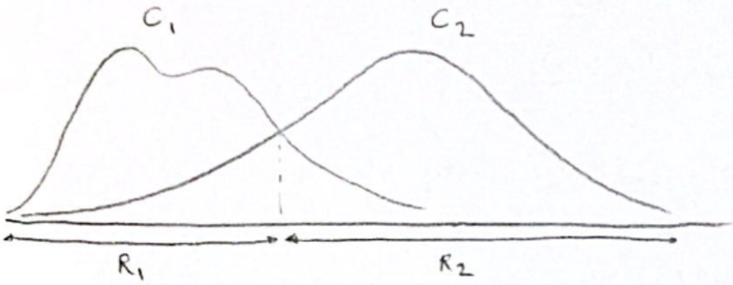
ثمین کبدی زاده

Hw1

① آ)

$a \leqslant b \xrightarrow{a,b>0} a^2 \leqslant ab \xrightarrow{a>0} |a| \leqslant (ab)^{1/2} \rightarrow a \leqslant (ab)^{1/2}$

ب)



$\rightarrow p(\text{error}) = p(x \in R_1, C_2) + p(x \in R_2, C_1) = \int_{R_1} p(x, C_2)\,dx + \int_{R_2} p(x, C_1)$

$p(x, C_2) \leqslant \{p(x, C_2)\, p(x, C_1)\}^{1/2}$

$p(x, C_1) \leqslant \{p(x, C_1)\, p(x, C_2)\}^{1/2}$

$\Rightarrow p(\text{error}) \leqslant \int_{R_1} \{p(x, C_2)\, p(x, C_1)\}^{1/2}\,dx + \int_{R_2} \{p(x, C_1)\, p(x, C_2)\}^{1/2}\,dx =$

$$\boxed{\int \{p(x, C_2)\, p(x, C_1)\}^{1/2}\,dx}$$

②

آ)

تعداد داده‌هایی که درست تشخیص داده شده است

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

تعداد کل داده‌ها

→ مدل چه بخشی از داده‌ها را به درستی تشخیص داده است.

تعداد داده‌ها در کلاس P که درست تشخیص داده شده است

$$precision = \frac{TP}{TP + FP}$$

تعداد کل داده‌هایی که توسط مدل P تشخیص داده شده است

→ چه بخشی از داده‌هایی که مدل آن را P تشخیص داده واقعا در کلاس P بوده اند

تعداد داده‌ها در کلاس P که درست تشخیص داده شده اند

$$Recall = \frac{TP}{TP + FN}$$

تعداد کل داده‌هایی که واقعا در کلاس P بوده اند

→ مدل چه بخشی از داده‌های کلاس P را به درستی تشخیص داده است.

ب) می‌دانیم هزینه‌ی این که فرد بیمار را سالم تشخیص دهیم بیشتر از آن است که فرد سالم را بیمار تشخیص دهیم بنابراین برای ما مهم است که مدل دانسته باشیم که برای افراد بیمار با درصد بالایی درست پاسخ دهد $\frac{TP}{TP + FN}$ (تشخیص درست / کل بیماران) بنابراین در این جا Recall معیار مناسبی است اگرچه دقت مدل هم برای ما اهمیت دارد و باید سعی کنیم افرادی که واقعا بیمار نیستند هم بیمار در نظر نگیریم اما در این جا با توجه به هزینه Recall برای ما مهم‌تر است.

ج) discriminability معیاری است که به ما نشان می‌دهد که توزیع‌ها چه قدر از هم فاصله دارند و متمایز هستند و تعریف آن برای دو متغیر تصادفی به صورت زیر است:

$$d' = \frac{|\mu_2 - \mu_1|}{\sigma}$$

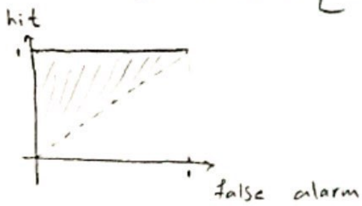
→ با افزایش فاصله میان میانگین‌های اول در توزیع و کاهش واریانس دو آن‌ها این تمایز بیشتر است.

