10-315 Introduction to Machine Learning (SCS Majors)
Lecture 6: Naive Bayes

Leila Wehbe
Carnegie Mellon University
Machine Learning Department

Reading:](http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf) [(http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf">)](http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf) Generative and Discriminative Classifiers by Tom Mitchell.

Lecture outcomes:

- Conditional Independence
- Naïve Bayes, Gaussian Naive Bayes
- Practical Examples

The Naïve Bayes Algorithm

Naïve Bayes assumes conditional independence of the X_i's:
$P(X_1, \ldots, X_d|Y) = \prod_i P(X_i|, Y)$

(more on this assumption soon!)

Using Bayes rule with that assumption:

$$P(Y = y_k|X_1, \ldots, X_d) = \frac{P(Y = y_k) \prod_i P(X_i|Y = y_k)}{P(X)}$$

- Train the algorithm (estimate $P(X_i|Y = y_k)$ and $P(Y = y_k)$)
- To classify, pick the most probable Y^{new} for a new sample $X^{\text{new}} = (X_1^{\text{new}}, X_2^{\text{new}}, \ldots, X_d^{\text{new}})$ as:

$$Y^{\text{new}} \leftarrow \underset{y_k}{\operatorname{argmax}}\ P(Y = y_k) \prod_i P(X_i^{\text{new}}|Y = y_k)$$

Naïve Bayes - Training and Prediction Phase - Discrete X_i

Training:

- Estimate $\pi_k \equiv P(Y = y_k)$, get $\hat{\pi}_k$
- Estimate $\theta_{ijk} \equiv P(X_i = x_{ij} | Y = y_k)$, get $\hat{\theta}_{ijk}$
    - θ_{ijk} is estimate for each label y_k:
        - For each variable X_i:
            - For each value x_{ij} that X_i can take.

- Prediction: Classify Y^{new}

$$
\begin{aligned}
Y^{\text{new}} &= \underset{y_k}{\operatorname{argmax}}\ P(Y = y_k) \prod_i P(X_i^{\text{new}} = x_j^{\text{new}} | Y = y_k) \\
&= \underset{y_k}{\operatorname{argmax}}\ \pi_k \prod_i \theta_{i, X_i^{\text{new}}, k}
\end{aligned}
$$

But... how do we estimate these parameters?

Naïve Bayes - Training Phase - Discrete X_i - Maximum (Conditional) Likelihood Estimation

$P(X|Y = y_k)$ has parameters θ_{ijk}, one for each value x_{ij} of each X_i.

$P(Y)$ has parameters π.

To follow the MLE principle, we pick the parameters π and θ that maximizes the (**conditional**) likelihood of the data given the parameters.

To estimate:

- Compute

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D(Y = y_k)}{|D|}$$

- For each label y_k:
    - For each variable X_i:
        - For each value x_{ij} that X_i can take, compute:

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij}|Y = y_k) = \frac{\#D(X_i = x_{ij} \wedge Y = y_k)}{\#D(Y = y_k)}$$

.

Naïve Bayes - Training Phase - Discrete $\boldsymbol{X_i}$

Method 1: Maximum (Conditional) Likelihood Estimation

$P(X|Y = y_k)$ has parameters θ_{ijk}, one for each value x_{ij} of each X_i.

To follow the MLE principle, we pick the parameters θ that maximizes the **conditional** likelihood of the data given the parameters.

Method 2: Maximum A Posteriori Probability Estimation

To follow the MAP principle, pick the parameters θ with maximum posterior probability given the conditional likelihood of the data and the prior on θ.

Naïve Bayes - Training Phase - Discrete X_i

Method 1: Maximum (Conditional) Likelihood Estimation

To estimate:

- Compute
$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D(Y = y_k)}{|D|}$$
- For each label y_k:
  - For each variable X_i:
    - For each value x_{ij} that X_i can take, compute:
$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} | Y = y_k) = \frac{\#D(X_i = x_{ij} \wedge Y = y_k)}{\#D(Y = y_k)}.$$

Method 2: Maximum A Posteriori Probability Estimation (Beta or Dirichlet priors)

- K: the number of values Y can take
- J: the number of values X can take (we assume here that all X_j have the same number of possible values, but this can be changed)

- Example prior for π_k where K > 2:
  - $\text{Dirichlet}(\beta_\pi, \beta_\pi, . . , \beta_\pi)$ prior. (optionally, you can choose different values for each parameter to encode a different weighting).
  - if $K = 2$ this becomes a Beta prior.

