01, 3.01600362e-01, 8.18344745e-01],
       [4.11050776e-04, 8.63104170e-01, 2.34865057e-01, 3.64494394e-01],
       ...,
       [2.77971284e-01, 2.31901574e-01, 7.38962242e-01, 7.24634242e-01],
       [8.80889024e-01, 6.63569657e-01, 1.42292201e-01, 8.14346636e-01],
       [7.90969863e-01, 7.40999899e-01, 1.40887845e-01, 7.71313002e-01]]),
 array([[0],
       [1],
       [1],
       ...,
       [1],
       [0],
       [0]]))
```

در بالا داده های ویژگی و تارگت ها را به طور مجزا نشان می دهیم و مشخص می کنیم و در دو دیتا فریم مجزا آن ها را قرار می دهیم که در این حالت X و y جدید ما می باشند.

۵ تمام قسمت های «۱» تا «۳» را با استفاده از داده های نرمال شده تکرار کنید و نتایج پیش بینی مدل را برای پنج نمونه داده نشان دهید.

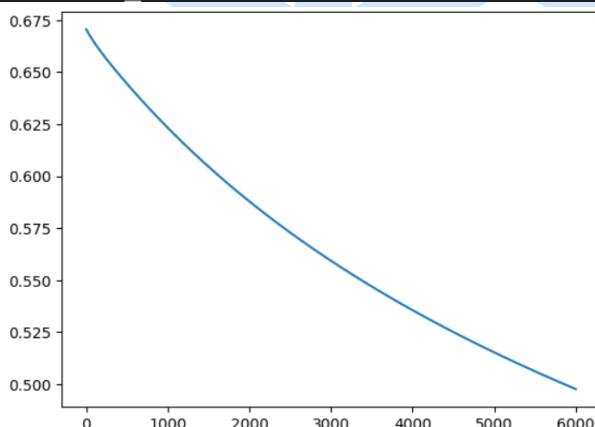
داده ها را که در قسمت قبل نرمال کرده ایم در این قسمت می خواهیم با استفاده از همین داده های جدید مراحل قبل را تکرار کنیم و نتایج مدل را نشان بدهیم :

```
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
```

همان طور که از قبل هم گفته ایم داده های feature و target را باید به دو دسته train و test تبدیل کنیم که در کد بالا آن را انجام داده ایم. حال باید تابع اتلاف را رسم کنیم و در ادامه دقت را در آموزش و ارزیابی را ببایم :

```
x_train = np.hstack((np.ones((len(x_train) , 1)) , x_train))
x_train.shape
m = 4
w = np.random.randn(m+1 , 1)
print(w.shape)
eta = 0.01
n_epochs = 6000
error_hist = []
for epoch in range(n_epochs):
    y_hat = logistic_regression(x_train , w)

    e = bce(y_train , y_hat)
    error_hist.append(e)
    grads = gradient(x_train , y_train , y_hat)
    w = gradient_descent(w , eta , grads)
    if(epoch + 1) % 100 == 0:
        print(f"Epoch = {epoch} , \t E = {e:.4} \t w={w.T[0]}")
plt.plot(error_hist)
```



این روند را دقیقاً مانند قبل پیش می بریم :

```
x_test = np.hstack((np.ones((len(x_test) , 1)), x_test))
x_test.shape
y_hat = logistic_regression(x_test , w)
accuracy(y_test , y_hat)
```

0.88

مشاهده می شود که با بالا بردن تعداد ایپاک ها درصد دقت بیشتر خواهد شد :

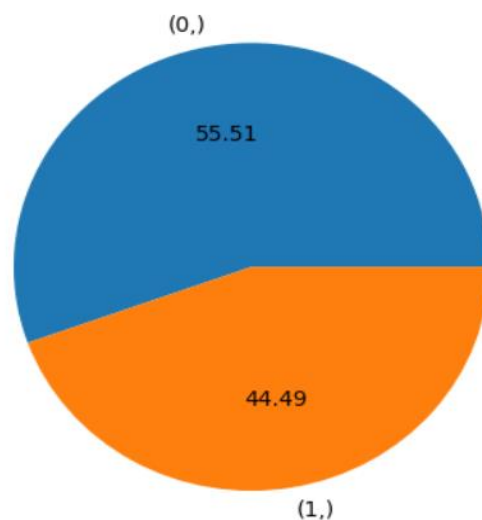
۶ با استفاده از کدنویسی پایتون وضعیت تعادل داده ها در دو کلاس موجود در دیتاست را نشان دهید. آیا تعداد نمونه های کلاس ها با هم برابر است؟ عدم تعادل در دیتاست می تواند منجر به چه مشکلاتی شود؟ برای حل این موضوع چه اقداماتی می توان انجام داد؟ پیاده سازی کرده و نتیجه را مقایسه و گزارش کنید.

برای مشخص کردن تعادل داده ها از کد آماده زیر استفاده کرده ایم :

