گزارش کار

شناسایی آماری الگو

تمرین ۲

آرمین خیاطی
9931153

سبحان نامی

9831767

1399/10/2

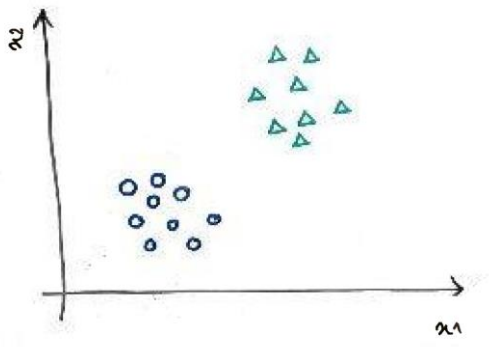
هدف

در این تمرین قصد داریم ابتدا مدل‌های Logistic Regression و Softmax Regression را بر روی دیتاست iris بصورت Multiclass پیاده سازی کنیم. در مدل Logistic Regression از روش‌های one-vs-one و one-vs-all برای Classification استفاده می‌کنیم و Accuracy هر یک از روش‌ها را بر روی داده‌های Train و Test محاسبه می‌کنیم. درصد داده‌های Train به Test بصورت ۸۰ به ۲۰ می‌باشد. در بخش دوم از مدل Bayesian Classification برای داده‌های قرار داده شده در فایل تمرین که بر اساس توزیع گوسی بدست آمده‌اند استفاده و سپس مقدار Accuracy را برای هر کدام از دیتاست‌ها محاسبه می‌کنیم.

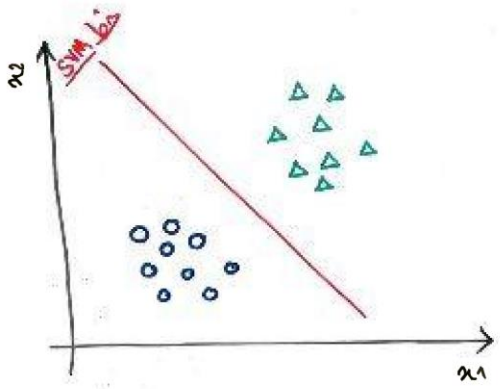
روش «یک در مقابل همه (One vs. All)» برای طبقه بندی داده‌های چند کلاسه

بسیاری از مسائل حوزه‌ی طبقه‌بندی (Classification) فقط دو کلاس (دو نوع برچسب) دارند. به این مسائل، طبقه‌بندی دودویی می‌گویند. برای مثال، فرض کنید می‌خواهیم سیستمی بسازیم که بتواند تفاوت دو کلاس متفاوت y1,y2 را بر اساس یک سری ویژگی (بُعد)، تشخیص دهد. این کار توسط الگوریتم‌های طبقه بندی به سادگی قابل انجام است. اما هنگامی که تعداد این طبقه‌ها (انواع برچسب‌ها) بالا و بالاتر می‌رود، کار برای الگوریتم سخت شده نیاز به الگوریتم‌های پیچیده‌تری هست.

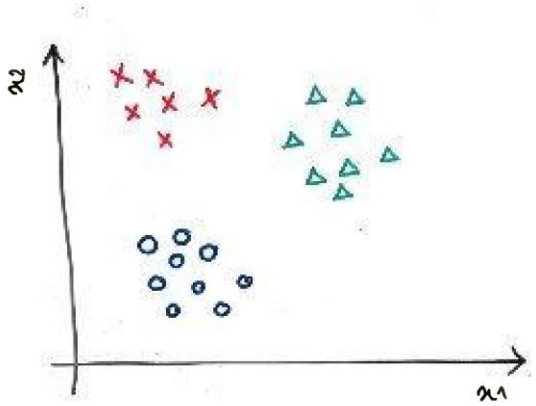
ما می‌توانیم داده‌ها را به فضای چند بُعدی جبر خطی نگاشت کنیم. حال فرض کنید می‌خواهیم کلاس‌های y1,y2 را با استفاده از فقط دو ویژگی x1,x2 از هم تفکیک کنیم. تصویر زیر، نگاشت شده‌ی داده‌ای است بر روی محور مختصات:



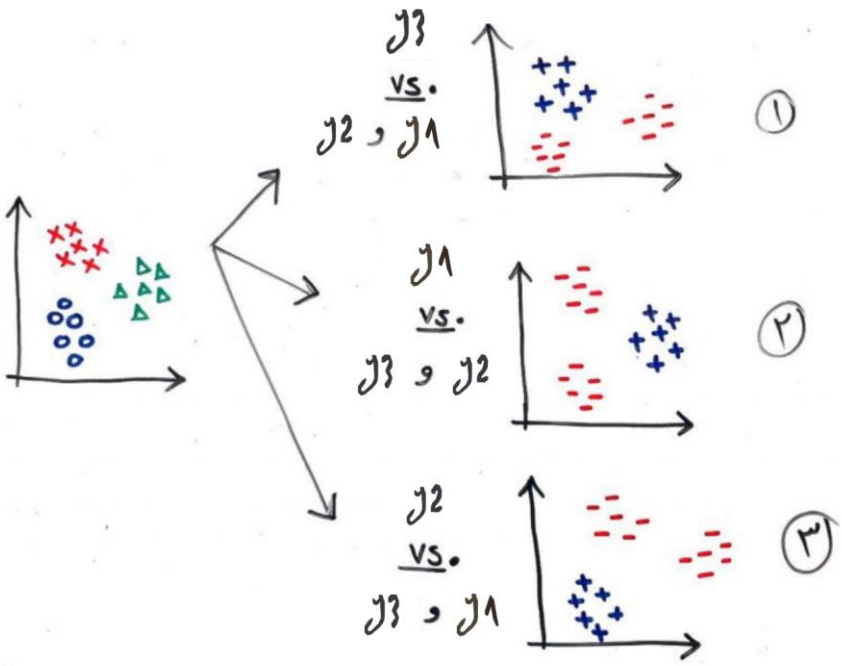
در این مسئله‌ی فرضی، ما دو ویژگی داریم (x1,x2) و نمونه‌های ما در این فضای دو بُعدی قرار دارند. نمونه‌های مثلثی کلاس y1 هستند و نمونه‌های دایره‌ای کلاس y2 . در این حالت می‌توانیم با استفاده از الگوریتمی مانند SVM، خطی را رسم کنیم که می‌تواند تمایز بین کلاس‌های y1,y2 را مشخص نماید.



حال فرض کنید، تعداد طبقه‌ها (انواع برچسب‌ها) بیشتر شود. برای مثال می‌خواهیم هر کلاس را به طبقه‌هایی مانند **y1 , y2 , y3** طبقه بندی کنیم. در این مسئله تعداد طبقه‌ها برابر ۳ است و تصویر زیر، نگاشت شده‌ی این داده‌ها بر روی محور مختصات مثال قبلی است:



فرض کنید نمونه‌های قرمز که با علامت ضربدر مشخص شده‌اند، کلاس‌های هستند که مربوط به طبقه‌ی **y3** است. نمونه‌های مثلثی و نمونه‌های دایره‌ای هم مانند قبل به ترتیب، کلاس‌های 1y و 2y هستند. همان‌طور که مشاهده می‌کنید، دیگر نمی‌توان با یک خط ساده، تمایزِ طبقه‌ها را کشف کرد. برای این‌کار بایستی از تکنیک‌های دیگری مانند تکنیک kernel، یا الگوریتم‌های طبقه بندی غیر خطی استفاده کرد. اما یک راه حل دیگر هم موجود است که در بسیاری از شرایط، دقت خوب و حتی بهتر از برخی از الگوریتم‌های پیشرفته‌ی طبقه بندی تولید می‌کند. این راه حل «یک در مقابل همه یا **One vs. All**» نام دارد. البته به این راه حل «یک در مقابل بقیه یا **One vs. Rest**» نیز می‌گویند. در این روش بایستی به تعداد طبقه‌ها (انواع برچسب‌ها)، مُدلِ طبقه بندی ساخت، و اعضای هر یک از کلاس‌ها را در مقابل بقیه قرار داد. با این کار می‌توان یک مسئله‌ی طبقه بندی چند کلاسه (Multi Class) را به چندین مسئله‌ی طبقه بندی دودویی (Binary) تبدیل کرد. کاری مانند شکل زیر:



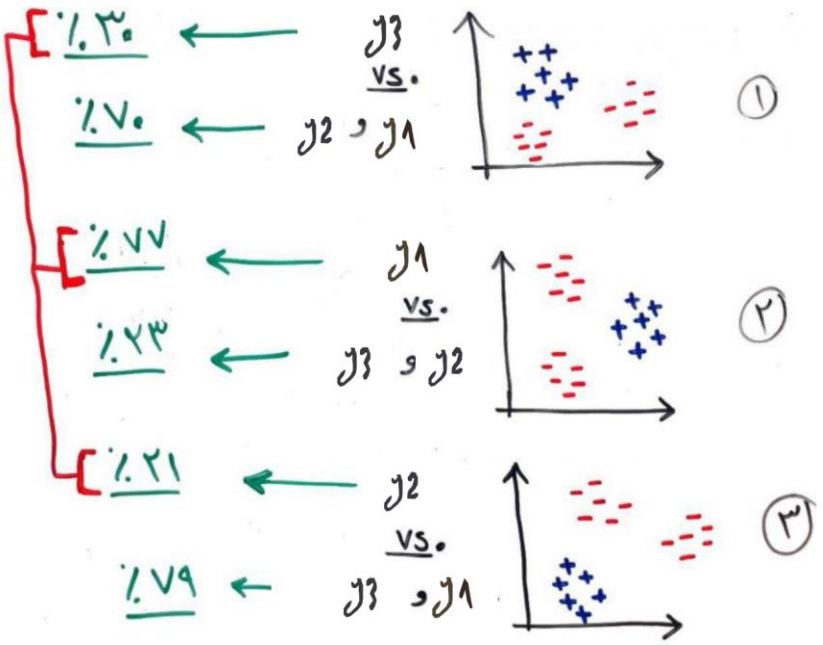
در این‌جا سه کلاس داریم (y , y2 , 3y1). پس نیاز به سه مجموعه‌ی داده به صورتی داریم که در هر یک، نمونه‌های یکی از کلاس‌ها در مقابل نمونه‌های بقیه‌ی کلاس‌ها قرار بگیرد (در مثال بالا، کلاس مورد نظر با علامت مثبت (+) و کلاس‌های بقیه که در مقابل آن قرار می‌گیرند با علامت منفی (-) نمایش داده شده اند). حال برای هر کدام از این مجموعه داده‌ها بایستی یک الگوریتم طبقه بندی دودویی طراحی کنیم:

۱. طبقه بندی دودویی «کلاس ایمیل 3y در مقابل کلاس‌های 1y و 2y»

۲. طبقه بندی دودویی «کلاس ایمیل 1y در مقابل کلاس‌های 2y و 3y»

۳. طبقه بندی دودویی «کلاس ایمیل 2y در مقابل کلاس‌های 1y و 3y»

این سه طبقه بندِ دودویی بر روی داده‌ها، یادگیری را انجام می‌دهند و بعد از یادگیری، یک نمونه‌ی جدید برای پیش‌بینیِ کلاس، به تمامیِ این ۳ الگوریتم که یادگرفته شده‌اند، داده می‌شود و هر کدام احتمال عضویت این نمونه را به یکی از دو کلاس خود، برمی‌گردانند. برای مثال فرض کنید ما سه الگوریتمِ رگرسیون لجستیک (Logistic Regression) را بر روی داده‌های بالا با استفاده از مدل «یک در مقابل همه (One vs. All)» اعمال کرده‌ایم و این سه الگوریتم یادگیری را انجام داده‌اند. حال یک نمونه‌ی جدید به این سه الگوریتم داده می‌شود و نتایج به صورت زیر است:



همان‌طور که می‌بینید احتمالِ کلاس 1y بیشتر از بقیه‌ی احتمال‌ها شد، پس این نمونه، یک نمونه کلاس 1y است. توجه کنید که برای مقایسه بایستی فقط طبقه‌های مثبت شده را با هم مقایسه کرد تا به یک نتیجه‌ی نهایی رسید.

روش «یک در مقابل یک (One vs. One)» در طبقه بندی

یکی از روش‌های طبقه بندی (Classification) داده‌های چند کلاسه، استفاده از روشِ «یک در مقابل همه» یا همان One vs. All است. اما این روش، در برخی از مواقع ضعف‌هایی نیز دارد. به همین دلیل روش «یک در مقابل یک» یا همان **One vs. One** به وجود آمد که در بسیاری از مواقع، کیفیتِ به مراتب بهتری، نسبت به روش قبلی (One vs. All) ارائه می‌دهد.

فرض کنید، مجموعه‌ی داده‌ی شما، ۱۰هزار نمونه داشته باشد. این ۱۰هزار نمونه در ۲۰ طبقه (کلاس) تقسیم بندی شده‌اند و شما به دنبال الگوریتمی هستید که بتواند بر روی این داده‌ها، طبقه بندی (Classification) را انجام دهد. اگر مانند قبل، از روش «یک در مقابل همه» استفاده کنید، نیاز به ۲۰ مجموعه‌ی داده دارید، تا یکی یکی، هر کدام از نمونه‌های یک طبقه را در مقابل نمونه‌های بقیه‌ی طبقه‌ها قرار دهید. در واقع در هر مجموعه‌ی داده، ۵۰۰ نمونه از یک کلاس در مقابل ۹۵۰۰ نمونه از کلاس‌های دیگر قرار می‌گیرند و یک الگوریتم طبقه بندی دودویی، بایستی بتواند طبقه بندی را برای این دو کلاس انجام دهد. به این حالت که تعداد نمونه‌های داخل یک کلاس خیلی کمتر از نمونه‌های یک کلاس دیگر باشد، حالت نامتوازن (imbalance) در طبقه بندی گفته می‌شود. این عدمِ توازن، باعث می‌شود کیفیت الگوریتم طبقه بندی دودویی کم شوند و در نهایت الگوریتمِ نهایی که همان الگوریتم «یک در مقابل همه» است، نتواند دقت و کیفیت مورد نظر را فراهم کند. در این دست از مسائل، می‌توان به الگوریتم «یک در مقابل یک (One vs. One)» رجوع کرد. این الگوریتم همانند الگوریتم «یک در مقابل همه (One vs. All)» عمل می‌کند با این تفاوت که در این الگوریتم، ما مجموعه داده‌ی کل را طوری تقسیم بندی می‌کنیم که نمونه‌های هر جفت از کلاس‌ها در مقابل یکدیگر قرار بگیرند. در واقع تقسیم‌بندی به گونه‌ای انجام می‌شود که در هر مجموعه‌ی داده‌ی تقسیم شده، نمونه‌های یک

طبقه در مقابل نمونه‌های طبقه‌ی دیگر قرار داشته باشند. در مثال قبل می‌خواستیم کلاس‌ها را به سه طبقه‌ی **y1 , y2 , y3** طبقه بندی کنیم. در روشِ «یک در مقابل یک (One vs. One)» نیاز به سه مجموعه‌ی داده جدا داریم:

۱. مجموعه‌ی داده‌ای که در آن کلاس‌های **y1** در مقابل کلاس‌های **y2** باشند

۲. مجموعه‌ی داده‌ای که در آن کلاس‌های **y1** در مقابل کلاس‌های **y3** باشند

۳. مجموعه‌ی داده‌ای که در آن کلاس‌های **y2** در مقابل کلاس‌های **y3** باشند

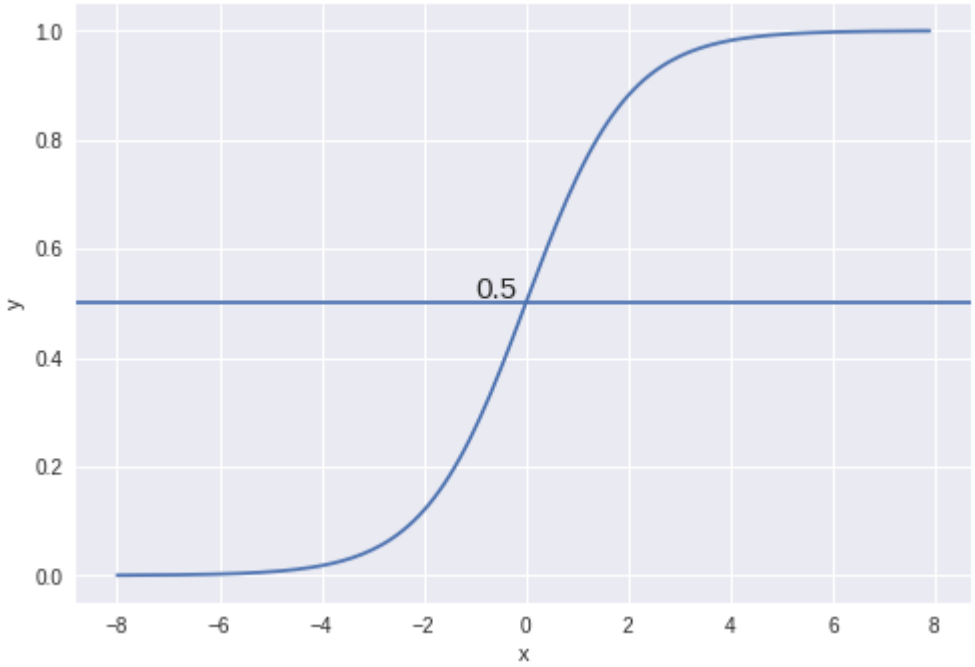
حال مانند قبل با استفاده سه الگوریتمِ طبقه بندی دودویی مانند رگرسیون لجستیک ((Logistic Regression، مدل‌هایمان را می‌سازیم. سپس بعد از یادگیری، نمونه‌ی جدید مانند قبل به الگوریتم‌های یادگرفته شده داده می‌شوند، و آن طبقه‌ای که بیشترین احتمال را در بین طبقه‌ها پیدا کند، به عنوان طبقه‌ی نهایی انتخاب می‌شود. روشِ «یک در مقابل یک» به تعداد بیشتری زیر مجموعه داده نیاز دارد. برای مثال اگر داده‌های شما در N طبقه برچسب خورده باشند، نیاز به N)*N-1)/2 زیر مجموعه‌ای هست که به صورت جفتی، نمونه‌های هر کلاس را در مقابل نمونه‌های کلاس دیگر قرار دهد. ولی در روش «یک در مقابل همه» فقط به N زیر مجموعه داده نیاز داریم. پس برای مثال اگر داده‌های ما در ۲۰ طبقه برچسب خورده باشد، در روش «یک در مقابل یک» نیاز به ۱۹۰ زیر مجموعه داده داریم در حالی که در روش قبلی (یک در مقابل همه) فقط به ۲۰ زیر مجموعه داده نیاز داشتیم. طبیعتاً برای هر کدام از مسائل در دنیای واقعی با توجه به دقت و سرعت مورد نیاز، می‌توان از یکی از این روش‌ها استفاده کرد.

