گزارش تمرین سوم

(شناسایی آماری الگو)

تهیه شده توسط:

سارا لیمویی، ملیکا زارع

استاد:

دکتر عظیمی فر

پاییز 1400

بسم الله الرحمن الرحیم

Linear Discriminant Analysis (LDA)

در این روش فرض می کنیم که متغیر تصادفی X یک بردار X=(X1,X2,...,Xp) است که از یک گاوسی چند متغیره با بردار میانگین کلاس خاص و یک ماتریس کوواریانس مشترک Σ گرفته شده است .به عبارت دیگر ماتریس کوواریانس برای همه کلاس های K مشترک است: Cov(X)=Σ

این ماتریس با ابعاد n*n که n تعداد feature ها می باشد.

از آنجایی که x از یک توزیع گاوسی چند متغیره پیروی می کند، احتمال p(X |y=i) به صورت زیر داده می شود: (μ_i میانگین ورودی برای دسته i است)

$$P(\mathbf{X}|y=i) = \frac{1}{(\sqrt{2\pi})^n|\Sigma|^{\frac{1}{2}}}\ exp(\frac{-1}{2}(\mathbf{X}-\boldsymbol{\mu}_i)^T\Sigma^{-1}(\mathbf{X}-\boldsymbol{\mu}_i))$$

هم چنین می دانیم که $P(y) = \phi_1^{1\{y=1\}}\phi_2^{1\{y=2\}}\cdots\phi_c^{1\{y=c\}}$ و در نهایت پارامتر های ما به صورت زیر در می آیند:

$$\phi_i^{MLE} = \frac{\sum_{j=1}^m 1\{y^{(j)}=i\}}{m}$$

$$\boldsymbol{\mu}_i^{MLE} = \frac{\sum_{j=1}^m 1\{y^{(j)}=i\}\mathbf{X}^{(j)}}{\sum_{j=1}^m 1\{y^{(j)}=i\}}$$

$$\Sigma^{MLE} = \frac{1}{m}\sum_{j=1}^m(\mathbf{X}^{(j)}-\boldsymbol{\mu}_{y^{(j)}})(\mathbf{X}^{(j)}-\boldsymbol{\mu}_{y^{(j)}})^T$$

زمانی که از P(y | X) لوگاریتم می گیریم تابع هدف یا objective function به صورت زیر بدست می آید:

$$\log(P(y\,|\,X)) = \log(\phi_i) - \frac{1}{2}{\mu_i}^T\,\Sigma^{-1}\,\mu_i + x^T\,\Sigma^{-1}\,\mu_i$$
$$= \log(\phi_i) - \frac{1}{2}(x^T - \mu_i)^T\,\Sigma^{-1}\,(x^T-\mu_i)$$

در مرز تصمیم گیری مجموعه نقاطی داریم که در آن دو کلاس به یک اندازه محتمل هستند. بنابراین احتمال را برابر قرار میدهیم تا به فرمول زیر برسیم:

$$log\frac{\phi_i}{\phi_j} - \frac{1}{2}(\mu_i+\mu_j)^T\,\Sigma^{-1}(\mu_i-\mu_j) + x^T\,\Sigma^{-1}(\mu_i-\mu_j) = 0$$

یا به صورت ساده تر داریم:

$$log\frac{\phi_i}{\phi_j} - \frac{1}{2}{\mu_i}^T\,\Sigma^{-1}\mu_i + \frac{1}{2}{\mu_j}^T\,\Sigma^{-1}\mu_j + x^T\,\Sigma^{-1}(\mu_i-\mu_j) = 0$$

کلاس LDA

در این کلاس توابع مورد نیاز برای پیاده سازی مدل LDA قرار گرفته اند. در تابع fit با گرفتن داده ی آموزشی پارامترهای مورد نیاز را حساب میکنیم. ابتدا تعداد کلاس ها را بدست می آوریم و سپس به ازای هر کلاس میانگین داده های آن کلاس و در نهایت ماتریس کوواریانس را بدست می آوریم تا در توابع دیگر استفاده بشوند.

در تابع phi نیز همانند فرمول های ارائه شده در بالا احتمال P(y) را حساب کرده ایم.

در تابع mean و cov با فرمول های بدست آمده میانگین و ماتریس کوواریانس را محاسبه می کنیم.

در تابع decision_boundry با گرفتن x و دو عدد i , j مرز بین دو کلاس i , j را خروجی می دهیم. توجه کنید که تعداد feature های ما دو می باشد پس معادله خط را x1 برحسب x2 در می آوریم.

در تابع plot_decision داده های x را به ازای هر کلاس با رنگ متفاوت کشیده و با استفاده از تابع predict کلس حدس زده شده کلاسیفایر را نیز نمایش میدهیم. اگر کلاس دیگری حدس زده باشد یا به اصطلاح miss classify کرده باشد با رنگ متفاوت نشان داده ایم. در نهایت خط decision boundry از تابع قبلی را رسم می کنیم.

در تابع plot_pdf به ازای هر کلاس pdf مربوط به داده های آن را سه بعدی رسم کرده ایم. سپس contour های این Pdf ها را جداگانه در plot دیگر نشان داده و داده های x و خط مرزبندی را نیز چاپ کرده ایم.

در تابع predict نیز با اجرای فرمول بدست امده قبل احتمال هر کلاس را بدست می آوریم و با عمل argmax کلاسی که بیشترین احتمال دارد را مشخص می کنیم.

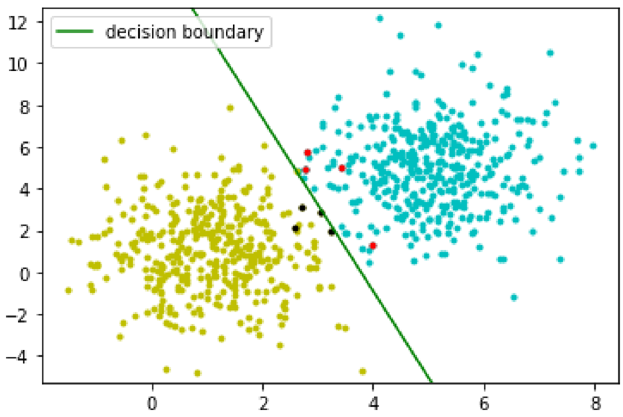
در تابع accuracy_metric نیز لیبل های حدس زده شده مدل را با مقدار واقعی آنها مقایسه کرده و درصد درستی آن را گزارش می دهیم.