- Example prior for θ_{ijk} where J>2 :
  - $\text{Dirichlet}(\beta_\theta, \beta_\theta, . . , \beta_\theta)$ prior. (optionally, you can choose different values for each parameter to encode a different weighting, you can choose a different prior per X_i or even per label y_k).
  - if $J = 2$ this becomes a Beta prior.

Method 2: Maximum A Posteriori Probability Estimation (Beta or Dirichlet priors)

- K: the number of values Y can take
- J: the number of values X can take

These priors will act as imaginary examples that smooth the estimated distributions and prevent zero values.

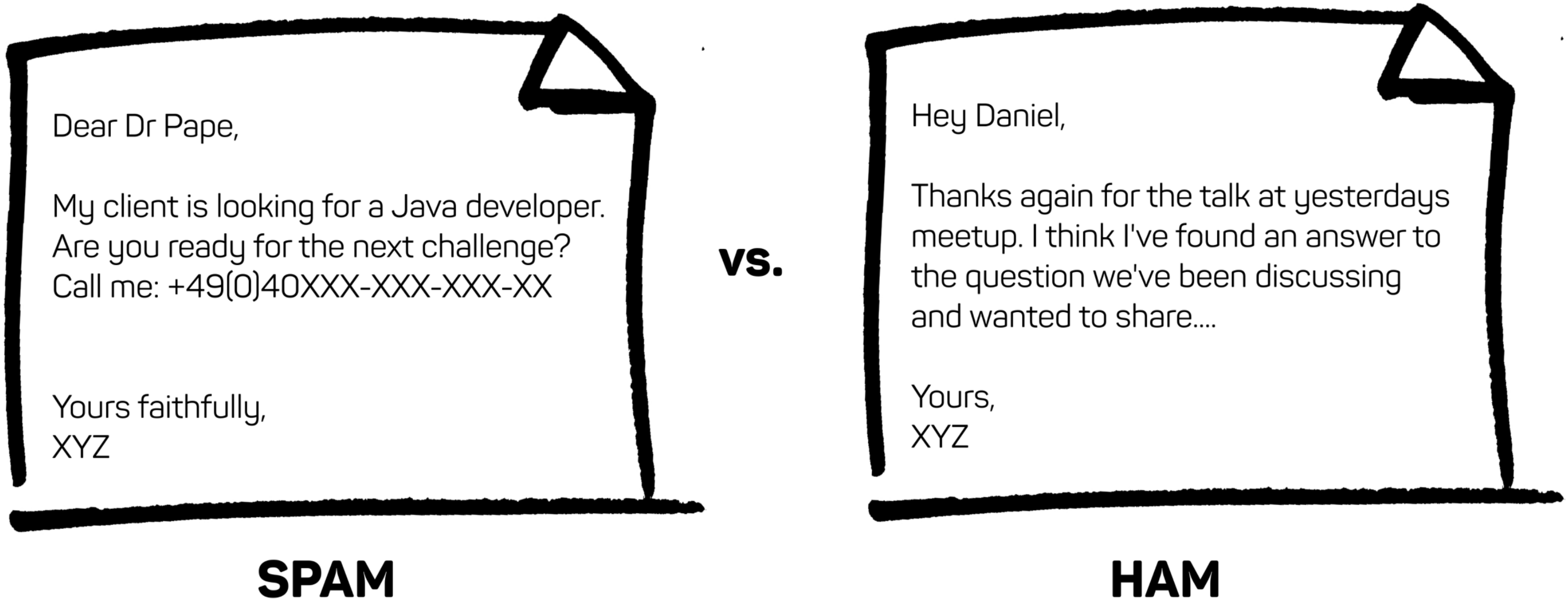
To estimate:

- Compute
$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D(Y = y_k) + (\beta_\pi - 1)}{|D| + K(\beta_\pi - 1)}$$
- For each label y_k:
  - For each variable X_i:
    - For each value x_{ij} that X_i can take, compute:
$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} | Y = y_k) = \frac{\#D(X_i = x_{ij} \wedge Y = y_k) + (\beta_\theta - 1)}{\#D(Y = y_k) + J(\beta_\theta - 1)}$$

.