تابع فعال ساز

به تابعی که هر مقدار عدد حقیقی را به صفر و یک نگاشت میکند تابع فعال ساز یا Activation Function می‌گویند. هر تابع فعال سازی که این خصوصیت را داشته باشد قابل استفاده است.

این توابع که ورودیِ آن‌ها یک عدد (کوچک یا بزرگ در بازه‌ی دلخواه) است و خروجیِ آن‌ها معمولاً یک عدد بین ۰ و ۱، یا ۱- و ۱+ است. در واقع این توابع یک عددِ ورودی را به یک بازه مشخص (مثلا ۱- تا ۱+) تبدیل می‌کنند. به توابعِ فعال‌ساز، توابع انتقال ((transfer function نیز می‌گویند. ما در این تمرین از تابع sigmoid و softmax که در ادامه توضیح داده می‌شود استفاده می‌کنیم.

Sigmoid



همانطور که میبینید مقدار این تابع همیشه بین صفر و یک و در X = 0 برابر با 0.5 است. بنابراین می‌توان سر حد احتمال برای هر دسته را 0.5 در نظر گرفت یعنی اگر احتمال ورودی ها کمتر از نیم شد مربوط به کلاس صفر و اگر بیشتر از نیم شد مربوط به کلاس یک است.

Softmax

تابع softmax زمانی استفاده می‌شود که قصد داریم از روش‌های طبقه‌بندی چند کلاسی مختلف استفاده کنیم. فرمول زیر تابع softmax را نشان می‌دهد. Softmax در هسته اصلی خود اجازه می‌دهد مقادیر کمتر از شاخص خود را حذف کرده (یا به عبارت دقیق‌تر سرکوب) و مقادیر بالاتر از میانگین خود را برجسته‌تر کنیم.

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}}$$

دلیل استفاده از تابع softmax این است که خروجی تابع فوق به ما اجازه می‌دهد یک توزیع احتمال را روی نتایج مختلف و متنوع انجام دهیم. تابع فوق پویایی زیادی در اختیار ما قرار می‌دهد، زیرا اجازه می‌دهد برخی از مسائل پیچیده را با اتکا بر طبقه‌بندی چند کلاسی حل کنیم.

در نتیجه تابع softmax گزینه ایده‌آلی برای به‌کارگیری روی مجموعه داده Iris است. برای کاربردهایی همچون طبقه‌بندی‌های باینری، باید از توابعی همچون Sigmoid استفاده کنیم، زیرا تابع فوق نسخه خاصی از تابع softmax است که تعداد کلاس‌ها را به دو مورد تقلیل می‌دهد.

دیتاست

برای این تمرین ما از دیتاست IRIS استفاده خواهیم کرد. در این تمرین از روش‌های one-vs-one و one-vs-all که توضیح داده شده برای جداسازی داده‌های دیتاست استفاده کرده‌ایم.

ردی ف	ویژگی	توضیحات
1	sepal length	طول کاسبرگ
2	sepal width	عرض‌کاسبرگ
3	petal length	طول‌گلبرگ
4	petal width	عرض‌گلبرگ
5	class	نوع گیاه ● Iris-setosa ● Iris-versicolor ● Iris-virginica

بخش اول

در این بخش Logistic Regression را با استفاده از روش‌های one-vs-one و one-vs-all و Softmax Regression را پیاده سازی می‌کنیم و میزان Accuracy آنها را روی دیتاست iris امتحان می‌کنیم.

پیاده سازی

در این تمرین از زبان پایتون و کتابخانه‌های Sklearn، Pandas، Numpy, seaborn, و Matplotlib برای خواندن فایل ها، انجام پیش پردازش ها و محاسبات و رسم نمودار ها استفاده شده است. کد این بخش از تمرین در فایل Logistic_one_vs_all.ipynb ، Logistic_one_vs_one.ipynb و Logistic_softmax.ipynb موجود است. در ادامه به تشریح توابع نوشته شده می پردازیم.

Logistic Regression به روش one-vs-all

Load Datasets

```
def raw_data():
 data = pd.read_csv('content/iris.data',names=['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal
Width', 'iris_class'])
 data["iris_class"].replace({"Iris-setosa": 0., "Iris-virginica": 1., "Iris-versicolor": 2.},
inplace=True)
 X_train, X_test, y_train, y_test = train_test_split(data.iloc[ : , 0:-1], data.iloc[ : , -1],
stratify=data['iris_class'], test_size=0.2, random_state=2020)
 return X_train, X_test, y_train, y_test
```

این تابع Iris.Data را بوسیله کتاب خانه Pandas درون برنامه وارد و داده ها را به دسته های Train و Test به کمک کتابخانه Sklearn و با نرخ 80 به 20 بر میگرداند.

prepared_data

```
def prepared_data():
 X_train, X_test, y_train, y_test = raw_data()
 X_train, X_test = addbias(X_train), addbias(X_test)
 y_train, y_test = relabel_data(y_train), relabel_data(y_test)
 return X_train, X_test, y_train, y_test
```

این تابع مقادیر دیتاست را برای روش one-vs-all آماده سازی می‌کند.

init_weights

```
def init_weights(shape):
 return np.zeros(shape, dtype='float64')
```

این تابع وزن های با مقادیر صفر برای شروع الگوریتم، تولید می‌کند.

relabel_data

```
def relabel_data(y):
   label = list(set(y))
   relabeled_data = np.zeros(len(y)*len(label)).reshape(len(y),len(label))
   for i in range(len(label)):
       relabeled_data[y==label[i],i] = 1
   return relabeled_data
```

برای تغییر در label های داده‌ها از این تابع استفاده می‌شود.

addbias

```
def addbias(x):
   return np.concatenate((np.ones((len(x))).reshape(-1,1), x),axis = 1)
```

Sigmoid

```
def sigmoid(x):
```

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

تابع سیگموییدی که توضیح داده شد را در اینجا پیاده سازی کرده ایم.

compute_loss

```
def compute_loss(y_, y):
 return -1/y.size * np.sum(y * np.log(y_) + (1 - y) * np.log(1 - y_), axis=0)
```

این تابع نیز برای محاسبه Cost طبق همان فرمول ارائه شده استفاده می شود.

gradient_dsc

```
def gradient_dsc(X, y, y_):
 return np.dot(X.T, (y_ - y)) / y.size
```

برای محاسبه gradient می‌باشد.

update_weight

```
def update_weights(w, lr, grad):
 return w - lr * grad.T
```

Mse

```
def mse(y_, y):
 diff = np.subtract(y_, y)
 ms = np.power(diff, 2, dtype='float64')
 return np.mean(ms)
```

این تابع برای محاسبه Mean Square Error بر روی مقادیر پیش بینی شده و مقادیر واقعی استفاده میشود.

Predict

```
def predict(X, weights):
   z = np.dot(X, weights.T)
   return sigmoid(z)
```

این تابع برای پیشبینی روی مقادیر جدید استفاده میشود و مطابق با فرمول Decision Boundary پیاده سازی شده است.

predict_label

```
def predict_label(X, weights):
```

```
 a = predict(X, weights)
 return np.argmax(a, axis=1)
```

Train

```
def train(X_train, y_train, lr, epochs, weights):
   for i in range(epochs):
       xw = np.dot(X_train, weights.T)
       prob = sigmoid(xw)
       grad = gradient_dsc(X_train, y_train, prob)
       weights = update_weights(weights, lr, grad)
       loss = compute_loss(prob, y_train)
       loss_log.append(loss)
   return weights
```

در این تابع بر اساس فرمول های شرح داده شده، مدل آموزش داده می شود.