با داده آموزش دیتاست اول داریم:

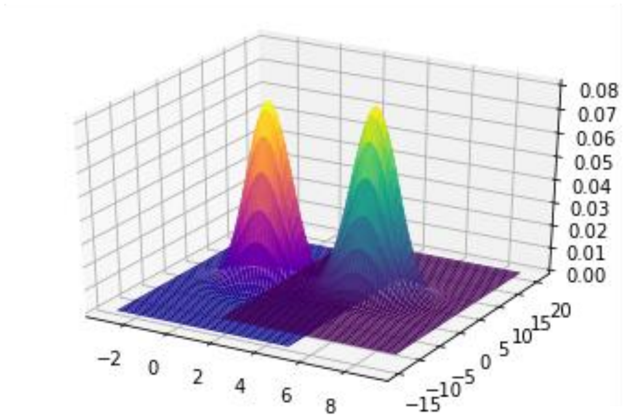
```
data = Dataset('BC-Train1.csv', 'BC-Test1.csv')

classifier = LDA()
classifier.fit(data.x_train, data.y_train)

# train
classifier.plot_decision(data.x_train, data.y_train)

classifier.plot_pdf(data.x_train, data.y_train, classifier.mean_MLE)
pred = classifier.predict(data.x_train)
print('train: ', classifier.accuracy_metric(pred, data.y_train))
pred = classifier.predict(data.x_test)
print('test: ',classifier.accuracy_metric(pred, data.y_test))
```

```
No normalization.
train:  99.0
test:  100.0
```

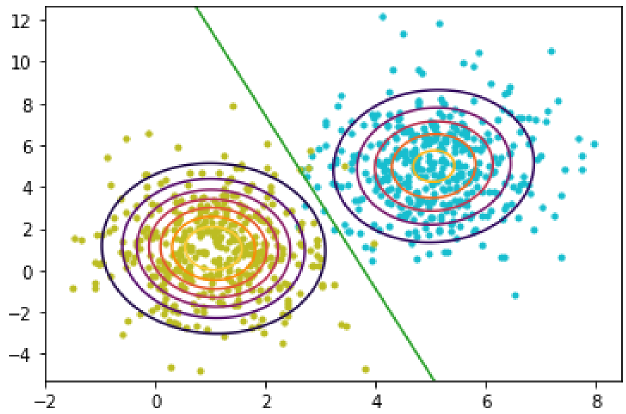
Train:
accuracy: 99.0
precison: [0.99, 0.99]
recall: [0.99, 0.99]
F1: [0.99, 0.99]
~.
~.
Test
accuracy: 100.0
precison: [1.0, 1.0]
recall: [1.0, 1.0]
F1: [1.0, 1.0]



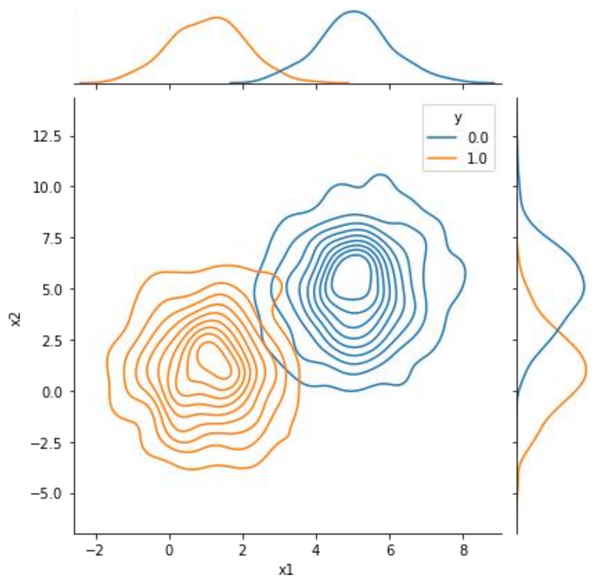
نمودار Decision Bound دیتاست 1BC-Train



نمودارPDF دیتاست 1BC-Train



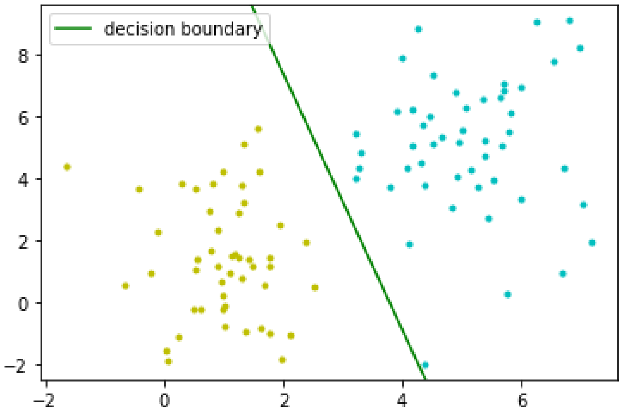
نمودار Contour دیتاست 1BC-Train



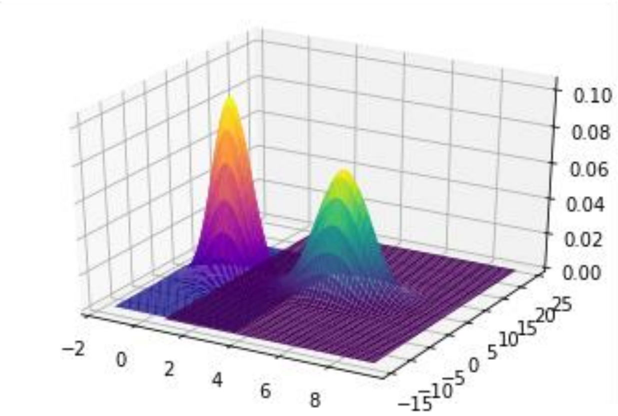
نمودار Contour دیتاست 1BC-Train

با داده تست دیتاست اول داریم:

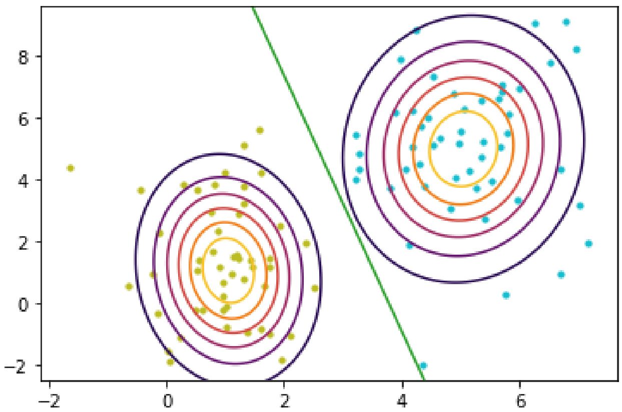
Test
accuracy: 100.0
precison: [1.0, 1.0]
recall: [1.0, 1.0]
F1: [1.0, 1.0]



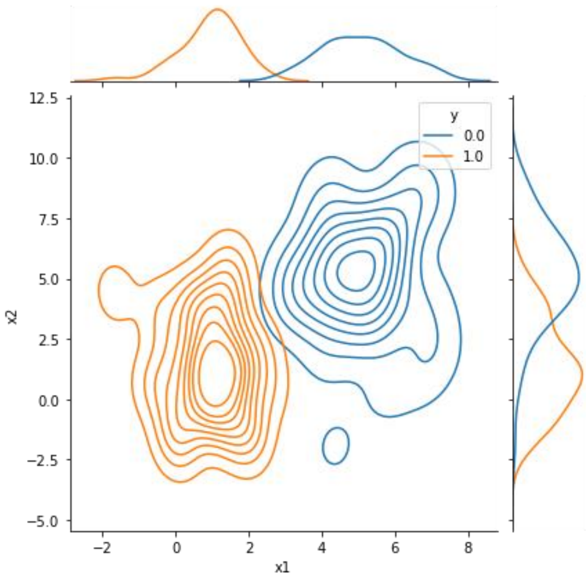
نمودار Decision Bound دیتاست 1BC-Test



نمودار PDF دیتاست 1BC-Test



نمودار Contour دیتاست 1BC-Test



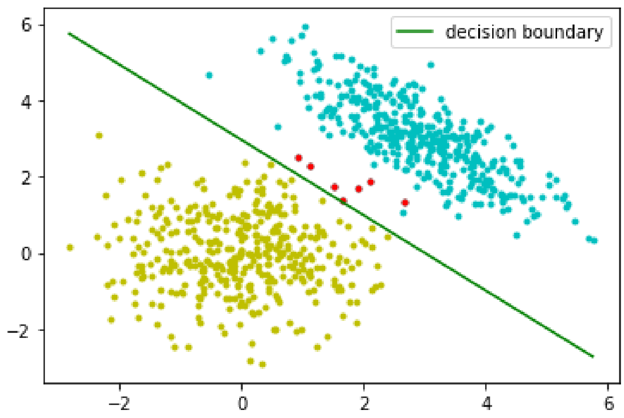
نمودار Contour دیتاست 1BC-Test

با داده آموزش دیتاست دوم داریم:

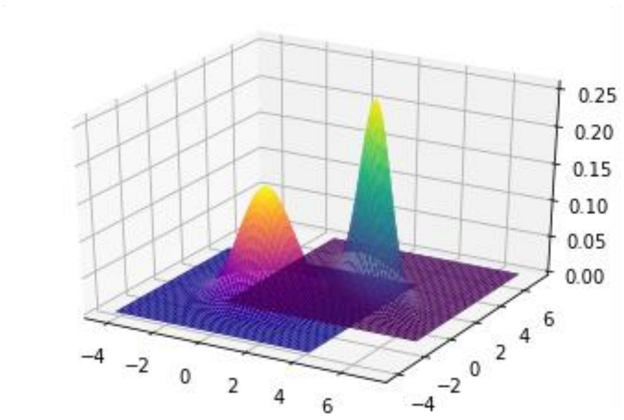
```
data = Dataset('BC-Train2.csv', 'BC-Test2.csv')

classifier = LDA()
classifier.fit(data.x_train, data.y_train)

# train
classifier.plot_decision(data.x_train, data.y_train)

classifier.plot_pdf(data.x_train, data.y_train, classifier.mean_MLE)
pred = classifier.predict(data.x_train)
print('train: ', classifier.accuracy_metric(pred, data.y_train))
pred = classifier.predict(data.x_test)
print('test: ',classifier.accuracy_metric(pred, data.y_test))
```

```
No normalization.
train:  99.125
test:  100.0
```

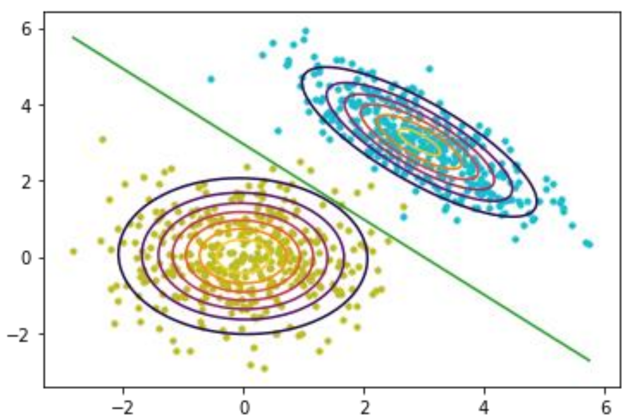
Train:
accuracy: 99.125
precison: [0.98, 1.0]
recall: [1.0, 0.98]
F1: [0.99, 0.99]
~.
~.
Test
accuracy: 100.0
precison: [1.0, 1.0]
recall: [1.0, 1.0]
F1: [1.0, 1.0]



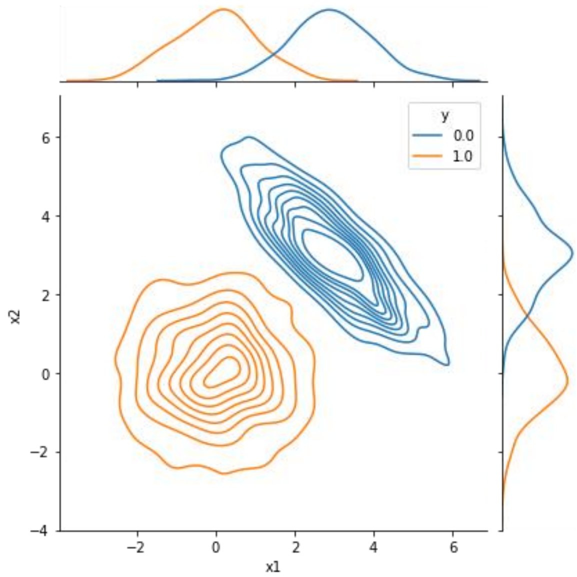
نمودار Decision Bound دیتاست 2BC-Train



نمودار PDF دیتاست 2BC-Train



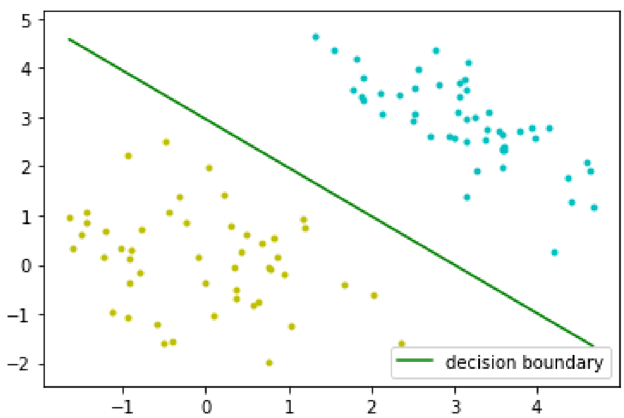
نمودار Contour دیتاست 2BC-Train



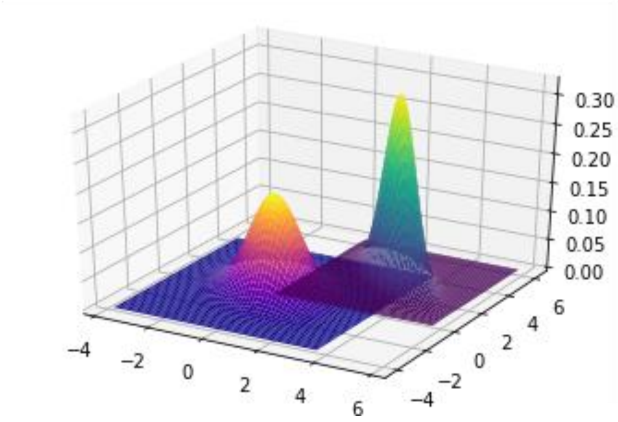
نمودار Contour دیتاست 2BC-Train

با داده تست دیتاست دوم داریم:

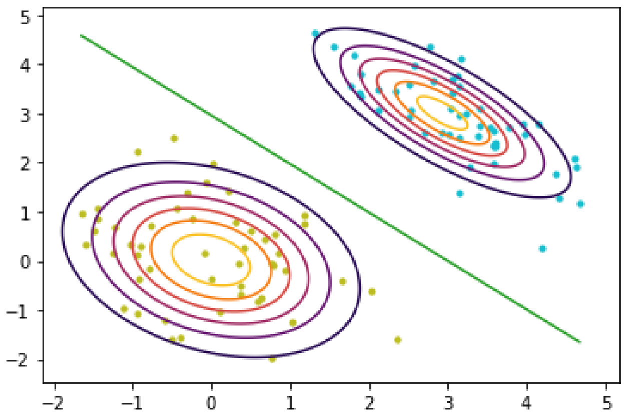
Test
accuracy: 100.0
precison: [1.0, 1.0]
recall: [1.0, 1.0]
F1: [1.0, 1.0]



