به نام خدا

درس مبانی سیستم های هوشمند مینی پروژه اول

نام و نام خانوادگی: سعید اصلانی امین اباد

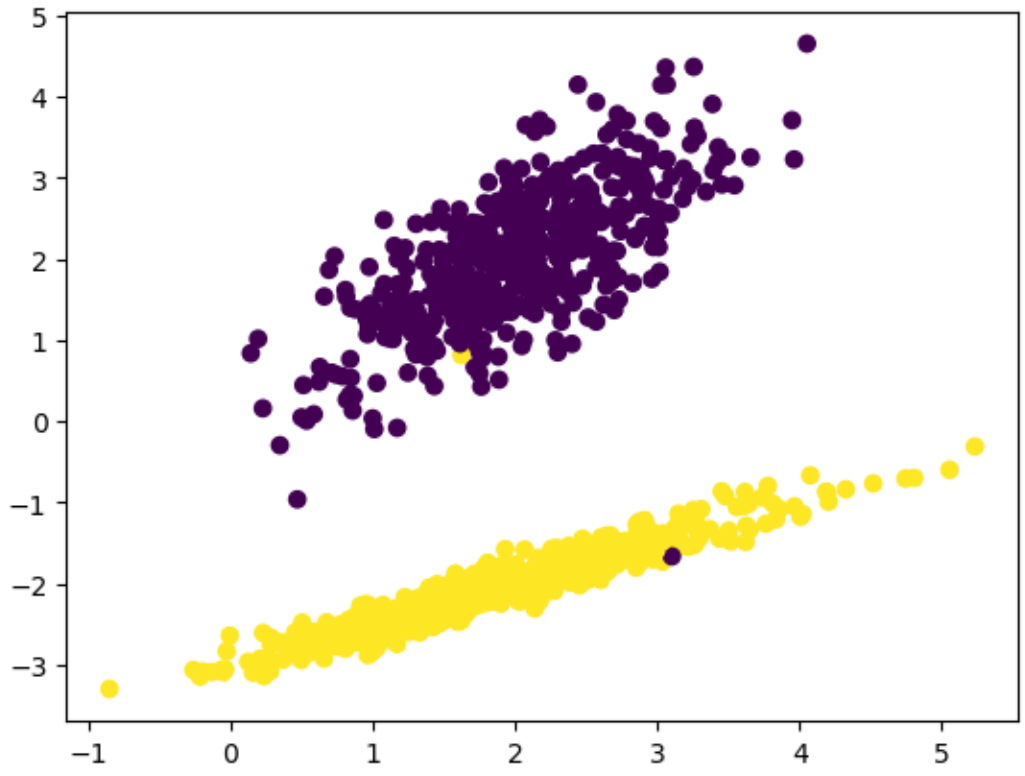
شماره دانشجویی:9819483

سوال اول

1.1

```
from sklearn.datasets import make_classification
x,y = make_classification(n_samples=1000, n_features=2
,n_redundant=0,n_classes=2,class_sep=2,n_clusters_per_class=1,random_state
=83)
```

با استفاده از make_classification دیتاست مورد نظر را تولید میکنیم. برای تعداد نمونه ها با samples-n و برای تعداد کلاس ها با classes-n و برای تعداد ویژگی ها با features-n کار می کنیم.



2.1

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier
x_train,x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2,random_state=83)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

با استفاده ازsplit-test-train داده ها را برای قسمت های آموزش و تست با نسبت های ۸۰ و ۲۰ جدا می کنیم.

```
model = LogisticRegression(solver='sag',max_iter=100, random_state=83)
model.fit(x_train, y_train)
model.predict(x_test), y_test
```

برای حالت طبقه بندی اول از LogisticRegression استفاده می کنیم و الگوریتم بهینه سازی آن را برابر ’sag‘ که بر پایه ي گرادیان نزولی است و iter-max که تعداد تکرار ها را مشخص می کند، برابر100 قرار می دهیم.

دقت اموزش و تست برابر زیر است:

```
model.score(x_train, y_train)
```

```
0.98875
```

```
model.score(x_test, y_test)
```

```
0.985
```

طبقه بند دوم آماده در پایتونی که استفاده کردیم، طبقه بند SGDClassifier است که بر پایه ي گرادیان نزولی است. برای loss ازتابع لگاریتمی استفاده شده است.

```
model = SGDClassifier(loss='log_loss',max_iter=100, random_state=83)
model.fit(x_train, y_train)
model.predict(x_test), y_test
```

دقت اموزش و تست برابر زیر است:

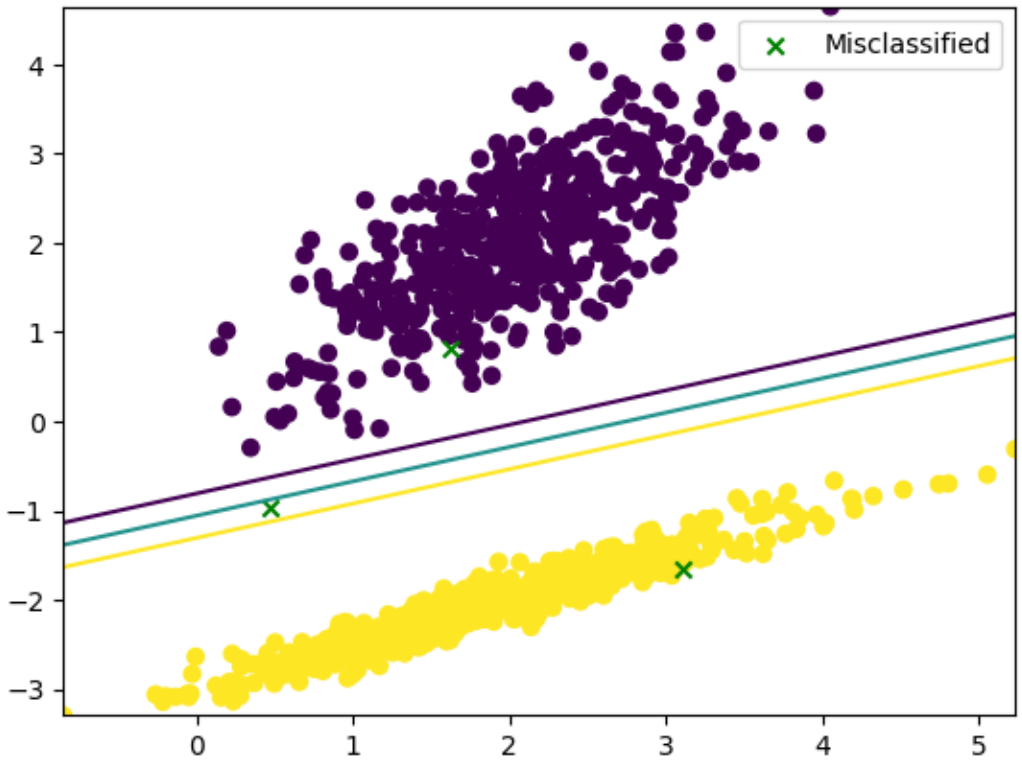
```
model.score(x_train, y_train)
```
```
0.9875
```

```
model.score(x_test, y_test)
```
```
0.985
```