۴) در classification، هدف این است که یک mapping از ویژگی‌ها (X) به خروجی‌ها که همان کلاس‌ها (ω) هستند پیدا کنیم به طوری که تعدادی از جفت‌های ورودی خروجی را دیده باشیم.

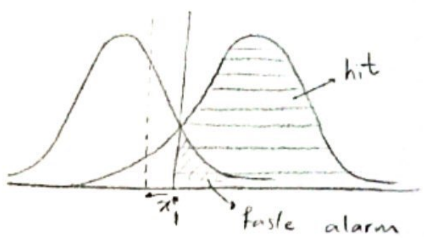
در روش generative تابع احتمال توأم متغیرهای ویژگی و کلاس ($p(x,\omega)$) را به دست آورده و سپس از روی آن توابع شرطی ($p(\omega_j|k)$) را جهت ترکیب مدل به دست می‌آوریم. مشکل این روش این است که برای به دست آوردن تابع توزیع توأم تعداد نمونه‌های زیادی می‌خواهیم و با مشکل کمبود داده مواجه هستیم.

در روش discriminative یک فرم از توابع شرطی ($p(\omega_j|k)$) را تخمین می‌زنیم. در این روش اطلاعات کمتری داریم ولی برای ساخت مدل به تعداد نمونه‌های کمتری احتیاج داریم و مدل به دست آمده ساده‌تر خواهد بود.

۵) می‌دانیم هرچه مساحت زیر نمودار ROC بیشتر باشد discriminability بالاتر است و تفکیک بهتری انجام شده است. یک مدل ایده‌آل مدلی است که مقدار hit یا true positive برای آن یک باشد. در واقع در مدل ایده‌آل مساحت زیر نمودار منحنی ROC برابر ۱ می‌شود:



صعودی بودن نمودار به این معنی است که همواره با افزایش false alarm، hit نیز افزایش می‌یابد. در واقع اگر به شکل زیر دقت کنیم:



برای این که false alarm افزایش یابد باید x^* به سمت چپ حرکت کند که در این صورت مساحت زیر نمودار برای هر دو افزایش می‌یابد. → hit ↑ false alarm ↑

رنگ آبی : $\frac{2+8+4+2}{4} , \frac{5+4+1+(-1)}{4} = 4 , 2.25$

رنگ قرمز : $\frac{-3+-3+-3+1}{4} , \frac{9+4+4+(-1)}{4} = -2 , 4$

chebyshev Distance $\rightarrow \max(|x_1-x_2|, |y_1-y_2|)$

Manhattan Distance $\rightarrow |x_1-x_2| + |y_1-y_2|$

Euclidean Distance $\rightarrow \sqrt{(x_1-x_2)^2 + |y_1-y_2|^2}$

$\boxed{\text{Nearest Centroid}}$

• برای محاسبه ی ما صرفاً تنها نزدیک ترین را در نظر گرفته شده است چون نقاط دورتر قطعاً مقدار بیشتری دارند (برای الگوریتم KNN)

- chebyshev Distance :
  - قرمز $\max(|2-(-2)|, |2-4|) = 4$
  - آبی $\max(|2-4|, |2-2.25|) = 2$ ✓

- Manhattan Distance :
  - قرمز $|2-(-2)| + |2-4| = 6$
  - آبی $|2-4| + |2-2.25| = 2.25$ ✓

- Euclidean Distance
  - قرمز $\sqrt{(2-(-2))^2 + (2-4)^2} = \sqrt{9^2 + 2^2}$
  - آبی $\sqrt{(2-4)^2 + (2-2.25)^2} = \sqrt{2^2 + (0.25)^2}$ ✓

$\boxed{\text{آبی}}$

Nearest Neighbor

- Chebyshev Distance

قرمز $\max(|2-1|, |2-4|) = 2$ → x و y بیشترین را با فاصله یکسانی دارد و دیگر را حساب نمی‌کنیم