```
# part 6 :

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

# value_count = y.value_counts()
# value_count
new_y = pd.DataFrame(y, columns=['Column_A'])
new_y.value_counts()
new_y.value_counts().plot.pie(autopct = "%.2f")
```



می بینیم که تعادل داده ها وجود ندارد و تعداد نمونه کلاس ها با یکدیگر برابر نیست.

عدم تعادل در دیتاست، به معنای عدم توازن در توزیع کلاس‌ها یا دسته‌های مختلف داده‌ها، می‌تواند به مشکلات مختلفی منجر شود. در زیر چند مشکل اصلی آورده شده است:

۱. **مشکل در آموزش مدل:**

- در دیتاست‌های ناتوانمند به تعداد نمونه‌های هر کلاس، مدل ممکن است با مشکل مواجه شود. این موضوع می‌تواند منجر به یادگیری ناکافی برای کلاس‌های کم‌نمونه شود.

۲. **تأثیرات انحرافی:**

- در صورتی که تعداد نمونه‌های یک کلاس زیاد باشد و برای کلاس‌های دیگر کم باشد، مدل ممکن است به سمتی خاص بیفتد و به کلاس‌های کم‌نمونه کمتر توجه کند. این موضوع می‌تواند به انحراف (bias) پیش‌بینی‌ها منجر شود.

۳. **دقت تخمین‌گر:**

- در صورت عدم تعادل، دقت تخمین‌گر برای کلاس‌های با تعداد نمونه بیشتر بالا می‌رود، اما برای کلاس‌های کم‌نمونه کاهش می‌یابد. این ممکن است باعث نادرست فهم شود که مدل بهترین عملکرد را ارائه می‌دهد.

۴. **افزایش هزینه آموزش:**

- برای مدل‌هایی که با دیتاست‌های ناتوانمند آموزش داده می‌شوند، احتمالاً نیاز به تلاش و هزینه زیادی برای دستیابی به عملکرد خوب دارند.

۵. **اهمال کلاس‌های کم‌نمونه:**

- ممکن است مدل به خاطر تعداد کم نمونه‌ها به صورت اشتباهی کلاس‌های کم‌نمونه را اهمال کند یا به اشتباه آنها را به عنوان نمونه‌های کلاس اکثریت (majority class) در نظر بگیرد.

۶. **پایداری نتایج:**

- دقت و کارایی مدل ممکن است در مواجهه با داده‌های جدید تحت تأثیر قرار گیرد، زیرا مدل ممکن است بر اساس نمونه‌های زیاد یک کلاس و بی‌توجه به کلاس‌های کم‌نمونه باشد.

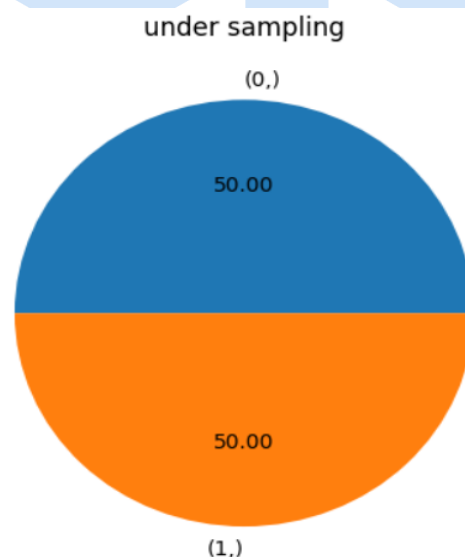
برای حل مشکلات مربوط به عدم تعادل دیتاست، روش‌هایی مانند **oversampling** افزایش نمونه‌های کم‌نمونه، **undersampling** کاهش نمونه‌های بیشتری، یا استفاده از الگوریتم‌های خاص مانند **SMOTE** معمولاً مورد استفاده قرار می‌گیرند.

حال ما در این قسمت از یکی از این الگوریتم‌های بالا استفاده می‌کنیم:

```
! pip install -U imbalanced-learn
# undersampling :
from imblearn.under_sampling import RandomUnderSampler

y = pd.DataFrame(y, columns=[''])
rus = RandomUnderSampler(sampling_strategy=1)
x_res_undersampling , y_res_undersampling = rus.fit_resample(X , y)
ax = y_res_undersampling.value_counts().plot.pie(autopct = '%.2f')
    = ax.set_title("under sampling")
```

با مراجعه به چند سایت می‌توانیم داده‌ها را به این صورت متعادل کنیم:



در این صورت تعداد نمونه کلاس ها با یکدیگر برابر شده است .

۷ فرآیند آموزش و ارزیابی مدل را با استفاده از یک طبقه بند آماده پایتونی انجام داده و این بار در این حالت چالش عدم تعادل داده های کلاس ها را حل کنید.

این طور که من متوجه شدم از این مورد این است که می خواهیم این بار با استفاده از کتابخانه های آماده و در حالت تعادل داده ها طبقه بندی را انجام بدهیم. پس باید با استفاده از کد مورد قبل تعادل را ایجاد کنیم و سپس با استفاده از کتابخانه آماده در پایتون طبقه بندی را انجام دهیم :

```
# part 7 :  
  
from sklearn.linear_model import LogisticRegression  
from sklearn.datasets import make_classification  
from sklearn.model_selection import train_test_split  
from sklearn.metrics import accuracy_score  
X = x_res_undersampling  
y = y_res_undersampling  
x_train , x_test , y_train , y_test = train_test_split(X , y , test_size =  
0.2)  
model = LogisticRegression()  
model.fit(X , y)  
y_hat = model.predict(x_test)  
model.score(x_test , y_test)  
y_test.shape ,  
y_hat = y_hat.reshape(244 , 1)  
y_test.shape , y_hat.shape  
  
from sklearn.metrics import accuracy_score  
score = accuracy_score(y_test, y_hat)  
score
```

0.9959016393442623

که می بینیم دقت آموزش و ارزیابی داده ها و نمونه ها افزایش یافته است :

حال می خواهیم مقایسه ای بین داده های متعادل نشده و داده های متعادل شده انجام دهیم. در حالت کلی متعادل کردن داده ها باعث تقسیم داده ها به اندازه مشخص خواهد شد و این باعث کاهش پراکندگی و خطای ناگهانی خواهد شد و طبیعتاً دقت کار ما را نیز بالاتر خواهد برد. برای همین موضوع می خواهیم با استفاده از کتابخانه های پایتون و در حالت عدم تعادل نیز دقت آموزش و ارزیابی مدل خودمان را بباییم :

```
# In imbalance mode and using ready libraries :

from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression , SGDClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