3.1

ابتدا مینیمم و ماکزیمم داده ها برای هر دو کلاس را مشخص می کنیم و خطی با مینمم و ماکزیمم کلاس اول و کلاس دوم با تعداد ۵۰۰ مقدار می کشیم.

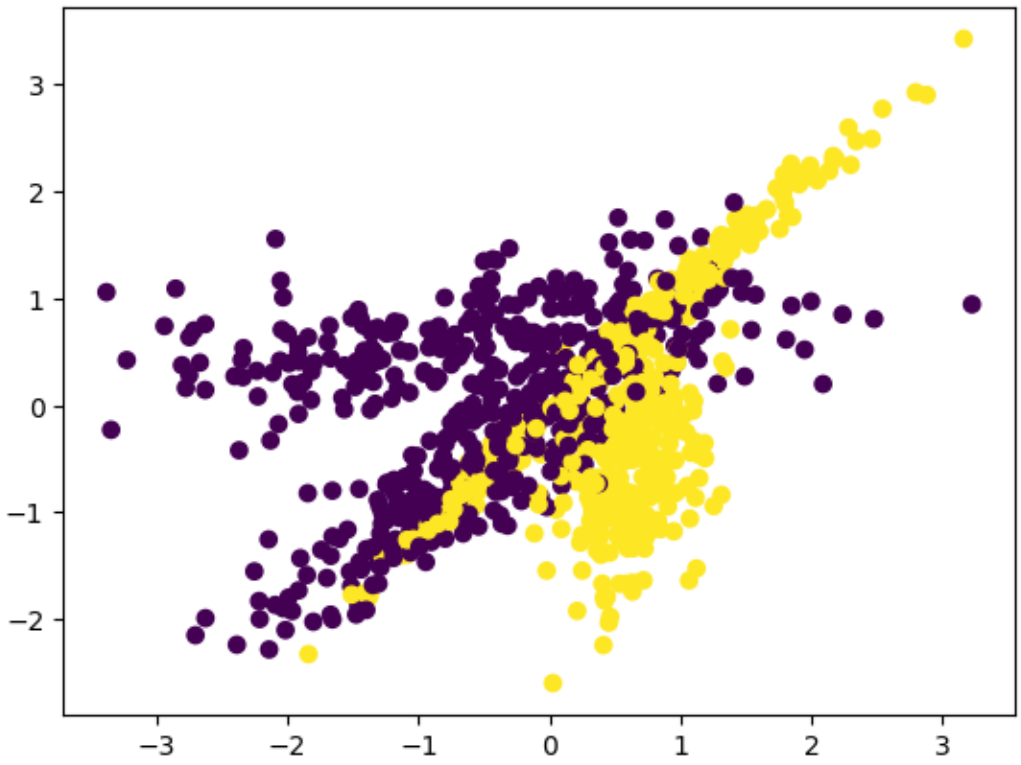
```
import numpy as np
import matplotlib.pyplot as plt
x1_min, x2_min = x.min(0)
x1_max, x2_max = x.max(0)

n=500
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m , x2m = np.meshgrid(x1r, x2r)

xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
ym = model.decision_function(xm)

predictions = model.predict(x)
misclassified_indices = np.where(predictions != y)[0]


plt.scatter(x[predictions == y, 0], x[predictions == y, 1],
c=y[predictions == y])
plt.scatter(x[misclassified_indices, 0], x[misclassified_indices, 1],
marker='x', c='green', label='Misclassified')

plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels=[-1, 0, 1])

plt.legend()
plt.show()
```



4.1

برای سخت تر کردن کار، می توانیم مقدار sep-class را کمتر کنیم. هر چه مقدار این پارامتر را کمتر کنیم، تو هم رفتگی داده ها نیز بیشتر می شود و همچنین می توانیم مقدار class-per-cluster-n را هم برای سخت تر شدن کار، بیشتر کنیم.

```
x, y= make_classification(n_samples=1000,
                          n_features=2,
                          n_redundant=0,
                          n_classes=2,
                          n_clusters_per_class=2,
                          class_sep=0.6,
                          random_state=83)

plt.scatter(x[:,0],x[:,1], c=[y])
plt.show
```




```
model = LogisticRegression()
model.fit(X,y)
x_train,x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = LogisticRegression(solver='sag', max_iter=200, random_state=83)
model.fit(x_train, y_train)
model.predict(x_test), y_test
model.score(x_train, y_train)
```

```
0.80375
```

```
model.score(x_test, y_test)
```