آبی $\max(|2-2|, |2-5|) = 3$ → همه نقاط به وضوح دورتر اند

$\max(|2-2|, |2-(-1)|) = 3$

$\max(|2-4|, |2-1|) = 2$

- Manhattan Distance:

قرمز $|2-1| + |2-4| = 3$ → همه نقاط به وضوح بیشتر از این نقطه فاصله دارند

آبی $|2-2| + |2-5| = 3$ → "

$|2-2| + |2-(-1)| = 3$

$|2-4| + |2-1| = 3$

- Euclidean Distance

قرمز $\sqrt{(2-1)^2 + (2-4)^2} = \sqrt{1^2 + 2^2}$ → "

آبی $\sqrt{(2-2)^2 + (2-5)^2} = \sqrt{3^2}$ → "

$\sqrt{(2-2)^2 + (2-(-1))^2} = \sqrt{3^2}$

$\sqrt{(2-4)^2 + (2-1)^2} = \sqrt{2^2 + 1^2}$

← مشاهده می‌شود که در این جا مقدار minimum برای هر دو کلاس آبی و قرمز یکسان است و مدل مجبور است به صورت رندم یکی از این کلاس‌ها را انتخاب کند

* در این سوال استفاده از الگوریتم nearest centroid مناسب‌تر بوده است.

Scanned with CamScanner

$$P(\mu|D) = \frac{P(D|\mu)P(\mu)}{\int P(D|\mu)p(\mu)\,d\mu} = \alpha \prod_{i=1}^{K} p(x_i|\mu)p(\mu)$$

α مستقل از μ است و آن را به عنوان یک ضریب بیرون می کشیم

$$= \alpha \prod_{i=1}^{K} \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \exp\left[-\frac{1}{2}(x_i-\mu)\Sigma^{-1}(x_i-\mu)^T\right]$$

$$\frac{1}{(2\pi)^{d/2}} |\Sigma_0|^{1/2} \exp\left[-\frac{1}{2}(\mu-\mu_0)\Sigma_0^{-1}(\mu-\mu_0)^T\right] \quad \alpha$$

$$\exp\left[-\frac{1}{2}\left\{\mu^T(n\Sigma^{-1}+\Sigma_0^{-1})\mu - 2\mu^T\left(\Sigma^{-1}\sum_{i=1}^{n} x_i + \Sigma_0^{-1}\mu_0\right)\right\}\right]$$

$$\rightarrow P(\mu|D) = N(\mu_K, \Sigma_K)$$

$$\boxed{\begin{aligned} \mu_K &= \Sigma_0\left(\Sigma_0+\tfrac{1}{n}\Sigma\right)^{-1}\hat{\mu}_K + \tfrac{1}{K}\Sigma\left(\Sigma_0+\tfrac{1}{K}\Sigma\right)^{-1}\mu_0 \\ \Sigma_K &= \tfrac{1}{K}\Sigma_0\left(\Sigma_0+\tfrac{1}{K}\Sigma\right)^{-1}\Sigma \end{aligned}}$$

۶-

آ)

برای حل معادله درجه ۳ به کمک حل معادله ی نرمال لازم است تا در ابتدا به یک معادله خطی برسیم. به این منظور به کمک PolynomialFeatures و fit کردن آن بر روی داده های موجود هر کدام از توان های ۱ تا ۳ ورودی(X) است را می سازیم:

حالت اولیه:

$$y' = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$$

پس از اعمال PolynomialFeatures:

$$y'' = \theta_3 x'_3 + \theta'_2 x'_2 + \theta'_1 x'_1 + \theta'_0 x'_0$$

با تولید ویژگی های جدید به یک معادله خطی می رسیم که می توان آن را به کمک روابط برای حل معادله نرمال حل کرد:

$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

- $\hat{\theta}$ is the model parametes $\theta_0, \theta_1, \ldots, \theta_n$
- Y_i is the ith target value among m targets
- $X = \begin{bmatrix} x^{(1)^T} \\ \ldots \\ x^{(m)^T} \end{bmatrix}$