Example: Text classification

- Classify which emails are spam?



[image by Daniel Pape (https://blog.codecentric.de/en/2016/06/spam-classification-using-sparks-dataframes-ml-zeppelin-part-1/)](https://blog.codecentric.de/en/2016/06/spam-classification-using-sparks-dataframes-ml-zeppelin-part-1/)

- Classify which emails promise an attachment?
- Classify which web pages are student home pages?

How shall we represent text documents for Naïve Bayes?

How can we express X?

- Y discrete valued. e.g., Spam or not
- X = ?

How can we express X?

- Y discrete valued. e.g., Spam or not
- $X = (X_1, X_2, \dots X_d)$ with d the number of words in English.
- (This is what we do in homework 2)

What are the limitations with this representation?

How can we express X?

- Y discrete valued. e.g., Spam or not
- $X = (X_1, X_2, \ldots X_d)$ with d the number of words in English.
- (This is what we do in homework 2)

What are the limitations with this representation?

- Some words always present
- Some words very infrequent
- Doesn’t count how often a word appears
- Conditional independence assumption is false…

Alternative Featurization

- Y discrete valued. e.g., Spam or not
- $X = (X_1, X_2, \ldots X_d)$ = document
- X_i is a random variable describing the word at position i in the document
- possible values for X_i : any word w_k in English
- X_i represents the ith word position in document
- X_1 = “I”, X_2 = “am”, X_3 = “pleased”

How many parameters do we need to estimate $P(X|Y)$? (say 1000 words per document, 10000 words in english)

Alternative Featurization

- Y discrete valued. e.g., Spam or not
- $X = (X_1, X_2, \ldots X_d)$ = document
- X_i is a random variable describing the word at position i in the document

- possible values for X_i : any word w_k in English
- X_i represents the ith word position in document
- X_1 = "I", X_2 = "am", X_3 = "pleased"

How many parameters do we need to estimate $P(X|Y)$? (say 1000 words per document, 10000 words in english)

Conditional Independence Assumption very useful:

- reduce problem to only computing $P(X_i|Y)$ for every X_i.

“Bag of Words” model

Additional assumption: position doesn't matter!

(this is not true of language, but can be a useful assumption for building a classifier)

- assume the Xi are IID:
  $P(X_i|Y) = P(X_j|Y) (\forall i, j)$
- we call this "Bag of Words"

Art installation in Gates building (now removed)

"Bag of Words" model

Additional assumption: position doesn't matter!

(this is not true of language, but can be a useful assumption for building a classifier)

- assume the Xi are IID:
  $P(X_i|Y) = P(X_j|Y)(\forall i, j)$
- we call this "Bag of Words"

Since all $\boldsymbol{X_i}$s have the same distribution, we only have to estimate one parameter per word, per class.

$\boldsymbol{P(X|Y = y_k)}$ **is a multinomial distribution:**

$$P(X|Y = y_k) \propto \theta_{1k}^{\alpha_{1k}} \theta_{2k}^{\alpha_{2k}} \dots \theta_{dk}^{\alpha_{dk}}$$

Review of distributions

$$P(X_i = w_j)\begin{cases} \theta_1, & \text{if } X_i = w_1 \\ \theta_2, & \text{if } X_i = w_2 \\ \dots \\ \theta_k, & \text{if } X_i = w_K \end{cases}$$

Probability of observing a document with α_1 count of w_1, α_2 count of w_2 ... is a multinomial:

$$\frac{|D|!}{\alpha_1! \cdots \alpha_J!}\theta_1^{\alpha_1}\theta_2^{\alpha_2}\theta_3^{\alpha_3} \cdots \theta_J^{\alpha_J}$$