Logistic Regression به روش one-vs-one

بیشتر توابع در این روش با روش قبل مشابه می‌باشند و فقط توابع متفاوت را توضیح خواهیم داد.

Load Datasets

```
def raw_data():
 data = pd.read_csv('content/iris.data',names=['Sepal Length', 'Sepal
Width', 'Petal Length', 'Petal Width', 'iris_class'])
 data["iris_class"].replace({"Iris-setosa": 0., "Iris-virginica": 1.,
"Iris-versicolor": 2.}, inplace=True)
 data1 = data[data.iris_class != 2] # 0 and 1
 data2 = data[data.iris_class != 1] # 0 and 2
 data3 = data[data.iris_class != 0] # 1 and 2
 return data1, data2, data3
```

این تابع پس از دریافت داده‌ها آنها را به دسته‌های مختلف تقسیم می‌کند.

prepared_data

```
def prepared_data():
 data1, data2, data3 = raw_data()
 data2["iris_class"].replace({2:1}, inplace=True)
 data3["iris_class"].replace({1:0, 2:1}, inplace=True)
 X_train1, X_test1, y_train1, y_test1 = train_test_split(data1.iloc[ : , 0:-1], data1.iloc[ : , -1],
                                                         stratify=data1['iris_class'], test_size=0.2,
```

```
random_state=2020)
 X_train2, X_test2, y_train2, y_test2 = train_test_split(data2.iloc[ : , 0:-1], data2.iloc[ : , -1],
                                                         stratify=data2['iris_class'], test_size=0.2,
random_state=2020)
 X_train3, X_test3, y_train3, y_test3 = train_test_split(data3.iloc[ : , 0:-1], data3.iloc[ : , -1],
                                                       stratify=data3['iris_class'], test_size=0.2,
random_state=2020)
  X_train1, X_test1 = addbias(X_train1), addbias(X_test1)
 X_train2, X_test2 = addbias(X_train2), addbias(X_test2)
 X_train3, X_test3 = addbias(X_train3), addbias(X_test3)
 res1 = (X_train1, X_test1, y_train1.to_numpy(), y_test1.to_numpy())
 res2 = (X_train2, X_test2, y_train2.to_numpy(), y_test2.to_numpy())
 res3 = (X_train3, X_test3, y_train3.to_numpy(), y_test3.to_numpy())
 return res1, res2, res3
```

این تابع مقادیر دیتاست را برای روش one-vs-one آماده سازی می‌کند.

predict_label

```
def predict_label(X, weights):
 labels = []
 for i in range(len(X)):
   temp = [0, 0, 0]
   l1 = 1 if predict(X[i], weights[0]) >= 0.5 else 0
   l2 = 1 if predict(X[i], weights[1]) >= 0.5 else 0
   l3 = 1 if predict(X[i], weights[2]) >= 0.5 else 0
   if l1 == 1:
     temp[1] = temp[1] + 1
   else:
     temp[0] = temp[0] + 1
   if l2 == 1:
     temp[2] = temp[2] + 1
   else:
     temp[0] = temp[0] + 1
   if l3 == 1:
     temp[2] = temp[2] + 1
   else:
     temp[1] = temp[1] + 1
   labels.append(temp)
 return [l.index(max(l)) for l in labels]
```

در این تابع، برچسبی که توسط سه مدل برای داده پیش بینی شده را تعیین کرده و حداکثر رای برای یک برچسب را به عنوان برچسب اصلی داده در نظر میگیرد.

prepare_data_for_acc

```
def prepare_data_for_acc():
 data1, data2, data3 = raw_data()
 X_train1, X_test1, y_train1, y_test1 = train_test_split(data1.iloc[ : ,
```

```
0:-1], data1.iloc[ : , -1],

stratify=data1['iris_class'], test_size=0.2, random_state=2020)
 X_train2, X_test2, y_train2, y_test2 = train_test_split(data2.iloc[ : ,
0:-1], data2.iloc[ : , -1],

stratify=data2['iris_class'], test_size=0.2, random_state=2020)
 X_train3, X_test3, y_train3, y_test3 = train_test_split(data3.iloc[ : ,
0:-1], data3.iloc[ : , -1],

stratify=data3['iris_class'], test_size=0.2, random_state=2020)
  X_train1, X_test1 = addbias(X_train1), addbias(X_test1)
 X_train2, X_test2 = addbias(X_train2), addbias(X_test2)
 X_train3, X_test3 = addbias(X_train3), addbias(X_test3)
 res1 = (X_train1, X_test1, y_train1.to_numpy(), y_test1.to_numpy())
 res2 = (X_train2, X_test2, y_train2.to_numpy(), y_test2.to_numpy())
 res3 = (X_train3, X_test3, y_train3.to_numpy(), y_test3.to_numpy())
 return res1, res2, res3
```

این تابع مشابه تابع بالاتر که توضیح دادیم هست فقط با انجام تغییرات کوچکی برای تابع تعیین دقت کتاب خانه sklearn داده ها را اماده میکند.

Softmax Regression

بیشتر توابع در این روش با روش‌های قبل مشابه می‌باشند و فقط توابع متفاوت را توضیح خواهیم داد.

softmax

```
def softmax(z):
   z -= np.max(z)
   sm = (np.exp(z).T / np.sum(np.exp(z),axis=1)).T
   return s
```

تابع سافتمکس که توضیح داده شد را در اینجا پیاده سازی کرده ایم.

train

```
def train(X_train, y_train, lr, epochs, weights):
   for i in range(epochs):
       xw = np.dot(X_train, weights.T)
       prob = softmax(xw)
```

```
        grad = gradient_dsc(X_train, y_train, prob)
        weights = update_weights(weights, lr, grad)
        loss = compute_loss(prob, y_train)
        loss_log.append(loss)
    return weights
```

در این تابع بر اساس فرمول های شرح داده شده، مدل آموزش داده می شود.

پارامتر ها

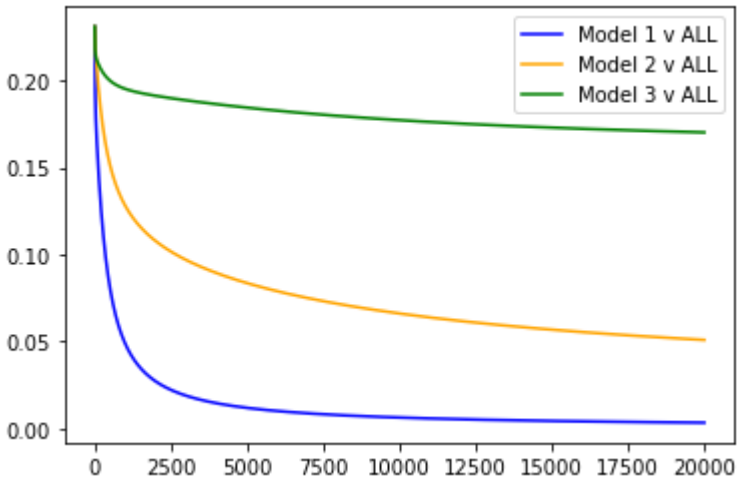
برای آموزش ما چند پارامتر را برای تنظیمات مدل تعیین کرده‌ایم.

- Epochs : این مقدار تعیین کننده تعداد تکرار و دفعات اجرای الگوریتم می‌باشد که ما 20000 انتخاب کرده‌ایم.
- Lr : این همان نرخ یادگیری است که ما 0.01 انتخاب کرده‌ایم.

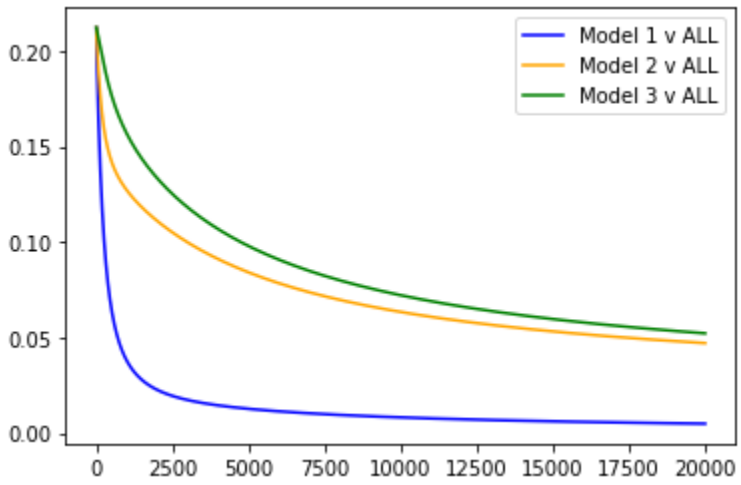
نتیجه

با توجه به نتایج بدست آمده که در جدول زیر مشاهده می‌کنید الگوریتم Softmax Regression کمی بهتر از دیگر روش‌ها در داده‌های Train عمل کرده است هرچند مقدار Accuracy برای هر ۳ روش برای داده‌های Test 100٪ بوده است.