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
```

In [44]:
```
x = np.linspace(-5, 5, num=20)
rng = np.random.default_rng(42)
y = - 0.5 *(x ** 3) + (2 * x**2) + x + 4
y_noisy = y + 5 * rng.normal(loc=0, scale=1, size=len(x))


poly_features = PolynomialFeatures(degree=3, include_bias=False)
x_poly = poly_features.fit_transform(x.reshape(-1, 1))

x_b = np.c_[np.ones((len(x_poly), 1)), x_poly]
theta_best = np.linalg.inv(x_b.T.dot(x_b)).dot(x_b.T).dot(y_noisy)
```

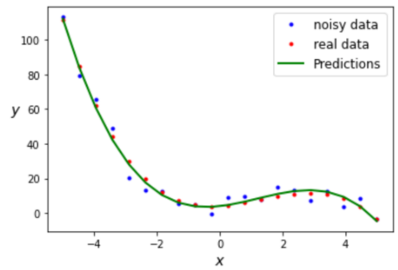
```
[ 3.9259314   2.03267046  1.99016431 -0.54790243]
```

In [45]:
```
theta_best
```

Out[45]:
```
array([ 3.9259314 ,  2.03267046,  1.99016431, -0.54790243])
```

حال با جایگذاری θ به دست آمده به معادله مورد نظر می رسیم که نمایش نمودار آن به شکل زیر است:

```python
In [76]: x_new = np.linspace(min(x),max(x), len(x)).reshape(len(x), 1) # linearly seperated points
x_new_poly = np.c_[np.ones((len(x), 1)),poly_features.transform(x_new)]
y_predict = x_new_poly.dot(theta_best)

plt.plot(x, y_noisy, "b.",label="noisy data")
plt.plot(x, y, "r.",label="real data")
plt.plot(x_new, y_predict, "g-", linewidth=2, label="Predictions")
plt.xlabel("$x$", fontsize=14)
plt.ylabel("$y$", rotation=0, fontsize=14)
plt.legend(loc="upper right", fontsize=12)
plt.show()
```



ب)

برای محاسبه تخمین نزول گرادیان لازم است تا گرادیان تابع هزینه محاسبه شود و در هر مرحله به کمک روابط موجود θ مرحله قبل را در جهتی که شیب به سمت هزینه کمتر است حرکت دهیم:

$$\frac{\partial}{\partial \theta_j}\mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

$$\nabla_{\boldsymbol{\theta}}\mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0}\mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1}\mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n}\mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m}\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathrm{MSE}(\boldsymbol{\theta})$$

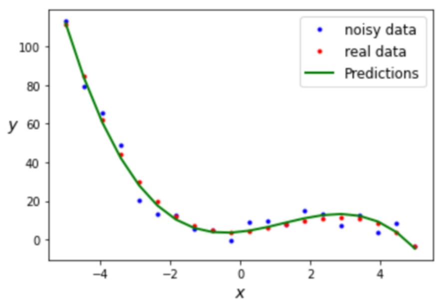
پس از این که به کمک روابط بالا و بعد از اجرای حدود 100000 گام به شکل زیر می رسیم که تا حد خوبی توانسته است تابع را تخمین بزند.

In [108]:
```python
eta = 0.0001  # learning rate
n_iterations = 100000
m = len(x)

theta = np.random.randn(4,1)  # random initialization
x_poly = np.c_[np.ones((len(x), 1)),poly_features.transform(x.reshape(len(x), 1))]

for iteration in range(n_iterations):
    gradients = 2/m * x_poly.T.dot(x_poly.dot(theta) - y_noisy.reshape(len(y), 1))
    theta = theta - eta * gradients
```

In [109]:
```python
theta
```

Out[109]:
```
array([[ 3.92522913],
       [ 2.03267046],
       [ 1.99020694],
       [-0.54790243]])
```

In [110]:
```python
x_new = np.linspace(min(x),max(x), len(x)).reshape(len(x), 1) # linearly seperated points
x_new_poly = np.c_[np.ones((len(x), 1)),poly_features.transform(x_new)]
y_predict = x_new_poly.dot(theta)