Review of distributions

Dirichlet Prior examples:

- if constant across classes and words:
$$P(\theta) = \frac{\theta^{\beta_\theta}\theta^{\beta_\theta}, \ldots \theta^{\beta_\theta}}{\mathrm{Beta}(\beta_\theta, \beta_\theta, \ldots, \beta_\theta)}$$
- if constant across classes but different for different words:
$$P(\theta) = \frac{\theta^{\beta_1}\theta^{\beta_2}, \ldots \theta^{\beta_J}}{\mathrm{Beta}(\beta_1, \beta_2, \ldots, \beta_J)}$$
- if different for different classes k and words:
$$P(\theta_k) = \frac{\theta^{\beta_{1k}}\theta^{\beta_{2k}}, \ldots \theta^{\beta_{Jk}}}{\mathrm{Beta}(\beta_{1k}, \beta_{2k}, \ldots, \beta_{Jk})}$$

MAP estimates for Bag of words:

(Dirichlet is the conjugate prior for a multinomial likelihood function)

$$\theta_{jk} = \frac{\alpha_{jk} + \beta_{jk} - 1}{\sum_m (\alpha_{mk} + \beta_{mk} - 1)}$$

Again the prior acts like halucinated examples

What βs should we choose?

Example: Twenty NewsGroups

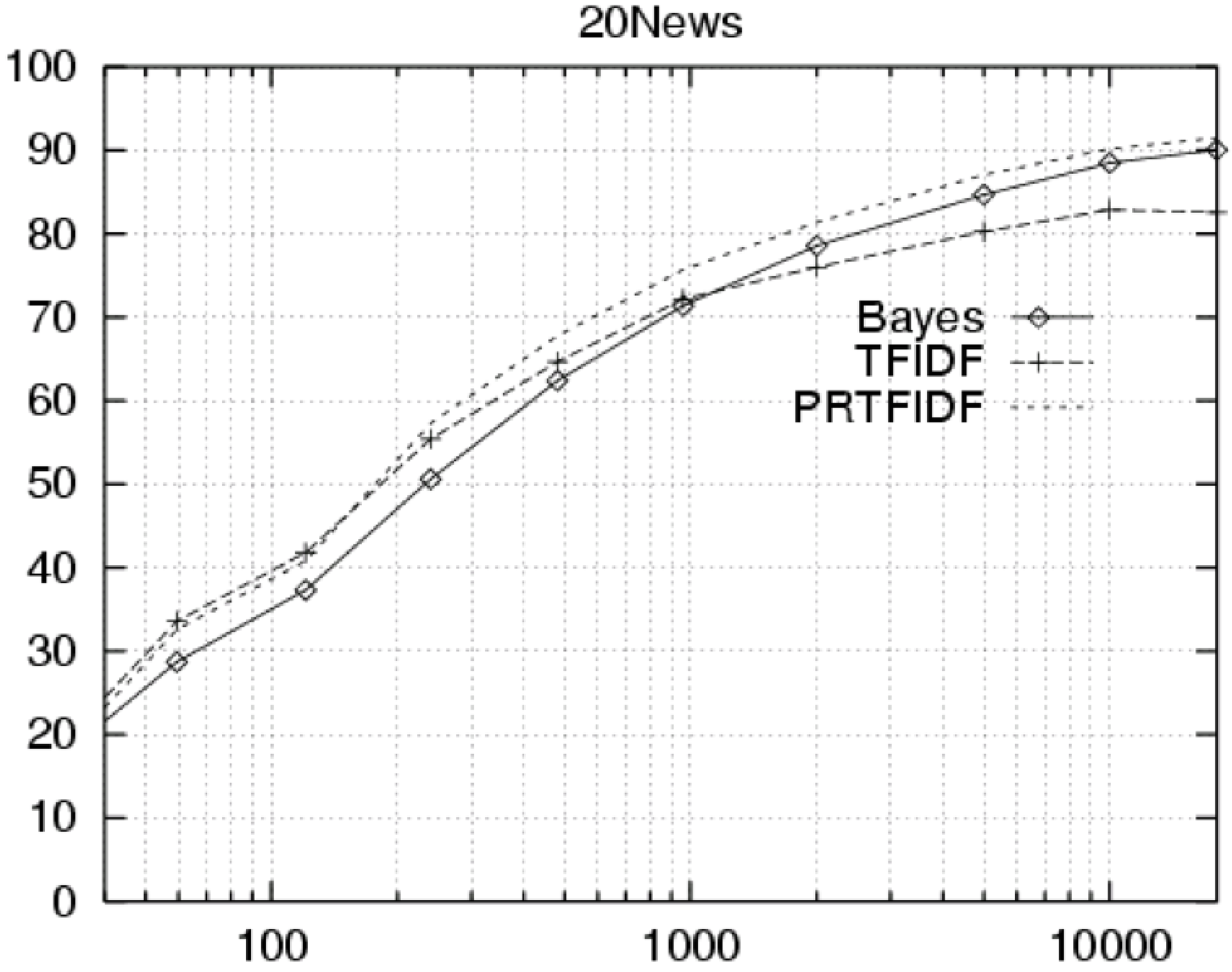
For code and data, see www.cs.cmu.edu/~tom/mlbook.html click on “Software and Data”.