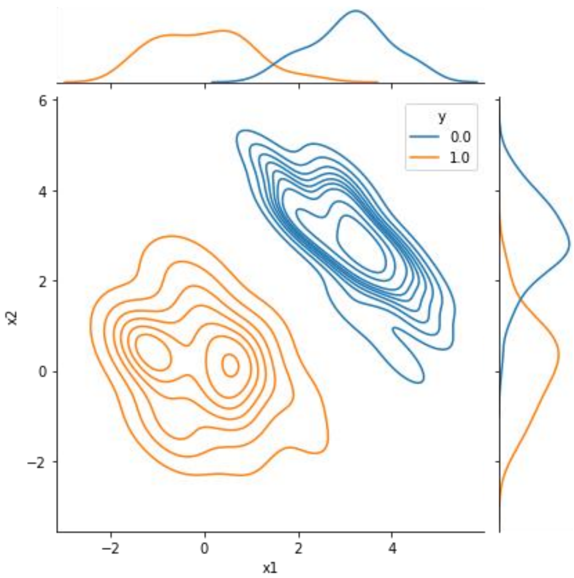
نمودار Decision Bound دیتاست 2BC-Test



نمودار PDF دیتاست 2BC-Test



نمودار Contour دیتاست 2BC-Test



نمودار Contour دیتاست 2BC-Test

مقایسه

با توجه به مقایسه ماتریس کوواریانس دو دیتاست داریم:

```
Cov_1_class_0 = array([[0.98559161, 0.08682345],        ≈ [[1 , 0],
                       [0.08682345, 3.96907885]])          [0 , 4]]

Mean_class_0 = [5.054563   , 4.99381302]                 ≈ [5 , 5]
Cov_1_class_1 = array([[ 0.98916202, -0.06353189],       ≈ [[1 , 0],
                       [-0.06353189,  3.99916563]])          [0 , 4]]

Mean_class_0 = [1.05561733, 1.05755972]                  ≈ [1 , 1]
```

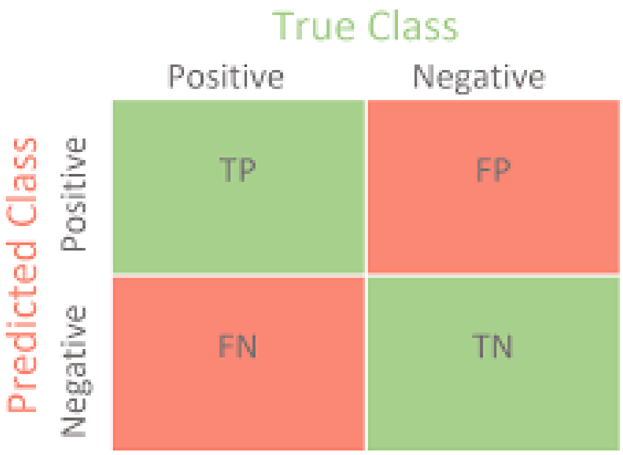
~.
~.

```
Cov_2_class_0 = array([[ 1.02744599, -0.82188321],      ≈ [[1 , -0.8],
                       [-0.82188321,  1.03709589]])        [-0.8 , 1]]

Mean_class_0 = [ 2.93103968e+00,  3.01210055e+00]        ≈ [3 , 3]


Cov_2_class_1 = array([[ 1.0482398 , -0.03059561],      ≈ [[1 , 0],
                       [-0.03059561,  1.02329013]])        [0 , 1]]

Mean_class_0 = [-1.94261375e-03,  2.02144056e-02]        ≈ [0 , 0]
```

همان طور که از قیاس انحراف معیار این دو دیتاست داریم contour دیتاست اول بیشتر شبیه به بیضی است و توزیع گوسی نامتقارن دارد اما در دیتاست دوم کلاس یک دایره ای شکل متقارن و کلاس صفر بیضی با محوریت در راستای حدود 135 درجه داریم.

Confusion Matrix

در این قسمت، confusion matrix با ابعاد n*n، پیاده سازی شده است. در شکل زیر، برای 2*2 آورده شده است.



با داشتن confusion matrix و به کمک فرمول های زیر، میتوان هر کدام از متریک ها را محاسبه کرد.

$$
\begin{aligned}
precision &= \frac{TP}{TP+FP} \\
recall &= \frac{TP}{TP+FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP+TN}{TP+FN+TN+FP}
\end{aligned}
$$

Accuracy Metric

```python
def accuracy_metric(self, y_predict, y):
    confusion_matrix = self.compute_confusion_matrix(y_predict, y)
    diagonal_sum = confusion_matrix.trace()
    sum_of_all_elements = confusion_matrix.sum()
    return diagonal_sum * 100 / sum_of_all_elements
```

F1 Metric

```python
def F1_metric(self, y_predict, y):
    F1_arr = []
    confusion_matrix = self.compute_confusion_matrix(y_predict, y)
    for label in range(self.c):
        row = confusion_matrix[label, :]
        col = confusion_matrix[:, label]
        tmp = round(2*confusion_matrix[label, label] / (row.sum()+col.sum()), 2)
        F1_arr.append(tmp)
    return F1_arr
```

Precision Metric

```python
def precision_metric(self, y_predict, y):
    precision_arr = []
    confusion_matrix = self.compute_confusion_matrix(y_predict, y)
    for label in range(self.c):
        row = confusion_matrix[label, :]
        tmp = round(confusion_matrix[label, label] / row.sum(), 2)
        precision_arr.append(tmp)
    return precision_arr
```

Recall Metric

```python
def recall_metric(self, y_predict, y):
    recall_arr = []
    confusion_matrix = self.compute_confusion_matrix(y_predict, y)
    for label in range(self.c):
        col = confusion_matrix[:, label]
        tmp = round(confusion_matrix[label, label] / col.sum(), 2)
        recall_arr.append(tmp)
    return recall_arr
```

Generate Dataset from Gaussian Distribution

در این قسمت، می خواهیم 2 مجموعه داده با 3 کلاس از توزیع گوسی که هر کدام 500 نمونه دارند، تولید کنیم. برای این کار از کتابخانه scipy.stats، تابع multivariate_normal را import می کنیم. در این قسمت ، هر کلاس، ماتریس cov مخصوص خودش را دارد که در ادامه پارامترهای توزیع های گوسی آورده شده اند.

Dataset-I

Class1: $\mu = \begin{pmatrix} 3 \\ 6 \end{pmatrix} \quad \sum = \begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix}$

Class2: $\mu = \begin{pmatrix} 5 \\ 4 \end{pmatrix} \quad \sum = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$

Class3: $\mu = \begin{pmatrix} 6 \\ 6 \end{pmatrix} \quad \sum = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Dataset-II

Class1: $\mu = \begin{pmatrix} 3 \\ 6 \end{pmatrix} \quad \sum = \begin{pmatrix} 1.5 & 0.1 \\ 0.1 & 0.5 \end{pmatrix}$

Class2: $\mu = \begin{pmatrix} 5 \\ 4 \end{pmatrix} \quad \sum = \begin{pmatrix} 1 & -0.20 \\ -0.20 & 2 \end{pmatrix}$

Class3: $\mu = \begin{pmatrix} 6 \\ 6 \end{pmatrix} \quad \sum = \begin{pmatrix} 2 & -0.25 \\ -0.25 & 1.5 \end{pmatrix}$

همان طور که در کد زیر نیز مشاهده می شود، به ازای هر کدام از mean و Σ های هر مجموعه داده، 500 نمونه داده از توزیع گوسی تولید کرده و همچنین در y-array، خروجی متناسب باکلاس هر نمونه را ایجاد می کنیم.

```
def generate_dataset(self, num_of_samples, mean, covariance, num_of_classes, seed=1000):
    x_data = []
    y_data = []
    for i in range(self.num_of_classes):
        self.mvd = multivariate_normal(cov = covariance[i], mean = mean[i], seed = seed)

        # generating 500 samples out of the distribution
        x_tmp = self.mvd.rvs(size = num_of_samples)
        y_tmp = np.array([i]*self.num_of_samples)

        x_data.append(x_tmp)
        y_data.append(y_tmp)

    # concatenate all x_datas and y_datas
    self.x = list(x_data[0]) + list(x_data[1]) + list(x_data[2])
    self.y = list(y_data[0]) + list(y_data[1]) + list(y_data[2])
    self.x = np.array(self.x)
    self.y = np.array(self.y).reshape(-1, 1)
```

نهایتا، همه ی داده ها و label ها را در آرایه ی نهایی به نام x و y ذخیره می کنیم تا در آخر، برای shuffle کردن و جدا کردن داده های test و train به نسبت 80% داده ی train و 20% داده ی test، از آن استفاده کنیم.