X = df[["feature1" , "feature2" , "feature3" , "feature4"]].values
y = df[["feature5"]].values
x_train , x_test , y_train , y_test = train_test_split(X , y , test_size =
0.2)
model = LogisticRegression(random_state = 93, solver='sag', max_iter=200)
model.fit(X , y)
y_hat = model.predict(x_test)
model.score(x_test , y_test)
```

0.9745454545454545

همان طور که می بینیم دقت داده ها و نمونه ها در این حالت که تعادلی وجود ندارد کمتر از حالت متعادل می باشد.

۳ سوال سوم)

۱ به این پیوند مراجعه کرده و یک دیتاست مربوط به «بیماری قلبی» را دریافت کرده و توضیحات مختصری در مورد هدف و ویژگی های آن بنویسید. فایل دانلودشده دیتاست را روی گوگل درایو خود قرار داده و با استفاده از دستور gdown آن را در محیط گوگل کولب بارگذاری کنید.

با توجه به اطلاعاتی که در خود سایت این دیتاست نوشته شده است می توان فهمید که این مجموعه داده شامل شاخص های مختلف مرتبط با سلامت برای نمونه ای از افراد است. در اینجا توضیح مختصری از هر ستون آورده شده است:

Heart Disease or Attack : نشان می دهد که آیا فرد دچار بیماری قلبی یا حمله قلبی شده است (دودویی: ۰ = خیر، ۱ = بله).

HighBP : وضعیت فشار خون بالا (باینری: ۰ = خیر، ۱ = بله).

HighChol: وضعیت کلسترول بالا (دودویی: ۰ = خیر، ۱ = بله).

CholCheck: دفعات بررسی کلسترول (طبقه ای).

BMI: شاخص توده بدن (مستمر).

Smoker: وضعیت سیگار کشیدن (دودویی: ۰ = خیر، ۱ = بله).

Stroke: سابقه سکته مغزی (باینری: ۰ = خیر، ۱ = بله).

Diabetes: وضعیت دیابت (دودویی: ۰ = خیر، ۱ = بله).

PhysActivity: سطح فعالیت بدنی (طبقه ای).

Fruits: فراوانی مصرف میوه (قسمتی).

Veggies: فراوانی مصرف سبزیجات (قسمتی).

HvyAlcoholConsump: وضعیت مصرف الکل سنگین (باینری: ۰ = خیر، ۱ = بله).

AnyHealthcare: دسترسی به هر مراقبت بهداشتی (باینری: ۰ = خیر، ۱ = بله).

NoDocbcCost: بدون پزشک به دلیل هزینه (باینری: ۰ = خیر، ۱ = بله).

GenHlth: ارزیابی سلامت عمومی (طبقه ای).

MentHlth: ارزیابی سلامت روان (مقوله ای).

PhysHlth: ارزیابی سلامت جسمانی (طبقه ای).

DiffWalk: وضعیت دشواری راه رفتن (باینری: ۰ = خیر، ۱ = بله).

Sex: جنسیت فرد (دودویی: ۰ = زن، ۱ = مرد).

Age: سن فرد (مستمر).

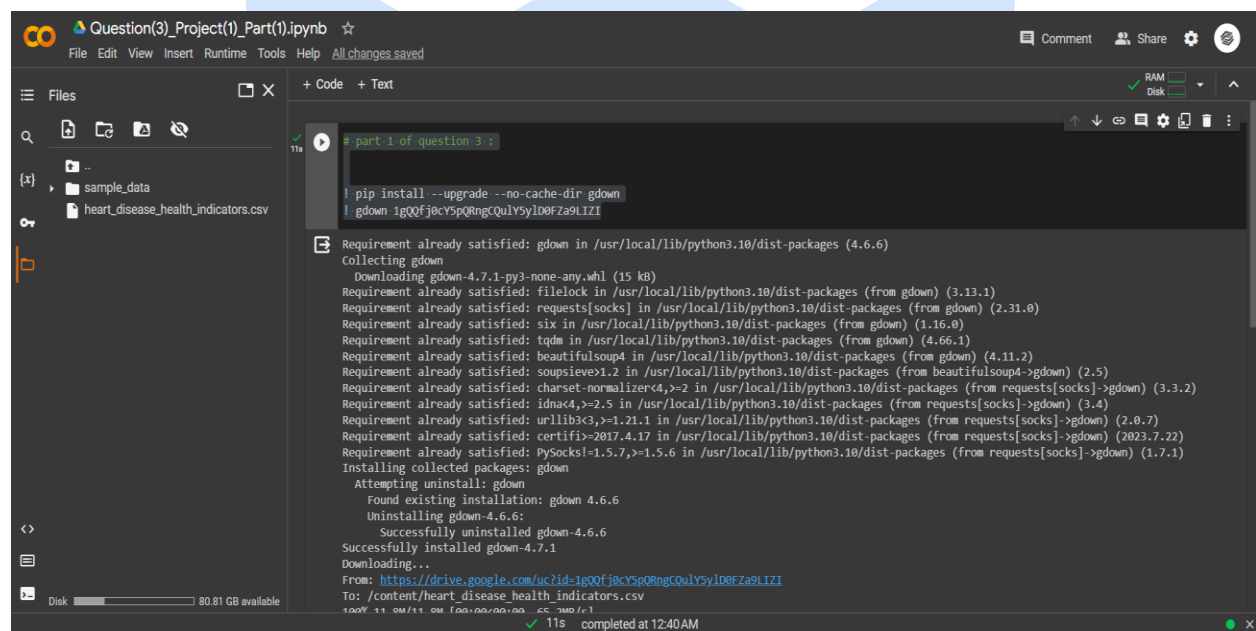
Education: مقطع تحصیلی (قسمتی).

Income: سطح درآمد (مقوله ای).

این مجموعه داده حاوی انواع اطلاعات مرتبط با سلامتی، عوامل سبک زندگی و اطلاعات جمعیتی برای گروهی از افراد است که آن را برای بررسی همبستگی ها و عوامل خطر بالقوه بیماری قلبی و سایر شرایط سلامتی مناسب می کند.

دقیقا مانند مورد قبل عمل می کنیم و قایل دانلود شده را در داخل کولب import می کنیم :