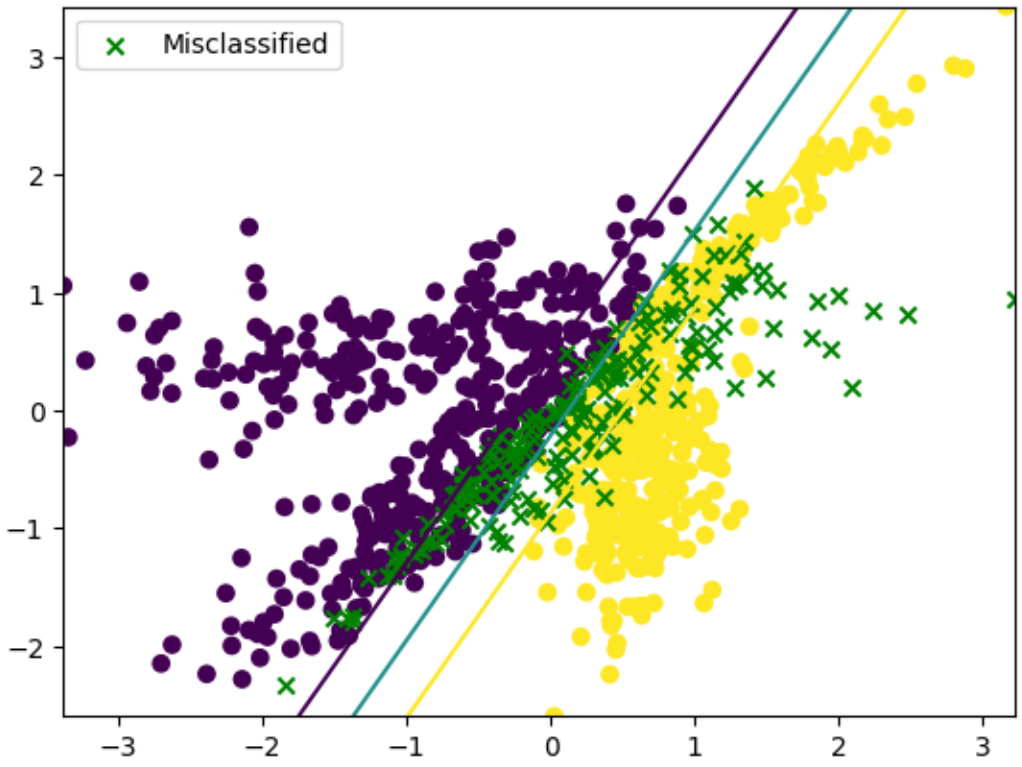
```
0.82
```

```
model1 = SGDClassifier(loss= 'log_loss', random_state=83)
model1.fit(x_train, y_train)
model1.score(x_train, y_train)
0.79375

model1.score(x_test, y_test)
```

```
0.825
```

```
x1_min, x2_min = X.min(0)
x1_max, x2_max = X.max(0)
```

```
n=500
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m , x2m = np.meshgrid(x1r, x2r)

xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
ym = model.decision_function(xm)

predictions = model.predict(X)
misclassified_indices = np.where(predictions != y)[0]


plt.scatter(X[predictions == y, 0], X[predictions == y, 1],
c=y[predictions == y])
plt.scatter(X[misclassified_indices, 0], X[misclassified_indices, 1],
marker='x', c='green', label='Misclassified')

plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels=[-1, 0, 1])

plt.legend()
plt.show()
```

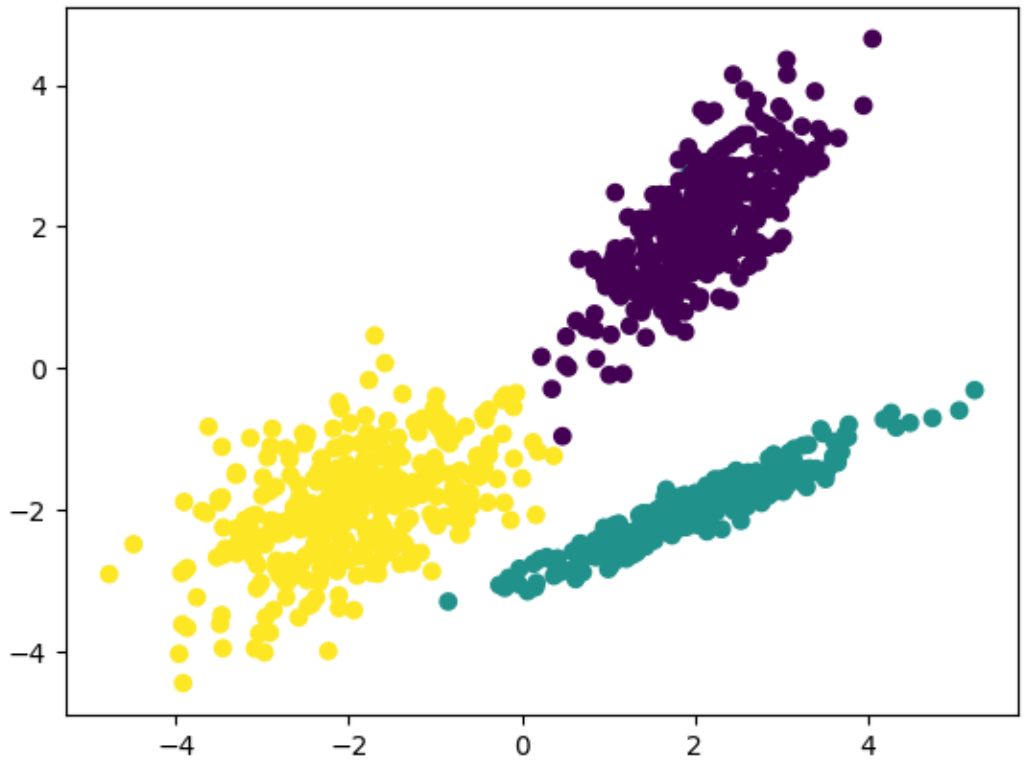


با تو هم رفتگی داده ها امکان جدا سازی و طبقه بندی آن ها سخت تر می شود. با این کار داده هایی که به اشتباه طبقه بندی شده اند افزایش یافته است و و نتایج برای داده های آموزش در روش logisticregression و sgd نیز به ترتیب از 0.9975 به 0.80375 و از 0.9975 به 0.79375 و برای داده های ارزیابی برای روش های ذکر شده از 0.995 به 0.82 و از 0.995 به 0.825 کاهش یافته است و دقت کلاسیفایرها کم شده است.