Quadratic Discrimination Analysis (QDA)

LDA فرض می‌کند که داده های هر کلاس از یک توزیع گاوسی چند متغیره با میانگین خاص کلاس، اما یک ماتریس کوواریانس که برای همه کلاس‌های i مشترک است، گرفته شده‌اند. QDA با فرض اینکه هر کلاس ماتریس کوواریانس Σ_i خود را دارد، یک رویکرد جایگزین ارائه می‌کند.

زمانی که از P(y | X) لوگاریتم می گیریم تابع هدف یا objective function به صورت زیر بدست می آید:

$$\log \left(P(y \mid X)\right) = \log(\phi_i) - \frac{1}{2}\log(|\Sigma_i|) - \frac{1}{2}(x^T - \mu_i)^T \, {\Sigma_i}^{-1} \, (x^T - \mu_i)$$

معادله مرز تصمیم گیری نیز به صورت زیر در می آید:

$$\begin{aligned} & log\,\frac{\phi_i}{\phi_j} - \frac{1}{2} log\,\frac{|\Sigma_i|}{|\Sigma_j|} - \frac{1}{2}{\mu_i}^T\, {\Sigma_i}^{-1}\mu_i + \frac{1}{2}{\mu_j}^T\, {\Sigma_i}^{-1}\mu_j && \approx c \\ & + x^T \left({\Sigma_i}^{-1}\,\mu_i - {\Sigma_j}^{-1}\mu_j\right) && \approx b \\ & -\frac{1}{2}\, x^T({\Sigma_i}^{-1} - {\Sigma_j}^{-1})x \; = 0 && \approx a \end{aligned}$$

در نهایت با باز کردن فرمول قبل با x1 ,x2 داریم:

$$c' + b_1'\, x_1 + b_2'\, x_2 + \; a_{00}'\, {x_1}^2 + (a_{10}' + a_{01}')x_1 x_2 + \; a_{11}'\, {x_2}^2 = 0$$

کلاس QDA

این کلاس همانند کلاس LDA پیاده سازی شده و همان توابعی که قبلا تعریف کردیم را اینجا داریم. تنها تفاوت کمی در فرمول های پیاده سازی وجود دارد که آن را تغییر داده ایم. هم چنین برای ماتریس کوواریانس نیز به ازای هرکلاس، ماتریس جداگانه محاسبه کرده ایم.

در تابع decision_boundry همان فرمول جدید بدست آمده را حساب کرده و به تفکیک مقادیر زیر را خروجی داده ایم.

$$c', b_1', b_2', a_{00}', a_{01}', a_{10}', a_{11}'$$

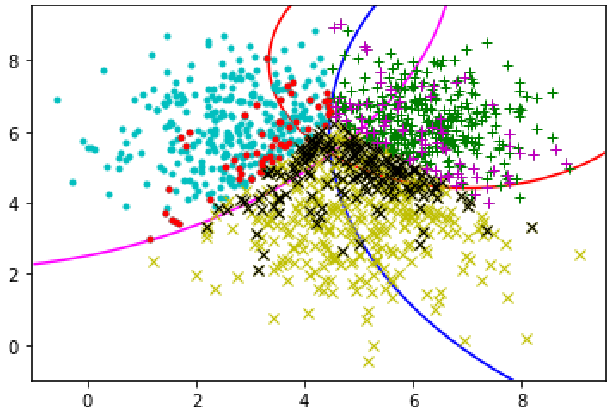
درتابع plot_decision با رسم همه نقاط داده با رنگ ها و شکل های متفاوت کلاس حدس زده شده و کلاس واقعی را نشان داده ایم. سپس به ازای هر جفت کلاس یک مرز تصمیم گیری از 1- تا 10 را رسم کرده ایم.

تابع plot_pdf نیز همانند قبل است تنها باید مرزتصمیم گیری را تغییر داد.

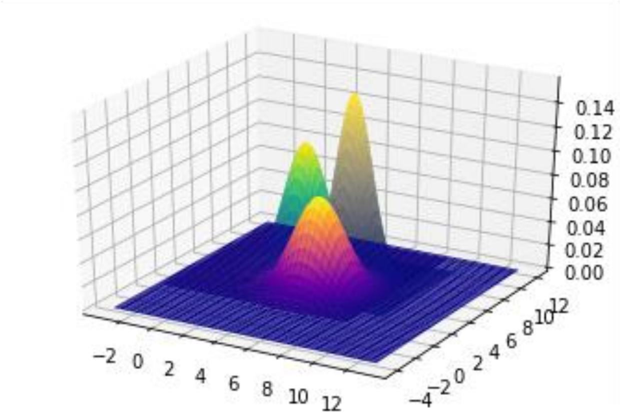
با داده آموزش دیتاست اول داریم:

```
classifier = QDA()
classifier.fit(dataset_1.x_train, dataset_1.y_train)

# train
classifier.plot_decision(dataset_1.x_train, dataset_1.y_train)

classifier.plot_pdf(dataset_1.x_train, dataset_1.y_train, classifier.mean_MLE)

pred = classifier.predict(dataset_1.x_train)

print('Train:')
classifier.compute_confusion_matrix(pred, dataset_1.y_train)
print('accuracy: ',classifier.accuracy_metric(pred, dataset_1.y_train))
print('precison: ',classifier.precision_metric(pred, dataset_1.y_train))
print('recall: ',classifier.recall_metric(pred, dataset_1.y_train))
print('F1: ',classifier.F1_metric(pred, dataset_1.y_train))

print("~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.")
print("~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.")
#test
pred = classifier.predict(dataset_1.x_test)
print('Test')
classifier.compute_confusion_matrix(pred, dataset_1.y_test)
print('accuracy: ',classifier.accuracy_metric(pred, dataset_1.y_test))
print('precison: ',classifier.precision_metric(pred, dataset_1.y_test))
print('recall: ',classifier.recall_metric(pred, dataset_1.y_test))
print('F1: ',classifier.F1_metric(pred, dataset_1.y_test))
```