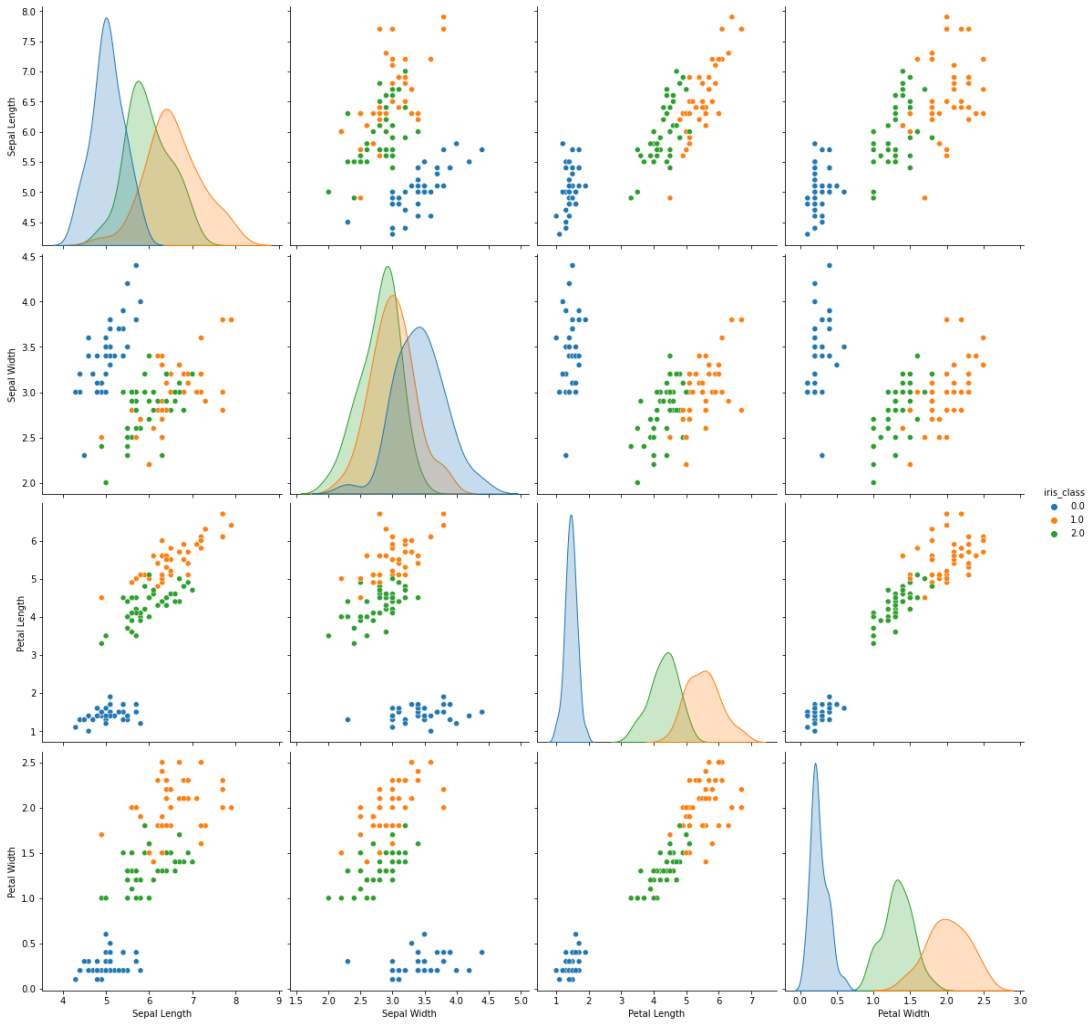
Test Accuracy	Train Accuracy	
1.0	0.9583333333333334	**one-vs-all**
1.0	0.9625	**one-vs-one**
1.0	0.975	**softmax**

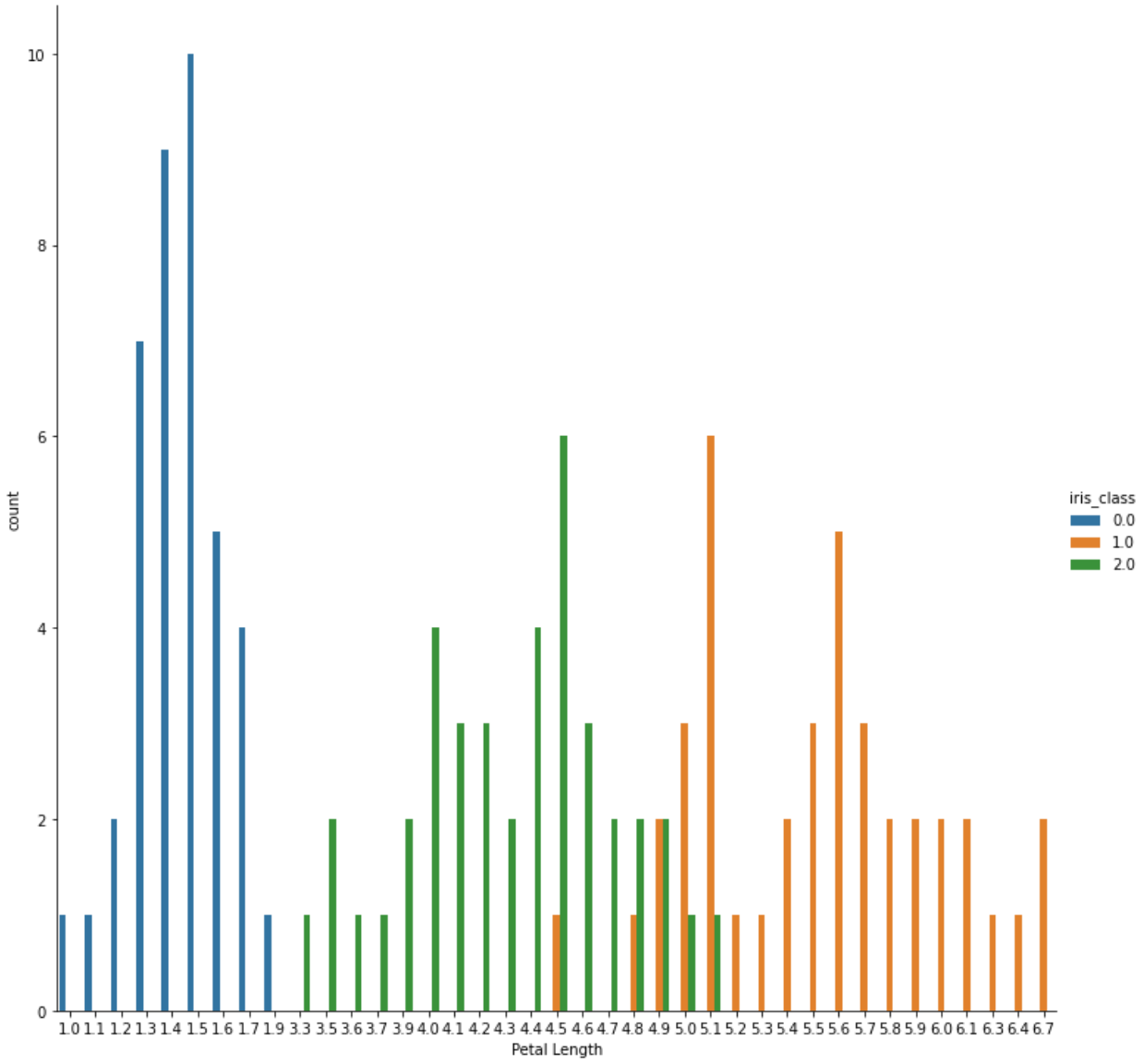


نمودار تابع هزینه برای روش one-vs-all



نمودار تابع هزینه برای روش softmax


Sepal Length
Sepal Width
Petal Length
Petal Width
iris_class
0.0
1.0
2.0



نمودار Petal Length برای روش one-vs-all

بخش دوم

در این بخش الگوریتم Gaussian Discriminant Analysis را بر روی دیتاست قرار داده شده در فایل تمرین پیاده سازی می‌کنیم و میزان Accuracy آن را بررسی می‌کنیم.