Can group labels into groups that share priors:

- comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.max.hardware, comp.windows.x
- misc.forsale
- rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey
- alt.atheism,
- soc.religion.christian,
- talk.religion.misc, talk.politics.mideast, talk.politics.misc, talk.politics.guns,
- sci.space, sci.crypt, sci.electronics, sci.med

- Naïve Bayes: 89% classification accuracy

Learning curve for 20 Newsgroups



Accuracy vs. Training set size (1/3 withheld for test)

Even if incorrect assumption, performance can be very good

Even when taking half of the email

- Assumption doesn’t hurt the particular problem?
- Redundancy?
- Leads less examples to train? Converges faster to asymptotic performance? (Ng and Jordan)

Even if incorrect assumption, performance can be very good

Even when taking half of the email

- Assumption doesn't hurt the particular problem?
- Redundancy?
- Leads less examples to train? Converges faster to asymptotic performance? (Ng and Jordan)

More recently, algorithms such as LSTMs and Transformers have become very good

- are able to capture the sequential aspect of language and produce more complex representations.
- They do have many parameters, but nowhere as much as we mentioned before (10000^{1000}).

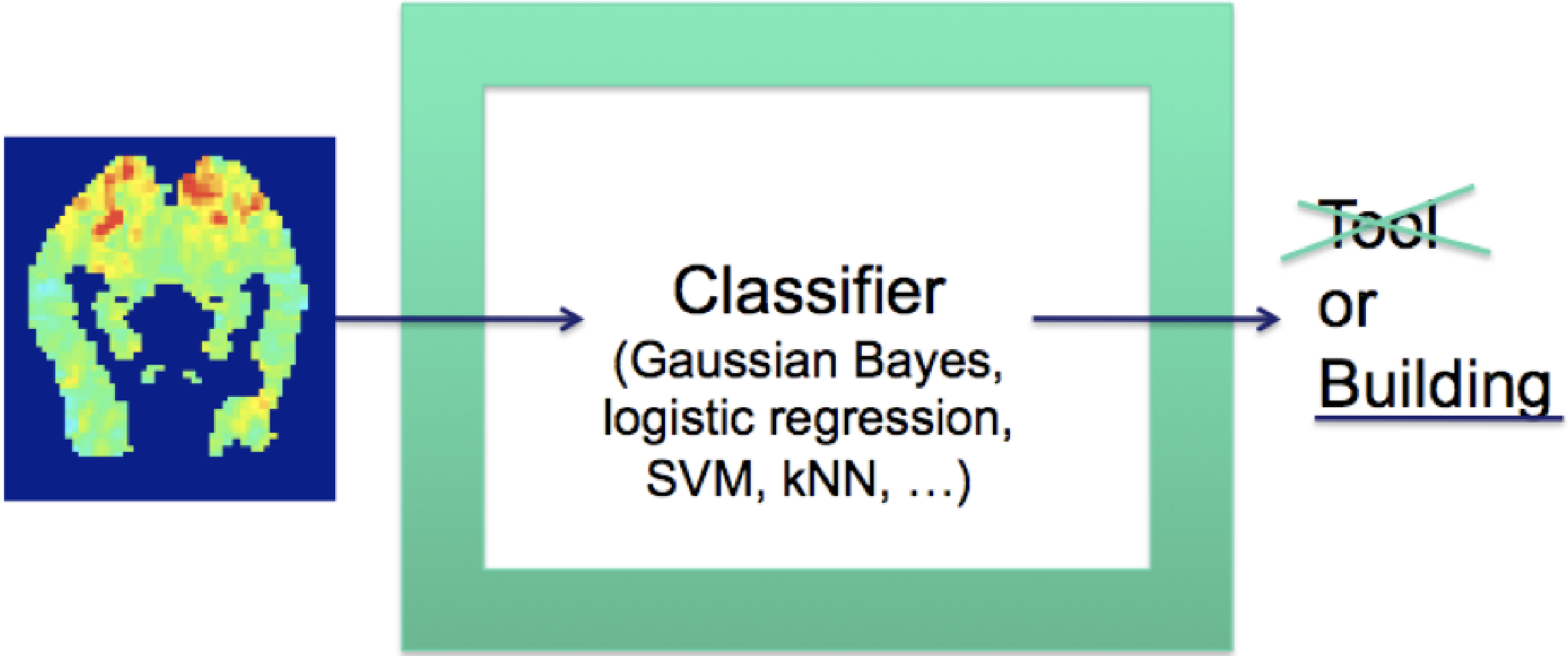
Continuous X_is

What can we do?

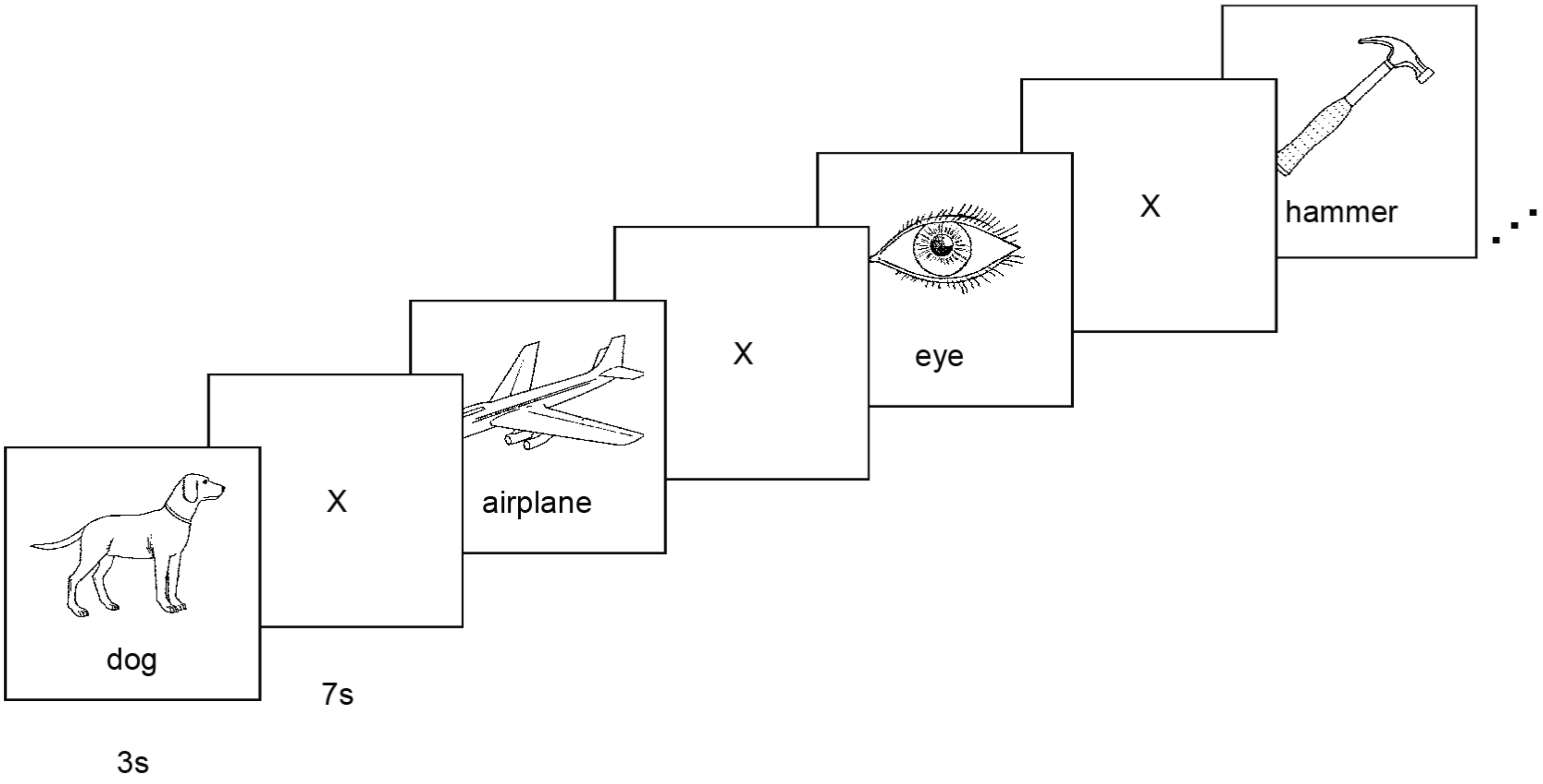
E.g. image classification, where X_i is real valued