```
# part 1 of question 3 :  
  
! pip install --upgrade --no-cache-dir gdown  
! gdown 1gQQfj0cY5pQRngCQulY5ylD0FZa9LIZI
```



```
# part 1 of question 3 :  
  
! pip install --upgrade --no-cache-dir gdown  
! gdown 1gQQfj0cY5pQRngCQulY5ylD0FZa9LIZI  
  
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.6.6)  
collecting gdown  
  Downloading gdown-4.7.1-py3-none-any.whl (15 kB)  
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: six in /usr/local/lib/python3.10/dist-packages (from gdown) (1.16.0)  
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)  
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)  
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) (3.4)  
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) (2023.7.22)  
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 4.6.6  
    Uninstalling gdown-4.6.6:  
      Successfully uninstalled gdown-4.6.6  
  Successfully installed gdown-4.7.1  
Downloading...  
From: https://drive.google.com/uc?id=1gQQfj0cY5pQRngCQulY5ylD0FZa9LIZI  
To: /content/heart_disease_health_indicators.csv  
100% 11s 11s completed at 12:40 AM
```

۲ ضمن توجه به محل قرارگیری هدف و ویژگی ها، دیتاست را به صورت یک دیتافریم درآورده و با استفاده از دستورات پایتونی، ۱۰۰ نمونه داده مربوط به کلاس «۱» و ۱۰۰ نمونه داده مربوط به کلاس «۰» را در یک دیتافریم جدید قرار دهید و در قسمت های بعدی با این دیتافریم جدید کار کنید.

داده ها قبل از pre process به این صورت می باشند :

```
# part 2 of question 3 :  
  
import pandas as pd  
import numpy as np  
df = pd.read_csv("/content/heart_disease_health_indicators.csv")  
df
```


	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	
0	0	1	1	1	40	1	0	0	0	0	...	1	0	
1	0	0	0	0	25	1	0	0	1	0	...	0	1	
2	0	1	1	1	28	0	0	0	0	1	...	1	1	
3	0	1	0	1	27	0	0	0	1	1	...	1	0	
4	0	1	1	1	24	0	0	0	1	1	...	1	0	
...	
253656	0	0	0	1	25	0	0	0	1	1	...	1	0	
253657	0	0	1	1	24	0	0	0	0	0	...	1	0	
253658	0	0	0	0	27	0	0	0	1	0	...	1	1	
253659	0	0	1	1	37	0	0	2	0	0	...	1	0	
253660	0	0	1	1	34	1	0	0	0	1	...	1	0	

253661 rows x 22 columns

با توجه به دیتافریم ها ستون اول مربوط به تارگت و هدف می باشد و ستون های بعدی همگی جزوه ویژگی ها می باشند. پس به صورت زیر عمل می کنیم :

هدف ما به صورت باینری می باشد. یعنی با توجه به اطلاعات سوال ، اگر داده تارگت ما که ستون اول داده ها می باشد برابر با یک باشد یعنی مراجعه کننده به بیماری مبتلا می باشد اما اگر برابر با صفر باشد می گوئیم که شخص بیماری ندارد. حال اگر بیماران را در کلاس یک و افرادی که بیمار نیستند را در کلاس صفر قرار دهیم ، دو کلاس خواهیم داشت. می خواهیم در این صورت ۱۰۰ نمونه از کلاس صفر و ۱۰۰ نمونه از کلاس یک را استخراج کنیم و آن ها را در یک دیتافریم جدیدی قرار دهیم. برای همین منظور می توانیم به صورت زیر پیش برویم :

```
# part 2 of question 3 :

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	Genl
8	1	1	1	1	30	1	0	2	0	1	...	1	0	
20	1	1	1	1	22	0	1	0	0	1	...	1	0	
26	1	1	1	1	37	1	1	2	0	0	...	1	0	
27	1	1	1	1	28	1	0	2	0	0	...	1	0	
47	1	1	1	1	25	1	0	0	0	1	...	1	0	
...	
750	1	1	1	1	25	1	0	2	1	1	...	1	0	
774	1	0	1	1	29	0	0	0	0	1	...	1	0	
784	1	1	0	1	31	0	0	2	0	0	...	1	0	
797	1	1	1	1	30	0	0	0	1	0	...	1	0	
807	1	1	1	1	32	0	0	2	1	0	...	1	0	