5.1

اگر با تعداد نمونه های یکسان، یک کلاس به داده ها اضافه شود مقدار داده های هر کلاس کمتر می شود و به طبع آن مقدار داده های بخش های testو train نیز کاهش می یابد و دقت شبکه کاهش خواهد یافت.

```
x, y=
make_classification(n_samples=1000,n_features=2,n_redundant=0,n_classes=3,
n_clusters_per_class=1,class_sep=2,random_state=83)









plt.scatter(x[:,0],x[:,1], c=[y])
plt.show
```




```
x_train,x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2,random_state=83)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
model = LogisticRegression(solver='sag', max_iter=200, random_state=83)
model.fit(x_train, y_train)
model.predict(x_test), y_test
model1.score(x_train, y_train)
```

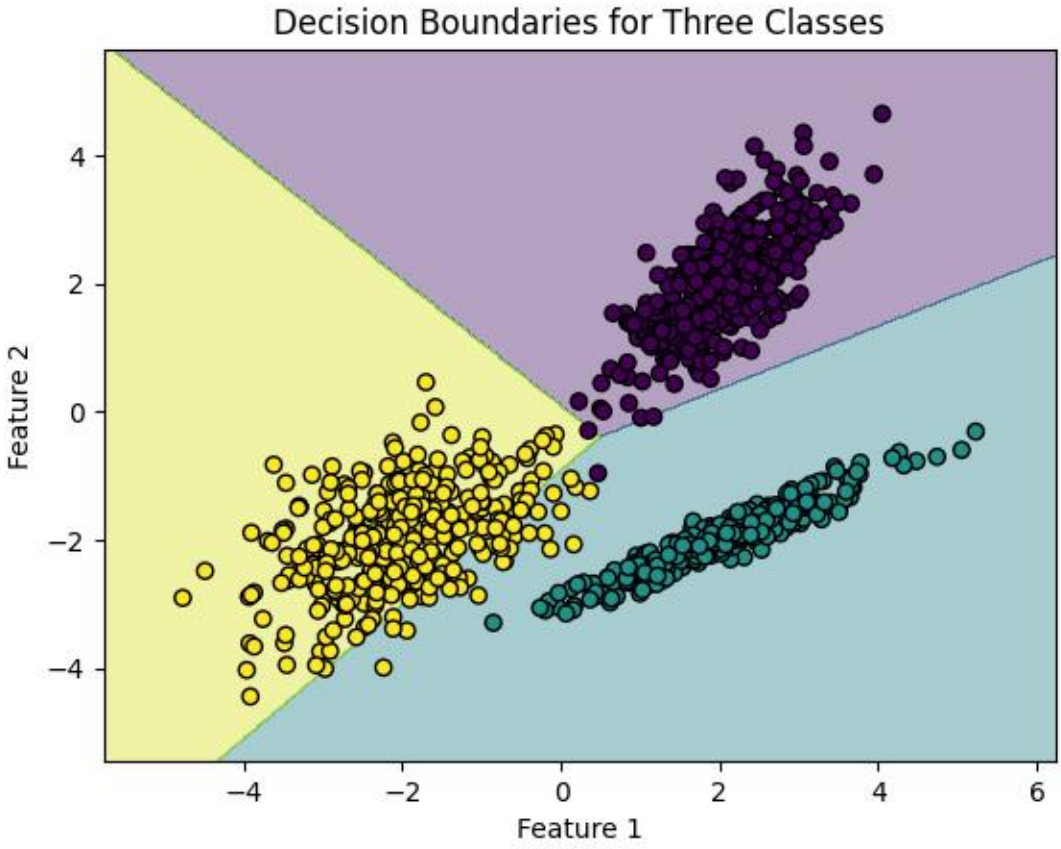
```
0.35875
```

```
model1.score(x_test, y_test)
```

```
0.33
```

```
x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max, 500),
                       np.linspace(x2_min, x2_max, 500))

# Predict class probabilities for each point in the meshgrid
Z = model.predict(np.c_[xx1.ravel(), xx2.ravel()])
Z = Z.reshape(xx1.shape)

# Plot the decision boundaries
plt.contourf(xx1, xx2, Z, alpha=0.4)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')  # Plot the data points
plt.xlabel('Feature 1')
```

```
plt.ylabel('Feature 2')
plt.title('Decision Boundaries for Three Classes')
plt.show()
```



1.2

این دیتاست شامل 1372 نمونه از اسکن‌های اسکناس‌های 50 دلاری آمریکا است. هر نمونه از اسکناس شامل 4 ویژگی است

واریانس تصویر :variance

کجی تصویر :skewness

چولگی تصویر :curtosis

انتروپی تصویر :entropy

هدف این دیتاست طبقه‌بندی اسکناس‌ها به دو دسته واقعی و جعلی است.

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1_oSSPAq90JDDHrFoGomoEJwCn4-1U8Fj

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data_banknote_authentication.txt', header=None)

a=['variance','skewness','curtosis','entropy','target']
df.columns=a



print(df)
```

چون دیتای ما دارای اسم برای ستون خود نیست، headerآن را برابر None می گذاریم سپس نام ستون ها را تعریف میکینم.

```
      variance  skewness  curtosis  entropy  target
0      3.62160   8.66610   -2.8073 -0.44699       0
1      4.54590   8.16740   -2.4586 -1.46210       0
2      3.86600  -2.63830    1.9242  0.10645       0
3      3.45660   9.52280   -4.0112 -3.59440       0
4      0.32924  -4.45520    4.5718 -0.98880       0
...        ...       ...       ...      ...     ...
1367   0.40614   1.34920   -1.4501 -0.55949       1
1368  -1.38870  -4.87730    6.4774  0.34179       1
1369  -3.75030 -13.45860   17.5932 -2.77710       1
1370  -3.56370  -8.38270   12.3930 -1.28230       1
1371  -2.54190  -0.65804    2.6842  1.19520       1

[1372 rows x 5 columns]
```

2.2

اهمیت بر زدن داده‌ها shuffling در یادگیری ماشین به این دلیل است که باعث می‌شود مدل یادگیری ماشین به طور برابر از همه داده‌ها یاد بگیرد. اگر داده‌ها بدون بر زدن استفاده شوند، ممکن است مدل یادگیری ماشین به طور ناخواسته از برخی از داده‌ها بیشتر از داده‌های دیگر یاد بگیرد. این می‌تواند باعث شود مدل

یادگیری ماشین دقیق نباشد و در برابر داده‌های جدید عملکرد خوبی نداشته باشد. بر زدن داده‌ها باعث می‌شود که داده‌ها به طور تصادفی مرتب شوند. این باعث می‌شود که مدل یادگیری ماشین به طور مساوی از همه داده‌ها یاد بگیرد و در برابر داده‌های جدید عملکرد بهتری داشته باشد.

این کار با کتابخانه آماده sickitlearn قابل انجام است و آن را مانند کد زیر فراخوانی می کنیم. نسبت تقسیم داده ها را 20% در نظر می گیریم که یعنی به نسبت 80% داده ی آموزش و 20% درصد داده ها، داده ی ارزیابی می شوند.