Classify a person’s cognitive state, based on brain image

- reading a sentence or viewing a picture?
- reading the word describing a “Tool” or “Building”?
- answering the question, or getting confused?


Classifier
(Gaussian Bayes,
logistic regression,
SVM, kNN, …)
~~Tool~~
or
Building

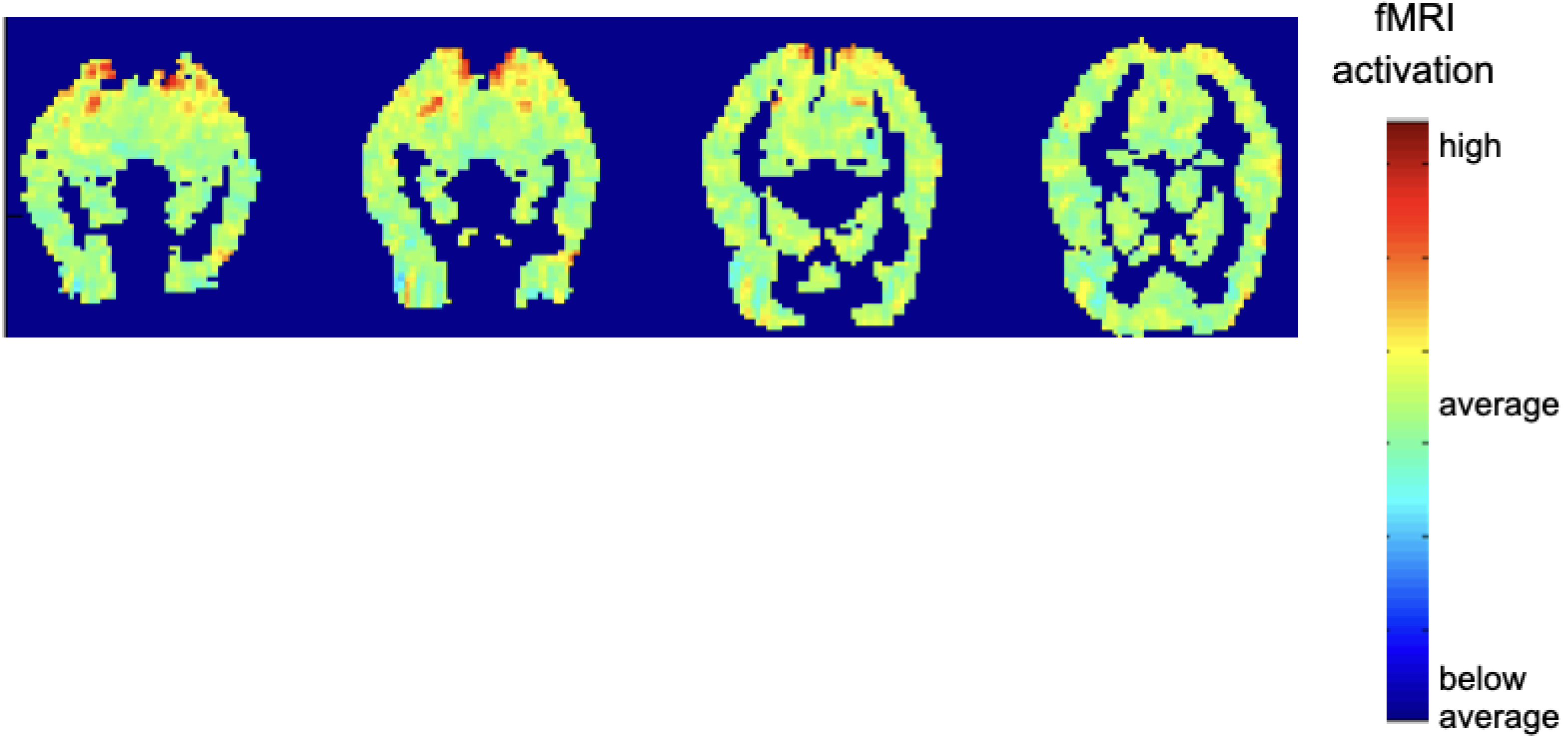
Stimulus for the study



60 distinct exemplars, presented 6 times each

[Mitchell et al. Science 2008 (https://science.sciencemag.org/content/320/5880/1191.abstract)](https://science.sciencemag.org/content/320/5880/1191.abstract), data available [online (https://www.cs.cmu.edu/afs/cs/project/theo-73/www/science2008/data.html)](https://www.cs.cmu.edu/afs/cs/project/theo-73/www/science2008/data.html).

Continuous X_i



Y is the mental state (reading “house” or “bottle”)

X_i are the voxel activities (voxel = volume pixel).

Continuous X_i

Naïve Bayes requires $P(X_i|Y = y_k)$ but X_i is continuous:

$$P(Y = y_k|X_1, \ldots, X_d) = \frac{P(Y = y_k) \prod_i P(X_i|Y = y_k)}{\sum_\ell (Y = y_\ell) \prod_i P(X_i|Y = y_\ell)}$$

What can we do?

Continuous X_i