Gaussian Discriminant Analysis

برای یک مسئله طبقه بندی که در آن ویژگی های ورودی x متغیرهای تصادفی با ارزش پیوسته هستند، مدل تجزیه و تحلیل تفکیک گاوسی فرض می کند که:

$p(y)$ طبق توزیع برنولی توزیع می شود:

$$y \sim \text{Bernoulli}(\phi) \iff p(y) = \phi^y (1-\phi)^{1-y}$$

$p(x|y)$بر اساس توزیع نرمال چند متغیره توزیع می شود:

$$x|y=0 \sim \mathcal{N}(\mu_0, \Sigma) \iff p(x|y=0) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0)\right)$$
$$x|y=1 \sim \mathcal{N}(\mu_1, \Sigma) \iff p(x|y=1) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)\right).$$

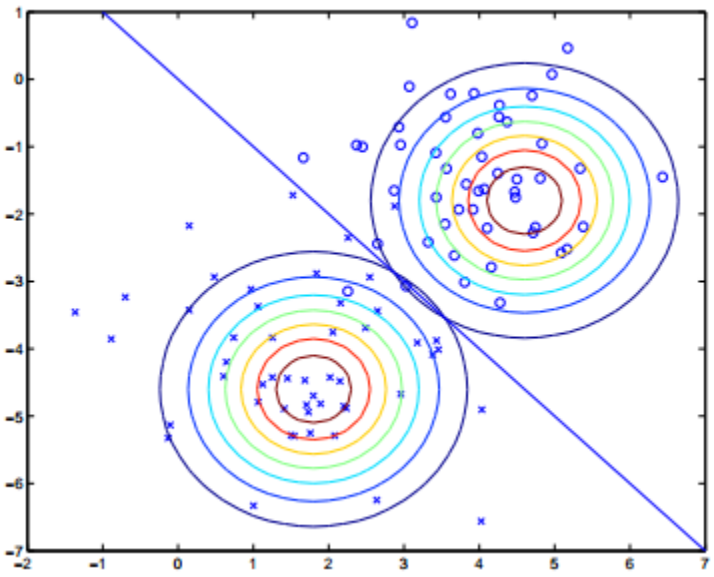
و برای به حداکثر رساندن log-likelihood از فرمول زیر استفاده می‌کند

$$\begin{aligned}\ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma\right) \\ &= \log \prod_{i=1}^{m} p\left(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma\right) p\left(y^{(i)}; \phi\right).\end{aligned}$$

با توجه به پارامترهای {φ، μ0، μ1، Σ} حداکثر برآورد احتمال پارامترها را می‌دهد:

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)}=1\}$$
$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=0\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=0\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}$$
$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T.$$

شکل زیر مرز تصمیم گیری خطی (که در آن p (y = 1 | x) = 0.5) از یک مدل تجزیه و تحلیل تفکیک گاوسی که در مجموعه ای از موارد آموزش با دو کلاس و خطوط توزیع گاوسی (با معنی متفاوت اما شکل و جهت یکسان) متناسب با نمونه های هر دو کلاس است.



پیاده سازی

plot_pdfs

```
def plot_pdfs(data, mus, sigmas, labels, priors, y_train):
   ax = plt.axes(projection="3d")

   colors = [('b', 'viridis'), ('r', 'plasma')]

   fig2 = plt.figure()
   ax2 = fig2.gca()

   for cls in range(len(labels)):
       ax.scatter3D(data[y_train == cls][:, 0], data[y_train == cls][:, 1],
np.ones(1) * -0.03, c=colors[cls][0])
       mu = np.asarray(mus[cls]).flatten()
       x = np.linspace(mu[0] - 3 * sigmas[cls][0, 0], mu[0] + 3 *
sigmas[cls][0, 0], 40).flatten()
       y = np.linspace(mu[1] - 3 * sigmas[cls][1, 1], mu[1] + 3 *
sigmas[cls][1, 1], 40).flatten()
```

```
        X, Y = np.meshgrid(x, y)
        pos = np.empty(X.shape + (2,))
        pos[:, :, 0] = X
        pos[:, :, 1] = Y
        print(pos.shape)
        rv = multivariate_normal([mu[0], mu[1]], sigmas[cls])
        Z = rv.pdf(pos)
        ax.plot_surface(X, Y, Z, cmap=colors[cls][1], linewidth=0.2,
alpha=0.9, shade=True)

        ax2.contour(X, Y, Z, cmap='coolwarm')
    x = np.linspace(np.min(data[:, 0]), np.max(data[:, 0]), 40).flatten().T
    y = np.linspace(np.min(data[:, 1]), np.max(data[:, 1]), 40).flatten().T
    b0 = 0.5 * mus[0].T.dot(np.linalg.pinv(sigmas[0])).dot(mus[0])
    b1 = -0.5 * mus[1].T.dot(np.linalg.pinv(sigmas[1])).dot(mus[1])
    b = b0 + b1 + np.log(priors[0]/priors[1])
    a = np.linalg.pinv(sigmas[0]).dot(mus[1] - mus[0])
    x1 = -(b + a[0]*x) / a[1]
    ax2.plot(x, x1)
```

تابع نمایش نمودار PDF.

plot_dec_boundary

```
def plot_dec_boundary(train_data, test_x, prediction, test_y, mus, sigmas,
priors, title):
    missed_0 = np.take(test_x, np.setdiff1d(np.where(prediction == 0),
np.where(test_y == 0)), axis=0)
    missed_1 = np.take(test_x, np.setdiff1d(np.where(prediction == 1),
np.where(test_y == 1)), axis=0)
    cl0 = np.delete(test_x, np.where(prediction != 0), axis=0)
    cl1 = np.delete(test_x, np.where(prediction != 1), axis=0)
    plt.plot(cl0[:, 0], cl0[:, 1], '.c')
    plt.plot(missed_0[:, 0], missed_0[:, 1], '.r')
    plt.plot(cl1[:, 0], cl1[:, 1], '.y')
    plt.plot(missed_1[:, 0], missed_1[:, 1], '.k')
    x = np.linspace(np.min(train_data[:, 0]), np.max(train_data[:, 0]),
40).flatten().T
    y = np.linspace(np.min(train_data[:, 1]), np.max(train_data[:, 1]),
40).flatten().T
    b0 = 0.5 * mus[0].T.dot(np.linalg.pinv(sigmas[0])).dot(mus[0])
    b1 = -0.5 * mus[1].T.dot(np.linalg.pinv(sigmas[1])).dot(mus[1])
    b = b0 + b1 + np.log(priors[0]/priors[1])
```

```
    a = np.linalg.pinv(sigmas[0]).dot(mus[1] - mus[0])
    x1 = -(b + a[0]*x) / a[1]
    plt.plot(x, x1)
    plt.title(title)
    plt.show()
    return
```

تابع نمایش Decision Boundary.

fit

```
def fit(X_train, y_train, classes, priors, means):
 x, y = X_train.values, y_train.values
 for i, label in enumerate(classes):
   priors[i] = y[y == label].size / y.size
   means[i] = np.mean(x[y == label], axis=0)
 covariance_matrix = covariance(x, y, classes, means)
 return means, priors, covariance_matrix
```

تابع ساخت مدل با توجه به الگوریتم.

covariance

```
def covariance(x, y, classes, means):
 covs = [0 for i in range(len(classes))]
 cov = np.zeros(shape=(x.shape[1], x.shape[1]))
 for i, label in enumerate(classes):
   members = x[y == label]
   x_mu = members - means[i]
   cov += (x_mu).T.dot(x_mu)
   cov /= x.shape[0] #delete if single
   covs[i] = cov # cov /= X.shape[0] out of loop if single
 return covs # return cov if single
```

تابع محاسبه کواریانس.

probability

```
def probability(x, mean, prior, covariance_matrix):
 xm = x - mean
 xm_covariance = (xm.dot(np.linalg.pinv(covariance_matrix))) * xm
 xm_covariance_sum = xm_covariance.sum(axis=1)
```

```
 return -0.5 * xm_covariance_sum + np.log(prior)
```

تابع محاسبه احتمال.

predict

```
def predict(X_train, classes, means, priors, covariance_matrix):
 x = X_train.values
 probabilities = np.zeros((x.shape[0], priors.size))
 for i,_ in enumerate(classes):
   probabilities[:, i] = probability(
       x, means[i]
       , priors[i]
       , covariance_matrix[i]) # covariance_matrix -> if single
 return np.argmax(probabilities, axis=1)
```

تابع بدست آوردن نتیجه.

predict_proba

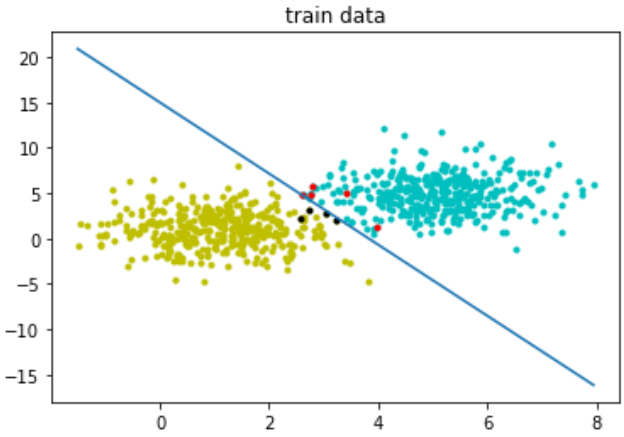
```
def predict_proba(x):
 probabilities = np.zeros((x.shape[0], priors.size))
 for i,_ in enumerate(classes):
   probabilities[:, i] = probability(
       x, means[i]
       , priors[i]
       , covariance_matrix)
 return probabilities
```

تابع محاسبه احتمال نتیجه.