```
from sklearn.model_selection import train_test_split
x = df[['variance','skewness','curtosis','entropy']].values
y = df[['target']].values
x_train,x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
((1097, 4), (1097, 1), (275, 4), (275, 1))
```

3.2

```
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 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
```

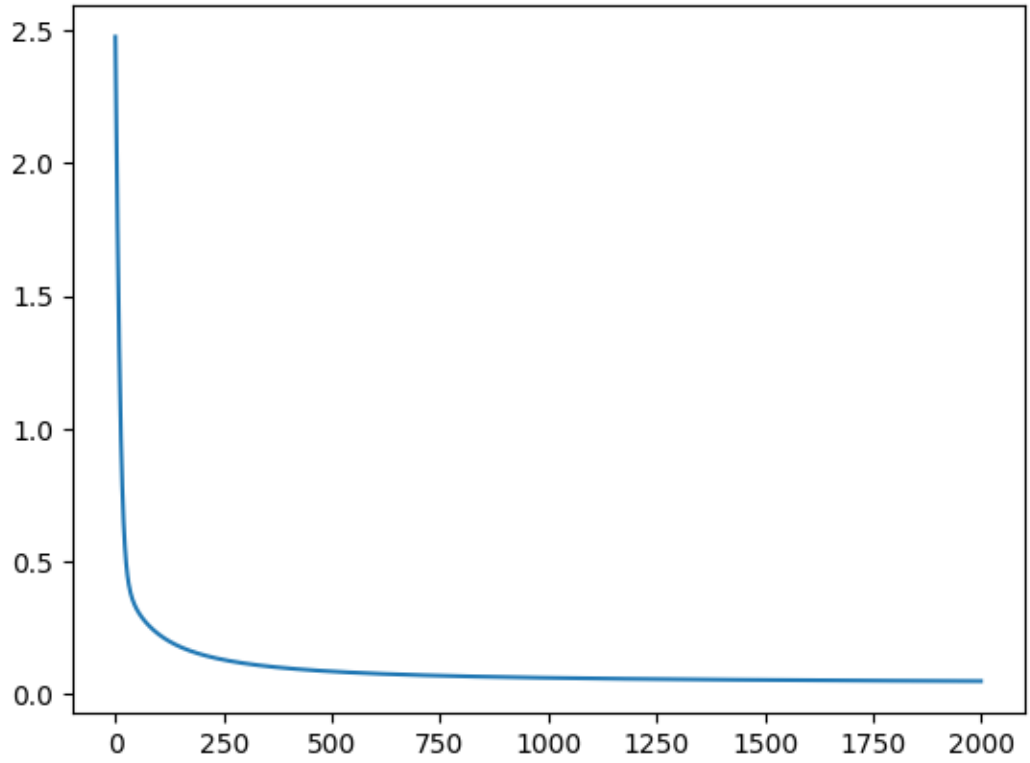
برای ایجاد hat-y ابتدا تابع sigmoid را تعریف می کنیم و سپس hat-y را ایجاد می کنیم. با استفاده از hat-y تابع logisticregression که ضرب ها xو ها wاست را تشکیل می دهیم و خروجی آن hat-yمی شود که همانy ای است که ما در شبکه خود ایجاد کرده ایم. سپس تابع اتلاف را با loss-log معرفی کرده و گرادیان آن را محاسبه می کنیم و eta که همان ضریب یادگیری است را در grads که حاصل ضرب x در اختلاف yاصلی ما با y بدست آمده است، ضرب کرده و w را هر سری با آن آپدیت می کنیم تا به w درست، برسیم.

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

x-train را به آرایه تبدیل می کنیم و یک ستون تماما یک به آن اضافه می کنیم تا بایاس را نیز در نظر بگیریم.

```
error_hist = []
eta=0.01
n_epochs=2000
w=np.random.randn(5,1 )
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)
```



w ما ماتریسی رندوم با تعداد سطرهای ویژگی+بایاس ما است 5 و تعداد ستون آن برابر عدد 1 می شود. ضریب آموزش را 0.01 و تعداد دورها را 2000 در نظر میگیریم.تابع اتلاف ما به شکل زیر تعریف می شود و به تعدادepoch های ما w را آپدیت می کند تا به مقدار نهایی و نزدیک ترین جواب برسیم hist-error خالی ای را تعریف می کنیم تا در هر بار که تابع اتلاف را محاسبه کردیم، مقدار آن را ذخیره کنیم.

نمودار تابع اتلاف یک ابزار بسیار مفید برای ارزیابی عملکرد یک مدل یادگیری ماشین در طول فرآیند آموزش است. با این حال، این نمودار به تنهایی نمی تواند با قطعیت کاملی در مورد عملکرد مدل نظر دهد.اگر مدل بیش از حد به داده های آموزشی عادت کند تابع اتلاف در داده های آموزشی ممکن است کاهش یابد، اما عملکرد بر روی داده

های تست ممکن است بهبود نیابد یا حتی بدتر شود. راه حل استفاده از مجموعه اعتبارسنجی برای ارزیابی عملکرد مدل در طول فرآیند آموزش است.

```
y_test=np.array(y_test)

y_test=y_test.reshape(-1,1)


y_test=y_test.reshape(-1,1)


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

train-yرا نیز به آرایه تبدیل می کنیم تا بتوانیم از آن استفاده کنیم و با استفاده از reshapeآن را به آرایه دو بعدی تبدیل می کنیم تا بتواند ضرب ماتریسی روی آن صورت گیرد

```
error_hist = []

w=[ 0.42983953 ,-1.4922111 , -0.79350524 ,-0.84238584 ,-0.34470033]
w=np.array(w)

w=w.reshape(-1,1)

y_hat = logistic_regression(x_test, w)
accuracy(y_test, y_hat)
```

```
0.9672727272727273
```

این بارw ها آپدیت نمی شوند و مقدار آخرین w بدست آمده در فرآیند آموزش را به عنوان ورودی برای داده های ارزیابی لحاظ می کنیم.