plt.plot(x, y_noisy, "b.",label="noisy data")
plt.plot(x, y, "r.",label="real data")
plt.plot(x_new, y_predict, "g-", linewidth=2, label="Predictions")
plt.xlabel("$x$", fontsize=14)
plt.ylabel("$y$", rotation=0, fontsize=14)
plt.legend(loc="upper right", fontsize=12)
plt.show()
```



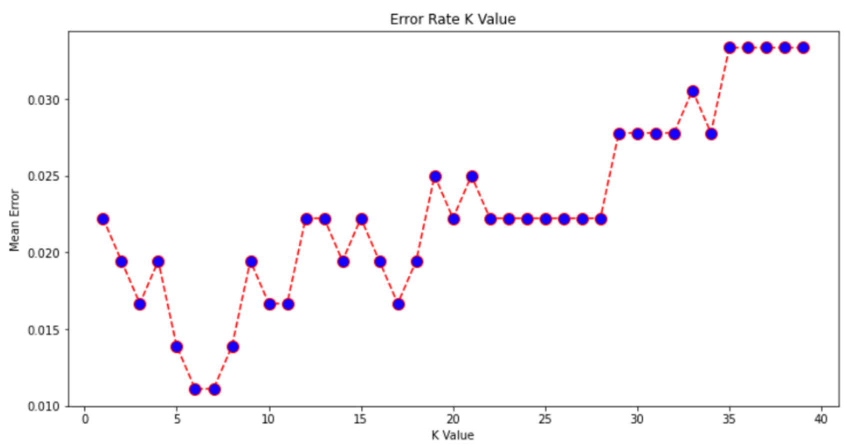
۸-

آ) برای اینکه مقدار k مناسب را به دست آوریم مدل KNN را بر روی بر روی تعداد همسایه ۱ تا ۴۰ اجرا می کنیم و مشاهده می شود که به ازای k=6 این الگوریتم خطای کمتری دارد. بنابراین در ادامه از تعداد همسایه ۶ استفاده شده است:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

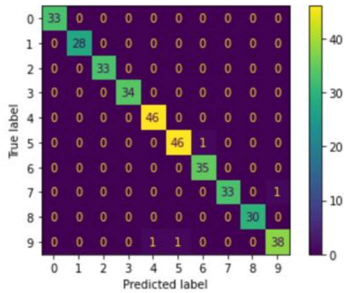
```
In [120]: error = []

          for i in range(1, 40):
              knn = KNeighborsClassifier(n_neighbors=i)
              knn.fit(x_train, y_train)
              pred_i = knn.predict(x_test)
              error.append(np.mean(pred_i != y_test))
```

```
In [121]: plt.figure(figsize=(12, 6))
          plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
                   markerfacecolor='blue', markersize=10)
          plt.title('Error Rate K Value')
          plt.xlabel('K Value')
          plt.ylabel('Mean Error')

Out[121]: Text(0, 0.5, 'Mean Error')
```



سپس یک طبقه بند $KNeighbors$ به وجود آورده و پس از فیت کردن داده های آموزش و تست $confusion\_matrix$ و $classification\_report$ را به دست می آوریم:

In [129]:
```
plot_confusion_matrix(KNN,x_test,y_test)
plt.show()
```



In [133]:
```
y_pred = KNN.predict(x_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       1.00      1.00      1.00        28
           2       1.00      1.00      1.00        33
           3       1.00      1.00      1.00        34
           4       0.98      1.00      0.99        46
           5       0.98      0.98      0.98        47
           6       0.97      1.00      0.99        35
           7       1.00      0.97      0.99        34
           8       1.00      1.00      1.00        30
           9       0.97      0.95      0.96        40

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

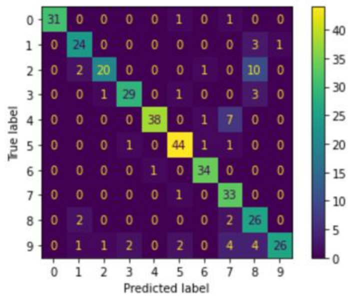
ب)