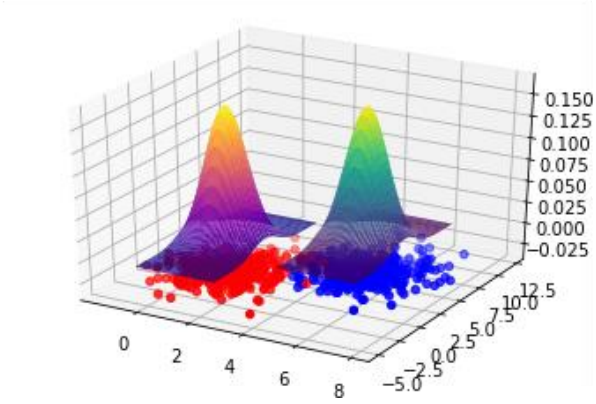
نتیجه

Test Accuracy	Train Accuracy	
1.0	0.98875	BC-1
1.0	0.99125	BC-2

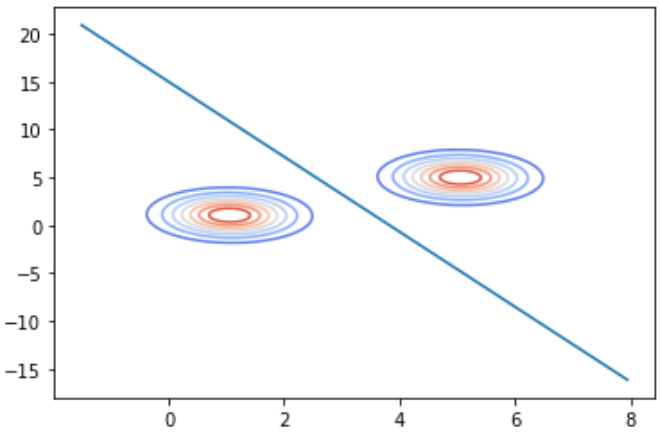
در ادامه نمودارهای مورد نظر اضافه شده است.