4.2

نرمال کردن داده ها برای منطقی کردن مقایسه پذیری و بهبود عملکرد الگوریتم های یادگیری ماشین صورت می گیرد. نرمال کردن داده ها باعث می شود تمام ویژگی ها به یک مقیاس یا بازه مشابه تبدیل شوند که قابل مقایسه تر و تفسیر پذیرتر باشند. مدل ممکن است به اندازه ی زیادی به داده های با ویژگی های بزرگتر (به دلیل مقیاس بزرگتر) وابستگی پیدا کند و از یادگیری الگوهای کلی دور شود.

روش اول نرمال کردن:

$$X_{normalized} = (X - X_{min})/(X_{max} - X_{min})$$

روش دوم نرمال کردن:

$$X_{normalixed} = (X - \mu)/ \sigma$$

```
df_normal=(df-df.min())/(df.max()-df.min())
df_normal
```

	variance	skewness	curtosis	entropy	target
0	0.769004	0.839643	0.106783	0.736628	0.0
1	0.835659	0.820982	0.121804	0.644326	0.0
2	0.786629	0.416648	0.310608	0.786951	0.0
3	0.757105	0.871699	0.054921	0.450440	0.0
4	0.531578	0.348662	0.424662	0.687362	0.0
...	...	...	...	...	...
1367	0.537124	0.565855	0.165249	0.726398	1.0

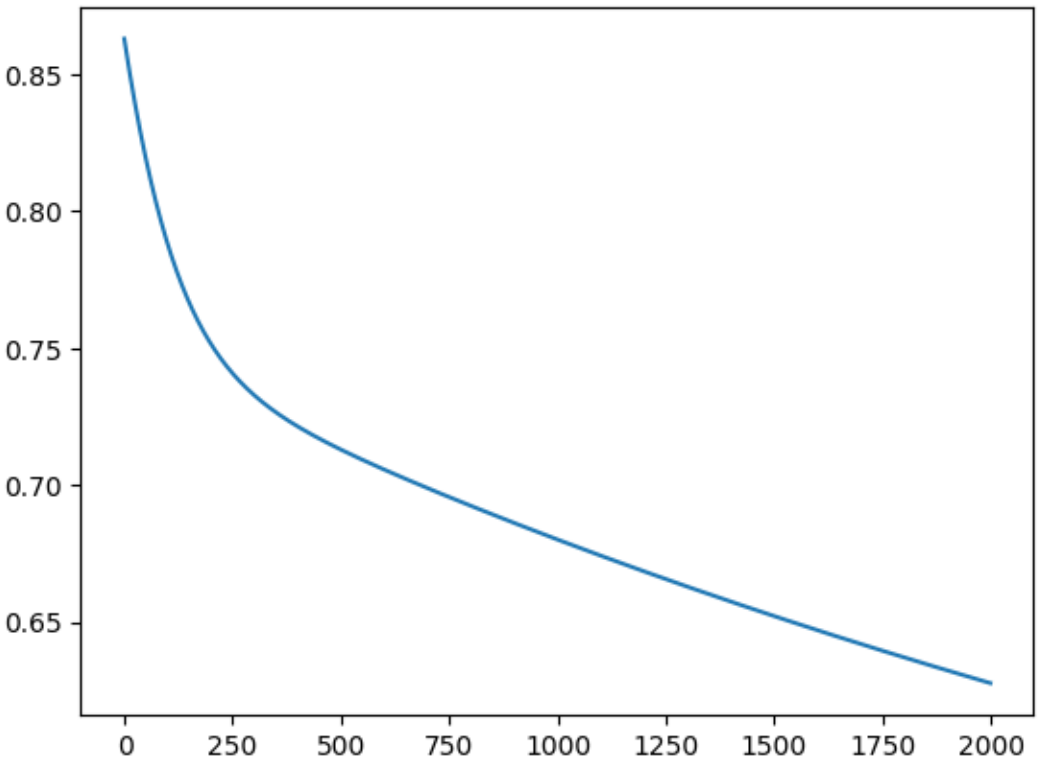
	variance	skewness	curtosis	entropy	target
1368	0.407690	0.332868	0.506753	0.808350	1.0
1369	0.237385	0.011768	0.985603	0.524755	1.0
1370	0.250842	0.201701	0.761587	0.660675	1.0
1371	0.324528	0.490747	0.343348	0.885949	1.0

1372 rows × 5 columns

4.2

```
x = df_normal[['x1','x2','x3','x4']].values
y = df_normal[['target']].values
x_train,x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
y_train=np.array(y_train)

y_train=y_train.reshape(-1,1)
x_train=np.asarray(x_train)
x_train = np.hstack((np.ones((len(x_train),1)), x_train))
x_train.shape
x = df_normal[['variance','skewness','curtosis','entropy']].values
y = df_normal[['target']].values
x_train,x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
y_train=np.array(y_train)
x_train=np.asarray(x_train)
x_train = np.hstack((np.ones((len(x_train),1)), x_train))
x_train.shape

y_train=y_train.reshape(-1,1)
error_hist = []
eta=0.01
n_epochs=2000
w=np.random.randn(5,1)
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)
```




```
      y_test=np.array(y_test)


y_test=y_test.reshape(-1,1)


y_test=y_test.reshape(-1,1)


x_test=np.asarray(x_test)
x_test = np.hstack((np.ones((len(x_test),1)), x_test))
x_test.shape
error_hist = []

w=[ 0.08691948, -1.60108064 ,-0.2467352 , -0.11823382 , 0.94108127]
w=np.array(w)
```

```
w=w.reshape(-1,1)

y_hat = logistic_regression(x_test, w)
accuracy(y_test, y_hat)
```

```
0.7781818181818182
```

بعد از نرمال سازی داده ها مقدار تابع اتلاف بیشتر شده است و در آخر به مقدار نزدیک 0.6 رسیده است که این با نتایج قبل از نرمال سازی داده ها متفاوت است. همچنین مقدار ارزیابی دقت داده های تست به مقدار 0.7090 درصد رسیده است که این عدد نیز کمتر از دقت حالت قبل می باشد .

```
y_hat[:5]
```

```
array([[0.42064783], [0.55665216], [0.52027776], [0.67938884],
       [0.47406966]]
```

6.2

```
a=df[df['target']==1]
b=df[df['target']==0]
len(a)
```

```
610
```

```
len(b)
```

```
762
```

برای فهمیدن تعداد داده ها برای هر کلاس مختلف، تعداد آن را با روش زیر میشماریم و متوجه می شویم که تعداد داده هایی که کلاس آن ها ۱ است با تعداد داده هایی که۹ کلاس آن ها ۰ است، یکسان نمی باشد.