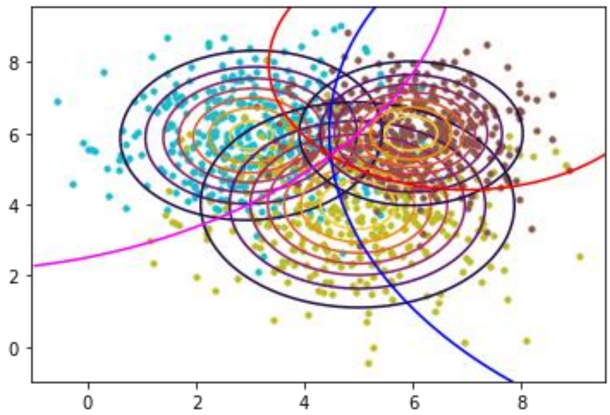
Train:
accuracy: 75.25
precison: [0.84, 0.65, 0.8]
recall: [0.77, 0.77, 0.72]
F1: [0.8, 0.7, 0.76]
~.
~.
Test
accuracy: 78.66666666666667
precison: [0.93, 0.7, 0.75]
recall: [0.82, 0.78, 0.76]
F1: [0.87, 0.74, 0.75]



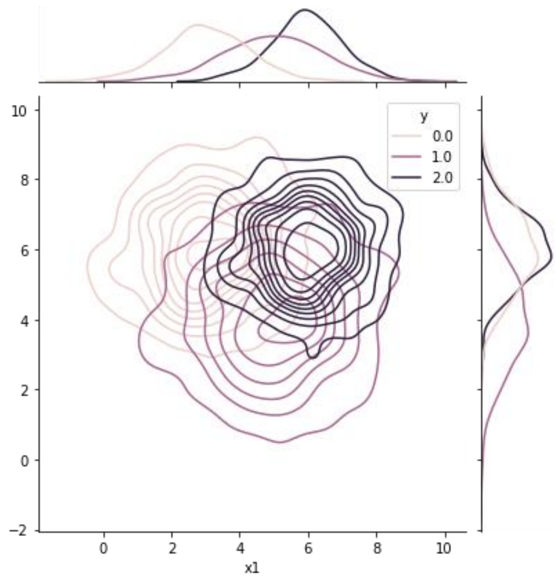
نمودار Decision Bound دیتاست 1



نمودارPDF دیتاست 1



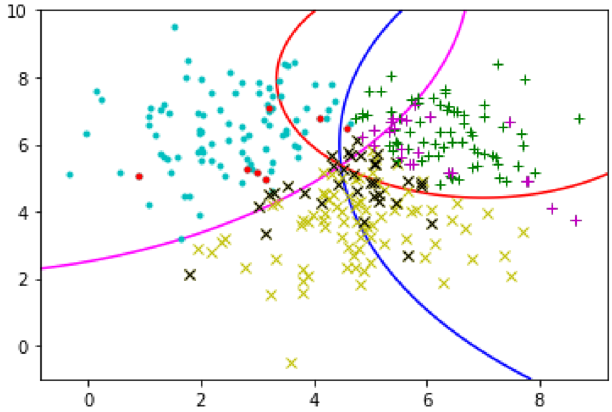
نمودار Contour دیتاست 1



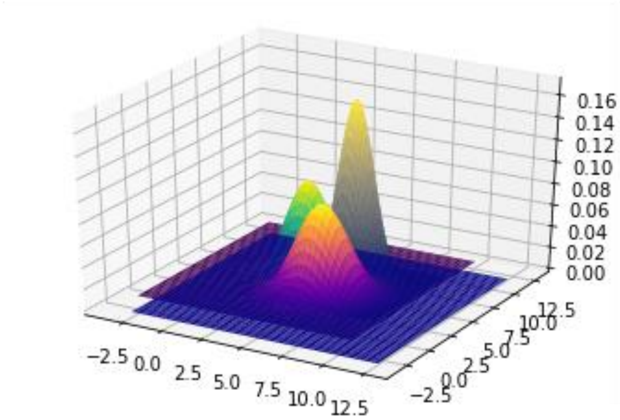
نمودار Contour دیتاست 1

با داده تست دیتاست اول داریم:

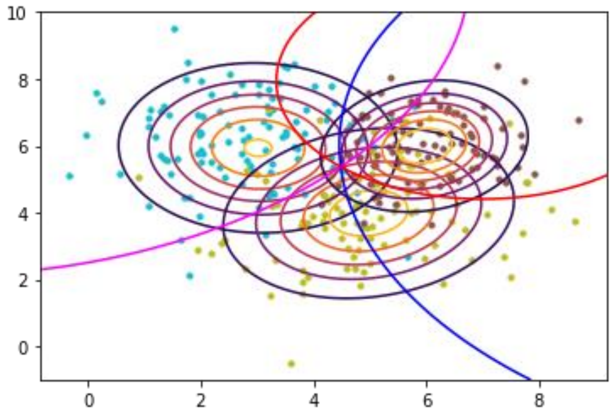
Test
accuracy: 78.66666666666667
precison: [0.93, 0.7, 0.75]
recall: [0.82, 0.78, 0.76]
F1: [0.87, 0.74, 0.75]