100 rows x 22 columns

با این حرکت داده ها را بر حسب label آنها به دو کلاس دسته بندی کرده ایم و آن ها را مطابق قبل در دو دیتافریم مجزا قرار داده ایم :

۳ با استفاده از حداقل دو طبقه بند آماده پایتون و در نظر گرفتن فرایدارمترهای مناسب، دو کلاس موجود در دیتاست را از هم تفکیک کنید. نتیجه دقت آموزش و ارزیابی را نمایش دهید.

در قسمت قبل داده های دو کلاس را در دو دیتافریم مجزا قرار داده ایم و حال می خواهیم این دو کلاس را با استفاده از دو کتابخانه آماده در پایتون از یکدیگر تفکیک کنیم : (در مورد مرز تصمیم گیری چیزی گفته نشده است.)

```
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

# 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)

# Separate features (X) and target (y)
X = combined_df.drop('HeartDiseaseorAttack', axis=1) # Assuming 'target'
is the column you want to predict
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 Logistic Regression model
logistic_model = LogisticRegression()
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=1000,
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}')

```

همان طور مانند قبل باید داده ها را به دو دسته test و train تقسیم بندی کنیم و از دو دیتافریمی که در مورد قبل ایجاد کرده ایم برای این مورد نیز بهره می بریم. از همان دو کتابخانه آماده ای که در سوال یک استفاده

```

Logistic Regression Accuracy: 0.70
Random Forest Accuracy: 0.62

```

کرده ایم در این مورد هم استفاده می کنیم و دقت آموزش و ارزیابی هر کدام را نیز نمایش می دهیم. می توانیم از کتابخانه های بیشتری نیز برای بالا بردن دقت در این مورد استفاده کنیم :

نکته مهم : برای تجسم مرزهای تصمیم و مناطق یک مدل آموزش دیده، اگر مجموعه داده شما دارای دو ویژگی باشد، می توانید از نمودار دو بعدی استفاده کنید. با این حال، اگر مجموعه داده شما بیش از دو ویژگی داشته باشد، تجسم مستقیم مرزهای تصمیم گیری چالش برانگیزتر می شود. در آن صورت، می توانید از تکنیک های کاهش ابعاد مانند تجزیه و تحلیل اجزای اصلی (PCA) برای نمایش داده های خود در فضای دوبعدی استفاده کنید.

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

# 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)

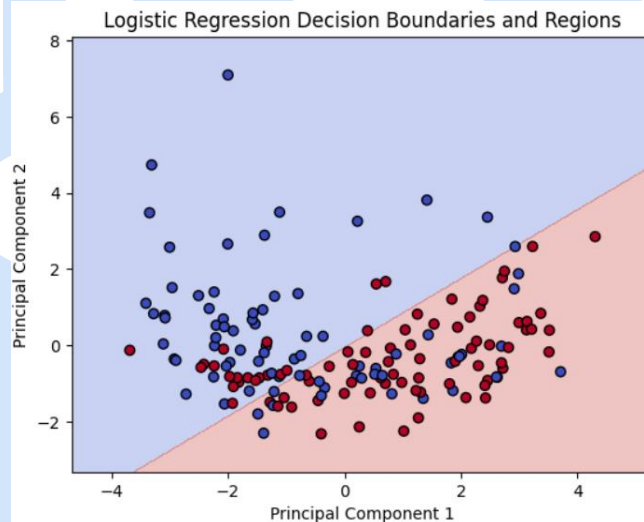
# Create a meshgrid to plot the decision boundaries
h = 0.02
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1

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()
```



حال می خواهیم کتابخانه های بیشتری را در این مورد با هم امتحان کنیم :

(کتابخانه های SVM و KNN در پایتون)

```
# part 3 of question 3 :

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# 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)
```

```

# Separate features (X) and target (y)
X = combined_df.drop('HeartDiseaseorAttack', axis=1) # Assuming 'target'
is the column you want to predict
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 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: 0.70
KNN Accuracy: 0.68

که می بینیم باز هم به دقت مشابه در آموزش و ارزیابی می رسم:

۴ در حالت استفاده از دستورات آمادهٔ سایکیت لرن، آیا راهی برای نمایش نمودار تابع اتلاف وجود دارد؟ پیاده سازی کنید.