مدل هایی که با داده های نامتوازن آموزش داده شده اند، ممکن است تمایل به پیش بینی کلاس اکثریت داشته باشند و در تشخیص کلاس های کمتری دچار مشکل شوند. وجود تعداد نامتوازن نمونه ها می تواند باعث شود که الگوهای کمتر مشاهده شوند و در نتیجه توانایی مدل در تشخیص و یادگیری این الگوها کاهش یابد. در صورتی که مدل تنها با داده های کلاس اکثریت آموزش ببیند، احتمال بروز overfitting به داده های این کلاس بیشتر است.برای حل این موضوع اگر تعداد داده های ما زیاد بود، می توانیم تعداد داده های کلاس بیشتر را کم کنیم تا تعداد یکسانی داشته باشند. روش دیگری برای درست کردن این موضوع، ایجاد داده ی فیک است. این کار را در این پروژه با میانگین گیری از دو سطر و ایجاد سطر جدید انجام دادیم .دیتا فریم جدید و خالی ای به اسم row-new ایجاد می کنیم و به تعداد اختلاف a و ، bبا استفاده از میانگین سطرهای بالایی و پایینی در a داده ی جدید ایجاد کرده target تمام داده های تولید شده را برابر ۱ قرار می دهیم و در نهایت تمامی داده های تولید شده ی جدید را در row-new قرار داده و آن را با a ابتدایی، مخلوط می کنیم.

```
new_rows = pd.DataFrame()
for i in range(len(b)-len(a)):

    v= a.iloc[i:i+2, :-1].mean()

    new_row = v.append(pd.Series({'target': 1}))
    new_rows = new_rows.append(new_row , ignore_index=True)


a.reset_index(drop=True, inplace=True)
new_rows.reset_index(drop=True, inplace=True)

updated_df = pd.concat([a.reset_index(drop=True), new_rows],
ignore_index=True)

updated_df
```

	variance	skewness	curtosis	entropy	target
0	-1.39710	3.31910	-1.392700	-1.994800	1.0
1	0.39012	-0.14279	-0.031994	0.350840	1.0
2	-1.66770	-7.15350	7.892900	0.967650	1.0
3	-3.84830	-12.80470	15.682400	-1.281000	1.0
4	-3.56810	-8.21300	10.083000	0.967650	1.0
...	...	...	...	...	...
757	-0.58961	-6.37235	4.711900	-0.308000	1.0
758	1.29199	-0.80495	-0.653650	0.551902	1.0
759	1.62945	2.94145	-3.827800	-1.954748	1.0
760	-1.49850	5.15040	-2.242640	-4.729100	1.0
761	-3.37975	2.74799	1.759210	-2.771995	1.0

762 rows × 5 columns

```
combined_df = updated_df.append(b)

combined_df
```

	variance	skewness	curtosis	entropy	target
0	-1.39710	3.31910	-1.392700	-1.99480	1.0
1	0.39012	-0.14279	-0.031994	0.35084	1.0
2	-1.66770	-7.15350	7.892900	0.96765	1.0
3	-3.84830	-12.80470	15.682400	-1.28100	1.0
4	-3.56810	-8.21300	10.083000	0.96765	1.0
...	...	...	...	...	...
757	2.66060	3.16810	1.961900	0.18662	0.0
758	3.93100	1.85410	-0.023425	1.23140	0.0
759	0.01727	8.69300	1.398900	-3.96680	0.0
760	3.24140	0.40971	1.401500	1.19520	0.0

variance	skewness	curtosis	entropy	target	
761	2.25040	3.57570	0.352730	0.28360	0.0

1524 rows × 5 columns

7.2

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
x_train,x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2,random_state=83)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
model = LogisticRegression(solver='sag',max_iter=100,
random_state=83,class_weight='balanced')
model.fit(x_train, y_train)
model.predict(x_test), y_test
model.score(x_train, y_train)
```

```
0.9699179580674567
```

```
model.score(x_test, y_test)
```

```
0.9709090909090909
```

برای چالش عدم تعادل داده ها از در LogisticRegression ارگومان class_weight='balanced' را اضافه میکنیم. با این دستور مدل به کلاس هایی که تعداد کمتری دارند بیشتر توجه میکند.

1.3

دیتاست "Heart Disease Indicators" شامل اطلاعاتی است که مربوط به شاخص های بیماری قلبی است. این دیتاست شامل ویژگی هایی است که از بررسی های پزشکی برداشت شده اند و هدف از آن تشخیص این است که آیا یک فرد بیماری قلبی دارد یا خیر

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1h-jnsobVt0lCsOySuq5N7xeaaFRtZaWP
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('/content/heart_disease_health_indicators.csv')
```

2.3

ستون اول HeartDiseaseorAttack که باید ستون اخر باشد پس ابتدا ان را درست میکنیم.

```
first_column = df.pop('HeartDiseaseorAttack')
df.insert(len(df.columns), 'HeartDiseaseorAttack', first_column)
df
```

	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	Veggies	...	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income	HeartDiseaseorAttack
0	1	1	1	40	1	0	0	0	0	1	...	0	5	18	15	1	0	9	4	3	0
1	0	0	0	25	1	0	0	1	0	0	...	1	3	0	0	0	0	7	6	1	0
2	1	1	1	28	0	0	0	0	1	0	...	1	5	30	30	1	0	9	4	8	0
3	1	0	1	27	0	0	0	1	1	1	...	0	2	0	0	0	0	11	3	6	0
4	1	1	1	24	0	0	0	1	1	1	...	0	2	3	0	0	0	11	5	4	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
25	0	0	1	25	0	0	0	1	1	1	...	0	1	0	0	0	0	4	6	8	0

	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	Veggies	...	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income	HeartDiseaseorAttack
36556																					
253657	0	1	1	24	0	0	0	0	0	1	...	0	3	0	0	0	0	7	5	3	0
253658	0	0	0	27	0	0	0	1	0	0	...	1	2	0	0	0	0	3	6	5	0
253659	0	1	1	37	0	0	2	0	0	1	...	0	4	0	0	0	0	6	4	1	0
253660	0	1	1	34	1	0	0	0	1	1	...	0	3	0	2	1	0	7	4	3	0