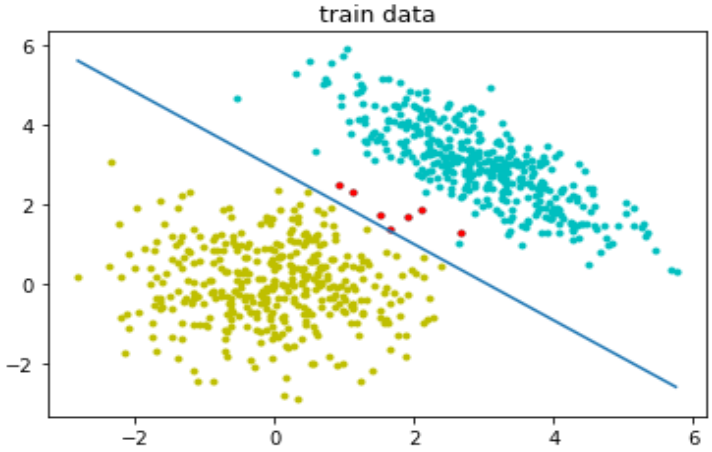
نمودار Decision Bound دیتاست 1BC-Train



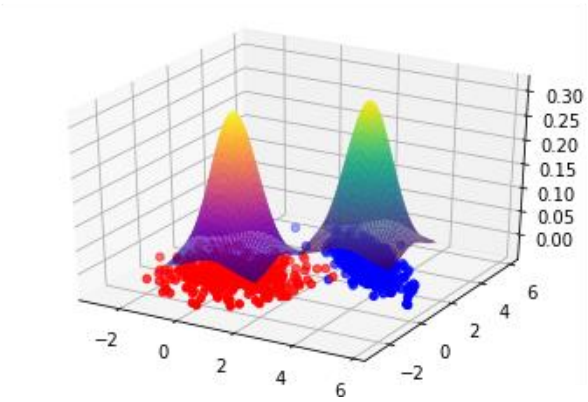
نمودار PDF دیتاست 1BC-Train



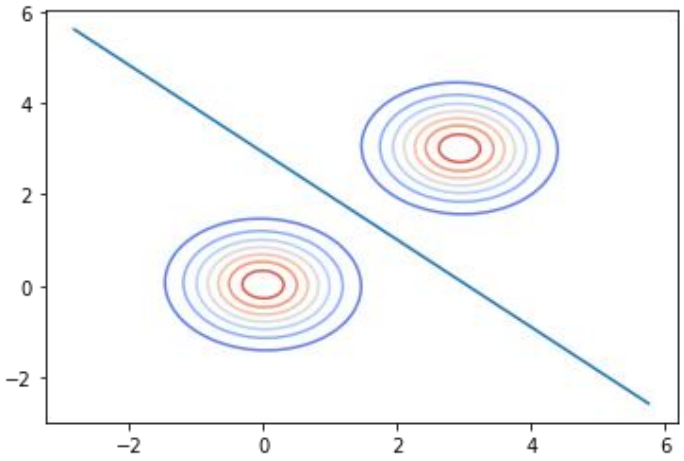
نمودار Contour دیتاست 1BC-Train



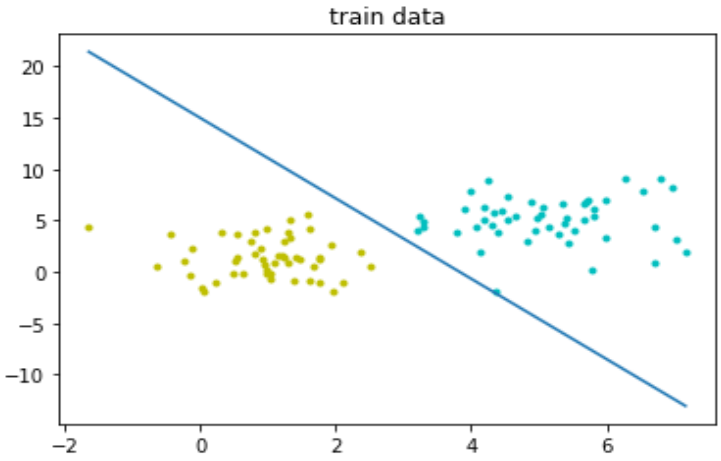
نمودار Decision Bound دیتاست BC-Train2



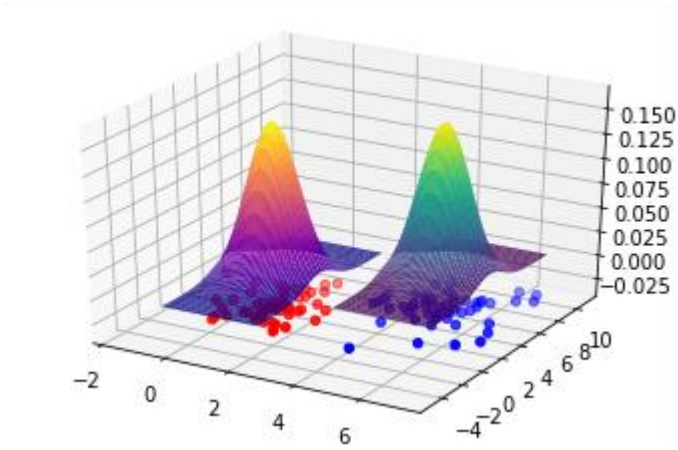
نمودار PDF دیتاست BC-Train2



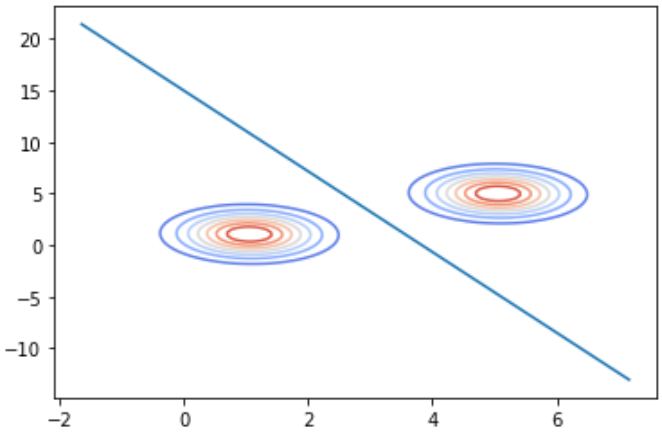
نمودار Contour دیتاست BC-Train2



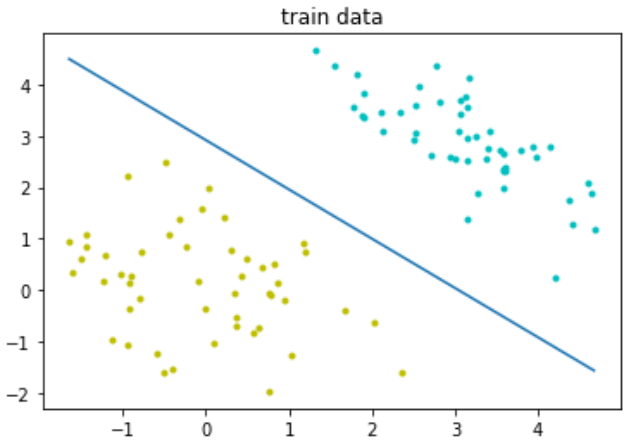
نمودار Decision Bound دیتاست BC-Test1



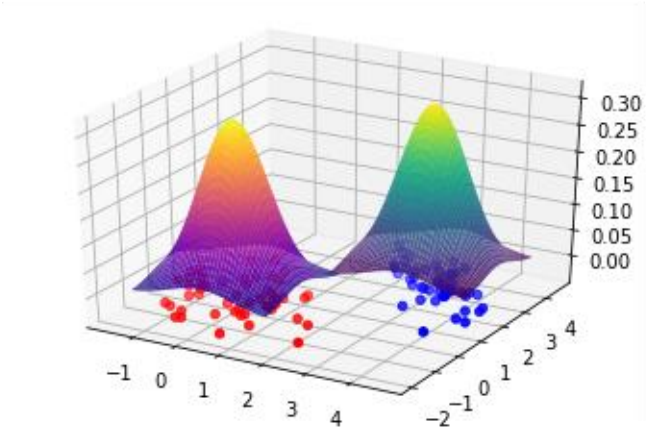
نمودار PDF دیتاست BC-Test1



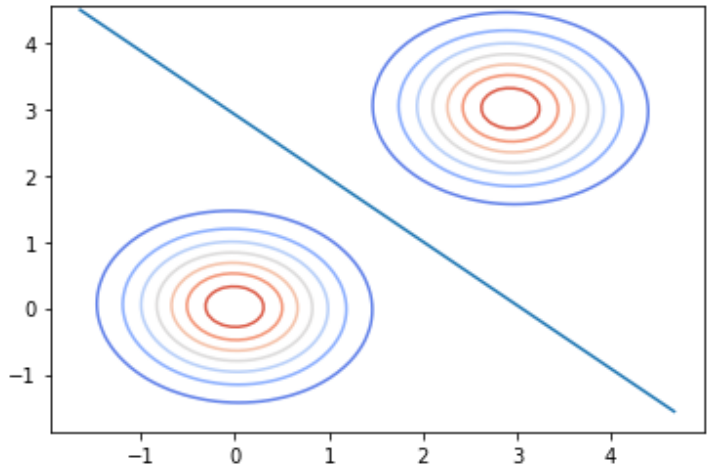
نمودار Contour دیتاست BC-Test1



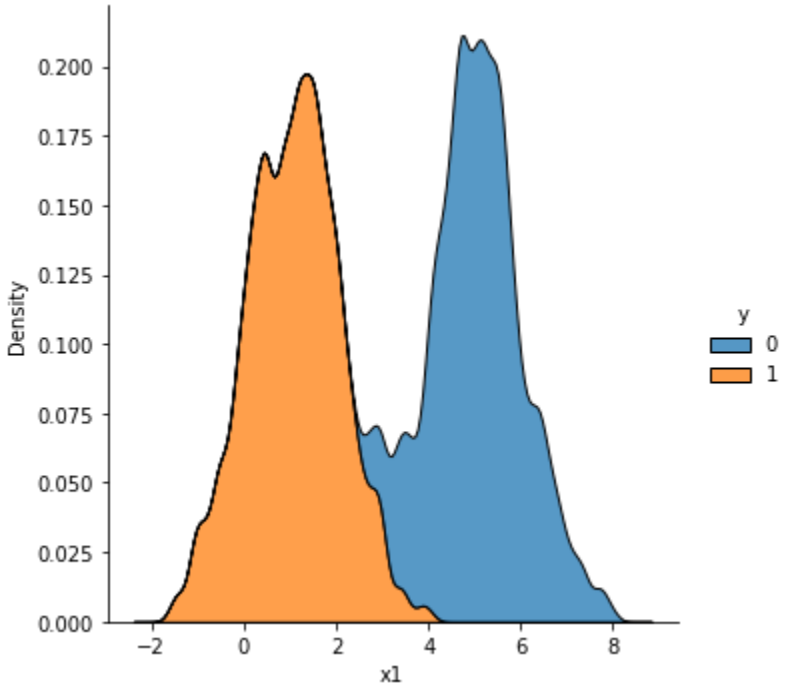
نمودار Decision Bound دیتاست BC-Test2



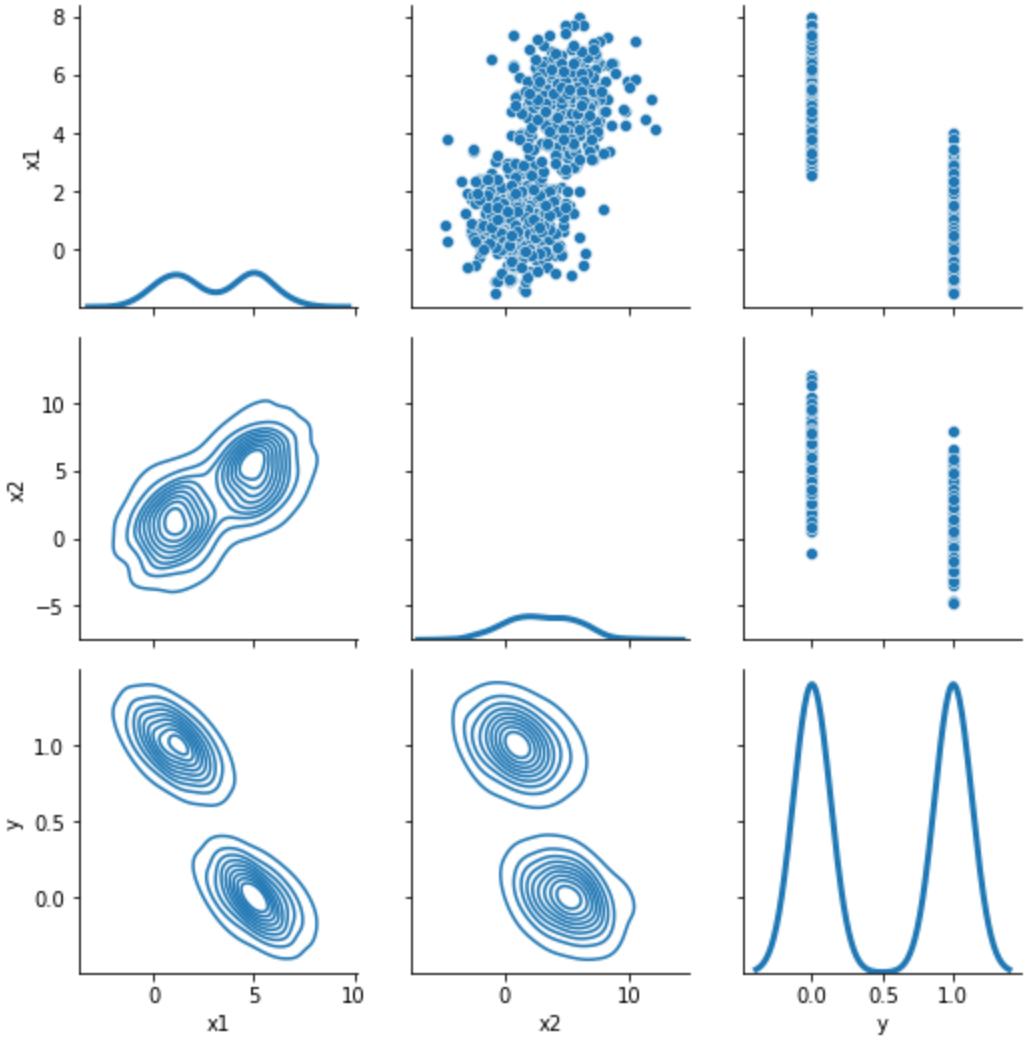
نمودار PDF دیتاست BC-Test2



نمودار Contour دیتاستBC-Test2



نمودار Density دیتاست 1BC-Train



نمودار داده‌های دیتاست 1BC-Train

تفاوت بین دو دیتاست

با توجه به نمودار های رسم شده میتواند نتیجه گرفت که دیتاست BC-Train1 با توجه به نمودار Contour آن که بیضی شکل است توزیع گاوسی آن نا متقارن میباشد اما دیتاست BC-Train2 با توجه به نمودار Contour آن که دایره ای شکل است توزیع گاوسی آن متقارن است.