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

باید هسته مرکزی 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 LogisticRegression از دستورات مربوط به محاسبه تابع اتلاف به صورت عادی پیروی نمی کردند و باید به صورت نقطه ای و iteration آن ها را مشخص می کردیم. برای همین منظور در این قسمت من از یک مدل آماده در سایکیت لرن استفاده کرده ام که دستورات آماده آن به صورت زیر می باشد :

```
# Part 4 :

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import log_loss

model = SGDClassifier(loss='log', random_state=83)
losses = []
epochs = 5000

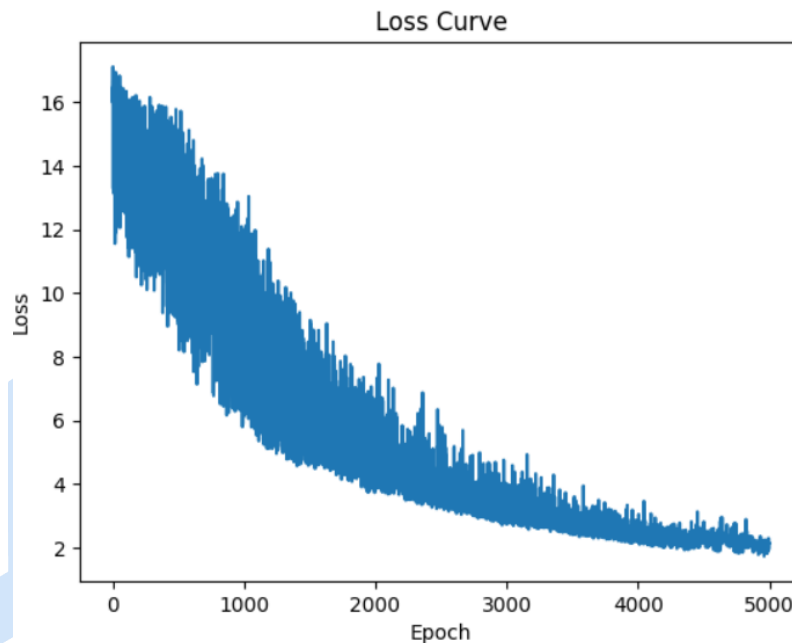
for _ in range(epochs):

    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('Loss Curve')
plt.show()
```

در ابتدا تمامی کتابخانه های مورد نیاز را در محیط import می کنیم و با استفاده از کلاس SGDClassifier مدل مورد نظرم 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 اتلافی را در داخل آن بریزیم. در این قسمت برای بهبودی عملکرد مقدار ایپاک ها را زیاد و روی ۵۰۰۰ قرار می دهیم اما این مدل برای ایپاک های کمتر از این هم جوابگو خواهد بود. در ادامه ایپاک ها را به مدل مورد نظرم اعمال می کنیم و تابع losses را append می کنیم.

در ادامه نیز با استفاده از دستورات و کتابخانه های پایتون تابع اتلافی مورد نظرم را رسم می کنیم :



می بینیم که در این حالت نیز هر چه تعداد ایپاک ها بیشتر می شود مقدار تلاف ما نیز کمتر می شود تا به یک مقدار مشخصی میل کند. حتی می توانیم برای قشنگ تر شدن ظاهر نمودار تابع اتلافی سوال ، از دستورات **curve fitting** استفاده کنیم. اما باید به تعداد ایپاک ها توجه کنیم زیرا انتخاب آن ها خیلی تاثیری زیادی در خطای سیستم دارد. اگر مقدار آن ها کم باشد تابع خطای زیادی را به ما نشان می دهد و از آن طرف نیز اگر تعداد ایپاک ها را زیاد انتخاب کنیم ممکن است مدل ما دچار **under modeling** شود، پس تعداد ایپاک ها مهم است.

۵ یک شاخصه ارزیابی غیر از (Accuracy) تعریف کنید و بررسی کنید که از چه طریقی می توان این شاخص جدید را در ارزیابی داده های تست نمایش داد. پیاده سازی کنید.

می توانیم از شاخص جدیدی که به تازگی در کلاس درس ارائه شده است ، استفاده کنیم :

ماتریس کانفیوژن (Confusion Matrix) یک ابزار آماری است که برای ارزیابی عملکرد یک مدل طبقه بندی استفاده می شود. این ماتریس به صورت یک جدول مربعی نمایش داده می شود که در آن هر سطر مربوط به یک کلاس واقعی و هر ستون مربوط به یک کلاس پیش بینی شده است.