253661 rows × 22 columns

```
class_1 = df.loc[df['HeartDiseaseorAttack'] == 1]
class_0 = df.loc[df['HeartDiseaseorAttack'] == 0]
df_1 = class_1.sample(100)
df_0 = class_0.sample(100)
new_data = pd.concat([df_0, df_1])
```

3.3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
x_train,x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=83)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
from sklearn.linear_model import LogisticRegression, SGDClassifier
model = LogisticRegression(solver='sag', max_iter=3000, random_state=83)
model.fit(x_train, y_train)
model.predict(x_test), y_test
print(model.score(x_train, y_train))
print(model.score(x_test, y_test))
```

```
0.86875
0.775
```

```
model1 = SGDClassifier(loss= 'log_loss', random_state=83)
model1.fit(x_train, y_train)
model.predict(x_test), y_test
```

```
0.86875
```

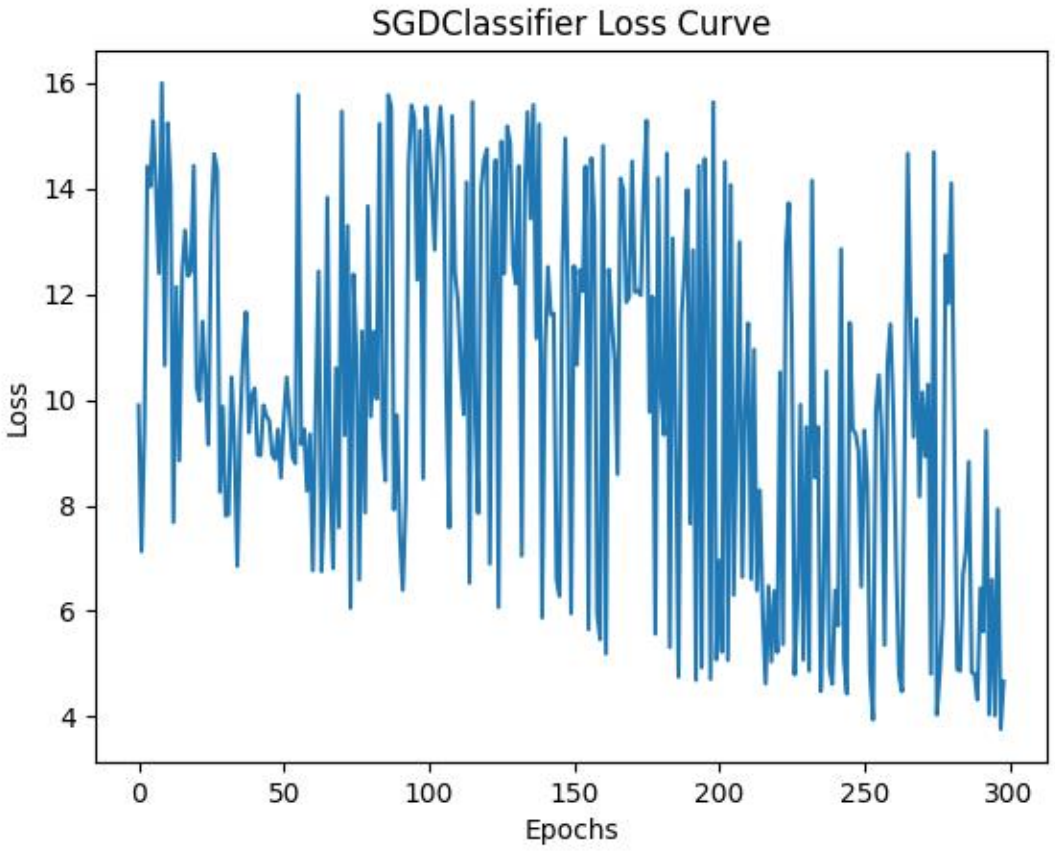
```
0.775
```

4.3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss
x_train,x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2,random_state=83)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
model1 = SGDClassifier(loss='log_loss', random_state=83)
model1.fit(x_train, y_train)

model1_proba = model1.predict_proba(x_train)

losses = []
for epoch in range(1, 300):
    model1.partial_fit(x_train, y_train, classes=np.unique(y_train))
```

```
    epoch_loss = log_loss(y_train, model1.predict_proba(x_train))
    losses.append(epoch_loss)

plt.plot(losses)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('SGDClassifier Loss Curve')
plt.show()
```

در هر دوره اموزش خطای لگاریتمی محاسبه و ذخیره میشود. در نهایت هر خطا متناسب با دوره ان رسم میشود.



هر چه دوره اموزشی بیشتر میشود خطا داده های اموزشی کاهش میابد.

شاخصهٔ ارزیابی جدید:

Confusion Matrixیک جدول است که عملکرد یک مدل طبقه‌بندی را در طبقه‌بندی نمونه‌ها نمایش می‌دهد. این جدول، تعداد نمونه‌های صحیح و نادرست طبقه‌بندی شده را برای هر کلاس خروجی نشان می‌دهد.در این جدول، TPتعداد نمونه‌های مثبت واقعی که به درستی طبقه‌بندی شده‌اند را نشان می‌دهد .FN تعداد نمونه‌های مثبت واقعی که به اشتباه طبقه‌بندی شده‌اند را نشان می‌دهد .FP تعداد نمونه‌های منفی واقعی که به اشتباه طبقه‌بندی شده‌اند را نشان می‌دهد .TN تعداد نمونه‌های منفی واقعی که به درستی طبقه‌بندی شده‌اند را نشان می‌دهد.

```
Confusion Matrix = | True Positive (TP) | False Negative (FN) |
                   | False Positive (FP) | True Negative (TN) |
```

```
from sklearn.metrics import confusion_matrix
y_perd=model.predict(x_test)
# Calculate the confusion matrix
confusion_matrix = confusion_matrix(y_test, y_perd)

# Print the confusion matrix
print(confusion_matrix)
```

```
[[14  2]
 [ 7 17]]
```

خروجی کد 40 داده ارزیابی را به فرم Confusion Matrix میدهد. یعنی 14داده‌TP و 2داده‌FN و 7داده‌FP و 17اده‌TN هستند.