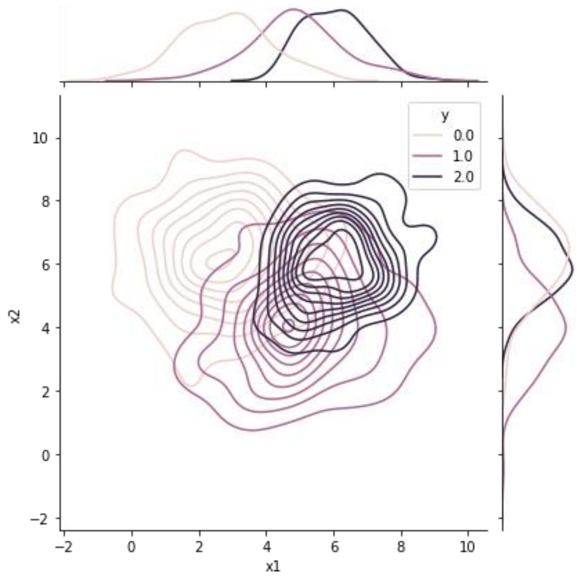
نمودار Decision Bound دیتاست تست 1



نمودار PDF دیتاست تست 1



نمودار Contour دیتاست تست 1

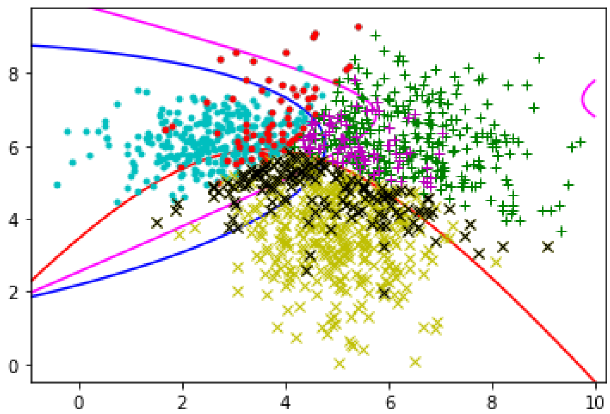


نمودار Contour دیتاست تست 1

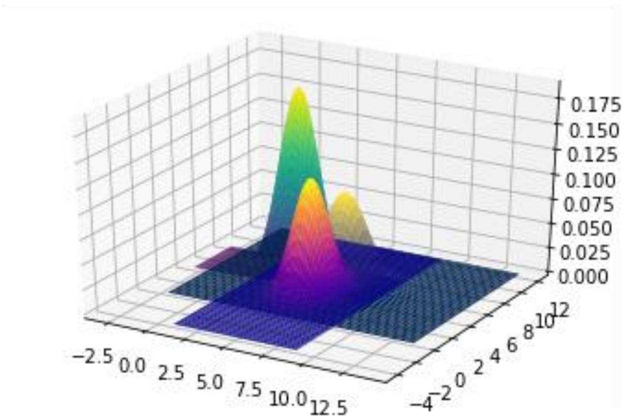
با داده آموزش دیتاست دوم داریم:

```
classifier = QDA()
classifier.fit(dataset_2.x_train, dataset_2.y_train)

# train
classifier.plot_decision(dataset_2.x_train, dataset_2.y_train)

classifier.plot_pdf(dataset_2.x_train, dataset_2.y_train, classifier.mean_MLE)

pred = classifier.predict(dataset_2.x_train)

print('Train:')
classifier.compute_confusion_matrix(pred, dataset_2.y_train)
print('accuracy: ',classifier.accuracy_metric(pred, dataset_2.y_train))
print('precison: ',classifier.precision_metric(pred, dataset_2.y_train))
print('recall: ',classifier.recall_metric(pred, dataset_2.y_train))
print('F1: ',classifier.F1_metric(pred, dataset_2.y_train))

print("~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.")
print("~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.")
#test
pred = classifier.predict(dataset_2.x_test)
print('Test')
classifier.compute_confusion_matrix(pred, dataset_2.y_test)
print('accuracy: ',classifier.accuracy_metric(pred, dataset_2.y_test))
print('precison: ',classifier.precision_metric(pred, dataset_2.y_test))
print('recall: ',classifier.recall_metric(pred, dataset_2.y_test))
print('F1: ',classifier.F1_metric(pred, dataset_2.y_test))
```

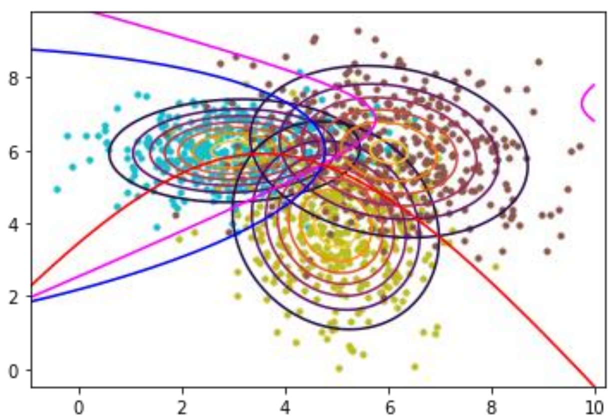
Train:
accuracy: 75.83333333333333
precison: [0.81, 0.7, 0.78]
recall: [0.74, 0.84, 0.7]
F1: [0.77, 0.77, 0.73]
~.
~.
Test
accuracy: 77.33333333333333
precison: [0.87, 0.77, 0.68]
recall: [0.78, 0.84, 0.69]
F1: [0.82, 0.8, 0.69]



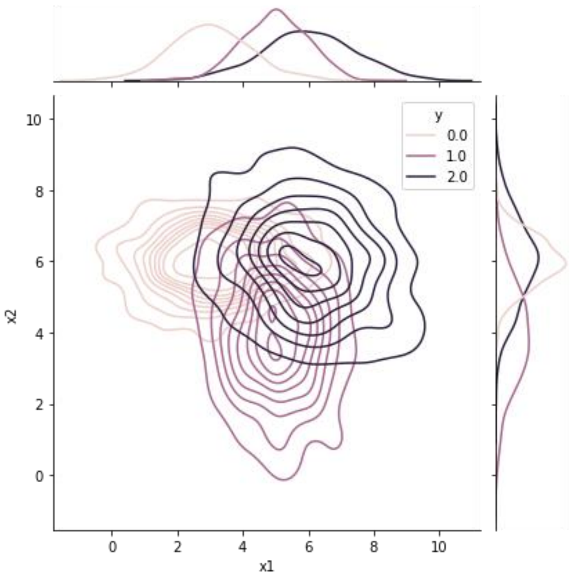
نمودار Decision Bound دیتاست 2



نمودار PDF دیتاست 2



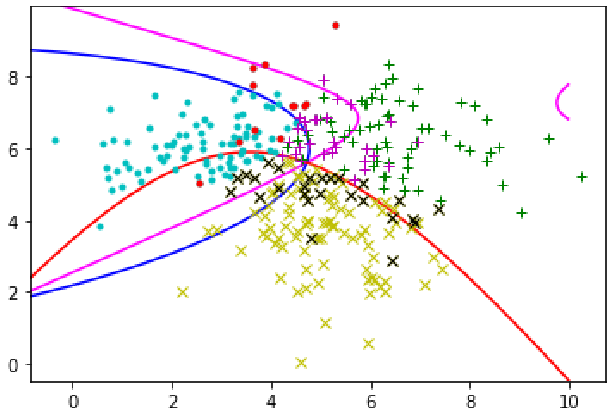
نمودار Contour دیتاست 2



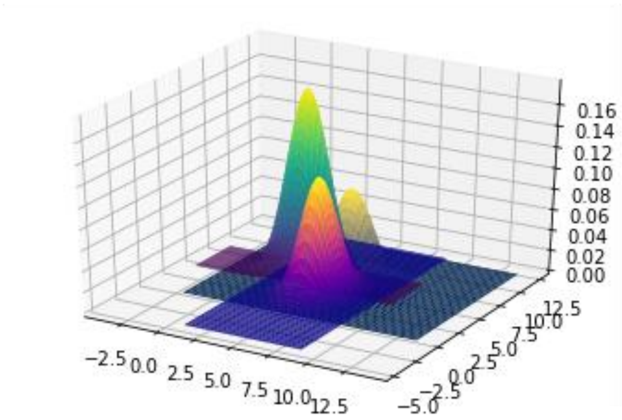
نمودار Contour دیتاست 2

با داده تست دیتاست دوم داریم:

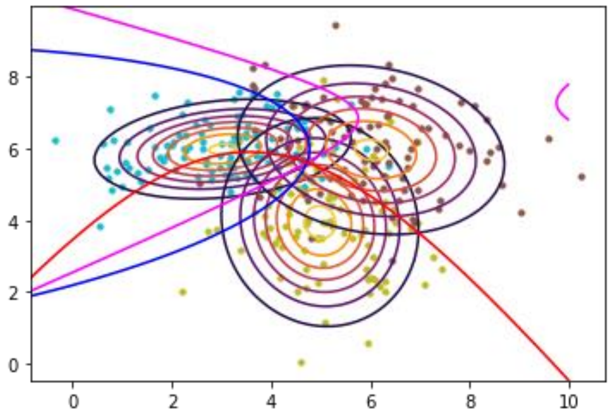
Test
accuracy: 77.33333333333333
precison: [0.87, 0.77, 0.68]
recall: [0.78, 0.84, 0.69]
F1: [0.82, 0.8, 0.69]