مشابه قسمت قبل اما این بار به کمک طبقه بند $GaussianNB$ مدل را بر روی داده های تست و آموزش fit کرده و مقادیر خواسته شده را نشان می دهیم:

In [136]:
```
gaussian = GaussianNB()
gaussian.fit(x_train,y_train)
```

Out[136]: `GaussianNB()`

In [137]:
```
plot_confusion_matrix(gaussian,x_test,y_test)
plt.show()
```



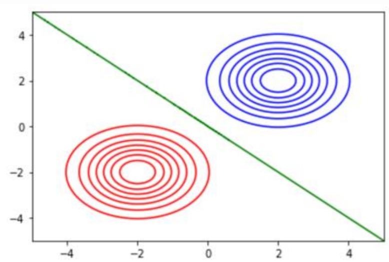
In [138]:
```
y_pred = gaussian.predict(x_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        33
           1       0.83      0.86      0.84        28
           2       0.91      0.61      0.73        33
           3       0.91      0.85      0.88        34
           4       0.97      0.83      0.89        46
           5       0.90      0.94      0.92        47
           6       0.92      0.97      0.94        35
           7       0.69      0.97      0.80        34
           8       0.57      0.87      0.68        30
           9       0.96      0.65      0.78        40

    accuracy                           0.85       360
   macro avg       0.86      0.85      0.84       360
weighted avg       0.88      0.85      0.85       360
```

-۹

در این سوال در ابتدا به کمک تابع *multivariate_normal* چهار تابع گاوسی به وجود آورده که دوتای اول(۱۱،۱۲) فاصله ی میانگین های بیشتری دارند و بنابرین بیشتر از یکدیگر متمایز اند و دوتای دومی(۲۱،۲۲) فاصله ی میانگین کمتری دارند و بیشتر در هم تنیده شده اند. مشاهده می شود که مرز تصمیم گیری در حالت اول از هر دو گاوسی فاصله دارد درحالی که در حالت دوم ممکن است داده هایی از هر کدام از گاوسی ها در سمت ناحیه در نظر گرفته شده برای کلاس مخالف نیز وجود دارد. لازم به ذکر است که در این سوال برای محاسبه مرز تصمیم باید مکان هایی انتخاب شود که در آن احتمال در نظر گرفته شده برای هر دو کلاس برابر است یا به عبارتی *normal1.pdf(pos)- normal2.pdf(pos)= 0* باشد که تنها کافی است کانتور آن را در سطح ۰ بکشیم تا خطی را نشان دهد که در آن مقدار این تابع برابر ۰ شده است که همان مرز تصمیم گیری است.

In [197]:
```
import matplotlib.pyplot as plt
import numpy as np
from   scipy.stats import multivariate_normal

u11 , s11 = [-2, -2],[[1, 0], [0, 1]]
u12 , s12 = [2, 2],[[1,0], [0, 1]]
normal11 = multivariate_normal(u11 , s11)
normal12 = multivariate_normal(u12 , s12)

N     = 200
X     = np.linspace(-5, 5, N)
Y     = np.linspace(-5, 5, N)
X, Y = np.meshgrid(X, Y)
pos  = np.dstack((X, Y))

plt.contour(X, Y, normal11.pdf(pos),colors='red')
plt.contour(X, Y, normal12.pdf(pos),colors='blue')
plt.contour(X, Y,normal11.pdf(pos)- normal12.pdf(pos),levels=[0],colors='green')
plt.show()
```



In [203]:
```
u21 , s21 = [0, 1],[[1, 0], [0, 1]]
u22 , s22 = [1, 2],[[1,0], [0, 1]]
normal21 = multivariate_normal(u21 , s21)
normal22 = multivariate_normal(u22 , s22)
plt.contour(X, Y, normal21.pdf(pos),colors='red')
plt.contour(X, Y, normal22.pdf(pos),colors='blue')
plt.contour(X, Y,normal21.pdf(pos)- normal22.pdf(pos),levels=[0],colors='green')
plt.show()
```