در هر سطر از ماتریس، تعداد نمونه های واقعی آن کلاس که به درستی پیش بینی شده اند و تعداد نمونه های واقعی آن کلاس که به اشتباه پیش بینی شده اند، نشان داده می شود .

معیارهای مختلفی از ماتریس کانفیوژن برای ارزیابی عملکرد یک مدل طبقه بندی استفاده می شود. برخی از این معیارها عبارتند از:

- دقت (Accuracy): نسبت کل نمونه های پیش بینی شده صحیح به کل نمونه ها.

$$\text{Accuracy} = (TP + TN) / (TP + FP + FN + TN)$$

- صحت (Precision): نسبت نمونه های پیش بینی شده صحیح به کل نمونه هایی که به عنوان آن کلاس پیش بینی شده اند.

$$\text{Precision} = TP / (TP + FP)$$

- فراخوانی (Recall): نسبت نمونه های پیش بینی شده صحیح به کل نمونه های واقعی آن کلاس.

$$\text{Recall} = TP / (TP + FN)$$

- F1-score: میانگین حسابی دقت و فراخوانی.

$$\text{F1-score} = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$$

انتخاب معیار مناسب برای ارزیابی عملکرد یک مدل طبقه بندی به عوامل مختلفی بستگی دارد، از جمله اینکه کدام کلاس ها از اهمیت بیشتری برخوردارند و اینکه خطاهای پیش بینی در کدام کلاس ها قابل قبول تر هستند. ماتریس کانفیوژن یک ابزار مفید برای ارزیابی عملکرد یک مدل طبقه بندی است. با استفاده از این ماتریس، می توان نقاط قوت و ضعف یک مدل را شناسایی کرد و برای بهبود عملکرد آن اقدامات لازم را انجام داد.

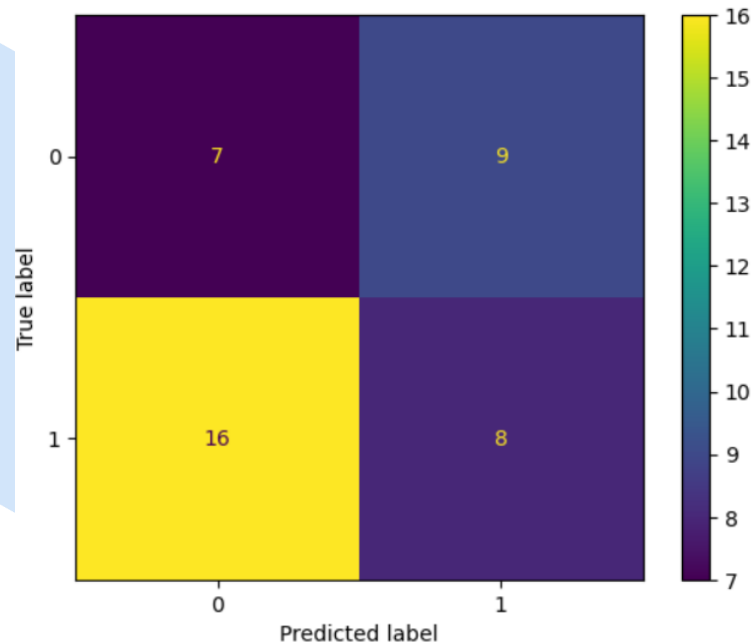
```
# Part 5 :

from sklearn.metrics import confusion_matrix , f1_score
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test , y_pred_logistic)
F1 = f1_score(y_test , y_pred_logistic , average=None)
F1

from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

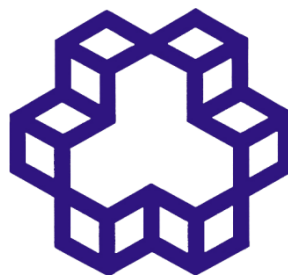
cm = confusion_matrix(y_test, y_pred_logistic)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

نتایج کد :



مراجع

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html



1928

K. N. Toosi University of Technology

Faculty of Electrical Engineering

[Mini Project(1)]

By:

[Iman Fekri]

Student number:

[9929083]

Professor:

[Dr.Aliyari]

autumn 2023