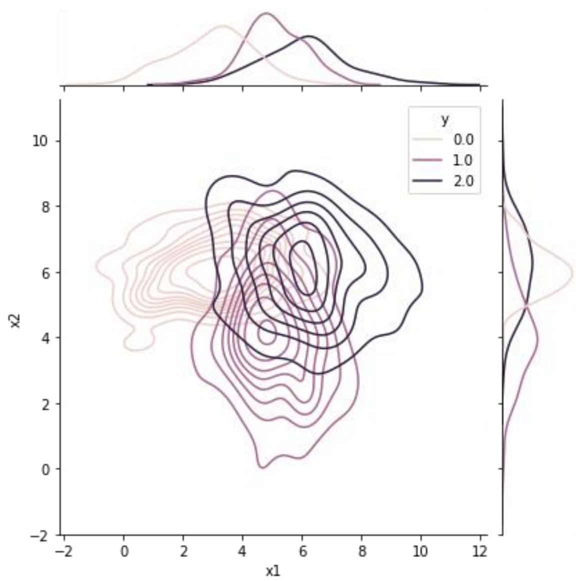
نمودار Decision Bound دیتاست تست 2



نمودار PDF دیتاست تست 2



نمودار Contour دیتاست تست 2



نمودار Contour دیتاست تست 2

مقایسه

تفاوت دیتاست که در این سوال ایجاد شده با دیتاست سوال قبلی در ماتریس های کوواریانس آن می باشد. در سوال قبل فرض کردیم ماتریس کوواریانس برای همه کلاس ها یکسان باشد و این باعث می شود که داده ها در کلاستر هایی به سایز و شکل یکسان حول میانگین شان قرار بگیرند. به همین دلیل رفتار داده ها یکسان و مرز تصمیم خطی است. در این سوال، کلاس ماتریس کوواریانس متفاوت دارند و مرز تصمیم دیگر خطی نیست. قطر فرعی نیز در دیتاست دوم صفر نبود و این یعنی فیچر ها با هم وابستگی دارند و مستقل نیستند.

به طور کل در ماتریس کوواریانس اگر قطر اصلی آن متفاوت باشد باعث می شود شکل Countor ما بیضوی و به سمت ویژگی ای که مقدار بزرگتری دارد کشیده شود. به دلیل صفر نبودن قطر فرعی کووریانس های دیتاست دوم دیدیم که Contour های بیضوی شکل کاملا در جهت افقی یا عمودی کشیده نشده-اند و کمی چرخش داشته اند. اما در دیتاست اول یا در سوال قبل این چرخش را به دلیل صفر بودن قطر فرعی نداریم و Contourها فقط در جهت افقی یا عمودی کشیده شده اند.

Naïve Bayes Classification

در این بخش می خواهیم یک classifier طراحی کنیم که جملات با تمایل مثبت و منفی را از هم جدا کند. برای این قسمت، از 3 مجموعه داده مربوط به نظرات مردم در 3 سایت imdb، ylep و amazon استفاده می کنیم.

ابتدا باید مجموعه ی داده را بررسی کنیم و روی متن، پیش پردازش انجام دهیم. با خواندن فایل مجموعه داده، تمام کلمات هر جمله از هم جدا می شود. بر روی همه ی جملات، 3 کار زیر را انجام می دهیم:

1) حذف stop words و commonly used words: برای این کار از کتابخانه ی nltk استفاده می کنیم.
2) حذف punctuation و کاراکتر های اضافی مثل <`,.<>!?: برای این کار از کتابخانه ی re یعنی regular expression ها در پایتون استفاده می کنیم.
3) Stem کردن کلمات: به این معنی است که هر کلمه را به ریشه ی خودش برگردانیم به این صورت که تمام کلماتی مثل program، programmer، programming، programs و ... به 'program' بازگردانده و ذخیره شود.

تمام این توابع در کلاس Dataset پیاده سازی شده اند و در preprocess فراخوانی می شوند.

اکنون به سراغ Naïve Bayes می رویم.

Likelihood

Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)\,P(c)}{P(x)}$$

Posterior Probability

Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

در این روش، می خواهیم با محاسبه ی prior یا همان p(c) و likelihood یا همان p(x | c)، p(c | x) را محاسبه کنیم. در عبارت بالا می توان از p(x) صرف نظر کرد.

نهایتا برای محاسبه ی کلاس x، از فرمول زیر استفاده می کنیم:

$$y^* = h_{NB}(\mathrm{x}) \; = \; \mathrm{arg}\max_y P(y)P(x_1,\ldots,x_n \mid y)$$
$$= \; \mathrm{arg}\max_y P(y)\prod_i P(x_i|y)$$

به عبارت دیگر، احتمال از هر کدام از کلاس ها بودن را محاسبه کرده و هر کدام بیشتر بود، به عنوان کلاس x، در نظر می گیریم. همان طور که گفته شد، p(y) یا prior، احتمال از کلاس j بودن را به ما می دهد. یعنی p(y=0) احتمال آنکه یک داده از کلاس 0 باشد و p(y=1) احتمال آنکه یک داده از کلاس 1 باشد را مشخص می کند. برای محاسبه ی prior، تعداد کل نمونه هایی از داده های train را که از کلاس j هستند به تعداد کل نمونه ها حساب می کنیم.

همچنین برای محاسبه ی likelihood یا p(x|c)، به ازای هر x یا feature، در اینجا feature ها همان کلمه ها هستند، برای هر c برابر می شود با تعداد تکرار آن کلمه در کلاس c تقسیم بر تعداد کل کلمات کلاس c.

:Laplace smoothing

به کمک این تکنیک، از بدست آوردن probability عضو یک کلاس بود برابر 0 جلوگیری می کنیم. در حالت هایی که کلمه ای تا به حال در داده های train دیده نشده باشد و یا در حداقل یکی از کلاس ها وجود نداشته باشد، حاصل product احتمال ها، صفر خواهد شد. برای جلوگیری از این مشکل، یک پارامتر alpha تعریف میکنیم و به همه ی کلمات، alpha را اضافه کرده تا هیچ گاه تعداد تکرار کلمه ای برابر صفر و در نتیجه prob=0 نشود.

:Log likelihood

در اینجا، از log likelihood به جای likelihood استفاده می کنیم. به این علت که حاصل ضرب تعدادی زیادی احتمال که اعدادی بین صفر و یک هستند، ممکن است باعث شود جواب نهایی خیلی کوچک شود و به سمت صفر میل کند، از تکنیک لگاریتم گیری استفاده میکنیم تا اعمال ضرب به جمع تبدیل شده و حاصل به صفر میل نکند.

نتیجه گیری Naïve Bayes:

فرض اصلی کلاسیفایر naïve bayes مستقل بودن ویژگی ها از هم دیگر و عدم تاثیر یک ویژگی بر ویژگی دیگر است. مزیت این فرض بر این است که میتوان فرمول محاسبه ی likelihood را ساده تر نوشت و داریم:

$$P(X|C_i) = P(x_1, x_2, \ldots, x_n | C_i)$$

$$= P(x_1|C_i) * P(x_2|C_i) * \ldots * P(x_n|C_i)$$

$$= \prod_{k=1}^{n} P(x_k|C_i)$$

خروجی Naïve Bayes:

```
Dataset yelp

<< data train >>
total accuracy:
96.375
accuracy on class-0:
96.7741935483871
accuracy on class-1:
95.96977329974811

<< data test >>
total accuracy:
79.5
accuracy on class-0:
74.22680412371135
accuracy on class-1:
84.46601941747574
```

```
Dataset amazon

<< data train >>
total accuracy:
96.25
accuracy on class-0:
93.71859296482413
accuracy on class-1:
98.75621890547264

<< data test >>
total accuracy:
84.5
accuracy on class-0:
76.47058823529412
accuracy on class-1:
92.85714285714286
```

```
Dataset imdb

<< data train >>
total accuracy:
97.0
accuracy on class-0:
97.68637532133675
accuracy on class-1:
96.35036496350365

<< data test >>
total accuracy:
87.0
accuracy on class-0:
95.49549549549549
accuracy on class-1:
76.40449438202248
```