Naïve Bayes requires $P(X_i|Y = y_k)$ but X_i is continuous:

$$P(Y = y_k|X_1, \ldots, X_d) = \frac{P(Y = y_k)\prod_i P(X_i|Y = y_k)}{\sum_\ell (Y = y_\ell)\prod_i P(X_i|Y = y_\ell)}$$

What can we do?

Common approach: assume $P(X_i|Y = y_k)$ follows a Normal (Gaussian) distribution

$$p(X_i = x|Y = y_k) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}}\exp\Big(-\frac{1}{2}\frac{(x_i - \mu_{ik})^2}{\sigma_{ik}^2}\Big)$$

Sometimes assume standard deviation

- is independent of Y (i.e., σ_i),
- or independent of Xi (i.e., σ_k)
- or both (i.e., σ)

Gaussian Naïve Bayes Algorithm – continuous X_i (but still discrete Y)

- Training:
  - Estimate $\pi_k \equiv P(Y = y_k)$
  - Each label y_k:
    - For each variable X_i estimate $P(X_i = x_{ij} | Y = y_k)$:
      - estimate class conditional mean μ_{ik} and standard deviation σ_{ik}

- Prediction: Classify Y^{new}

$$Y^{\text{new}} = \underset{y_k}{\operatorname{argmax}}\ P(Y = y_k) \prod_i P(X_i^{\text{new}} = x_j^{\text{new}} | Y = y_k)$$
$$= \underset{y_k}{\operatorname{argmax}}\ \pi_k \prod_i \mathcal{N}(X_i^{\text{new}}; \mu_{ik}, \sigma_{ik})$$

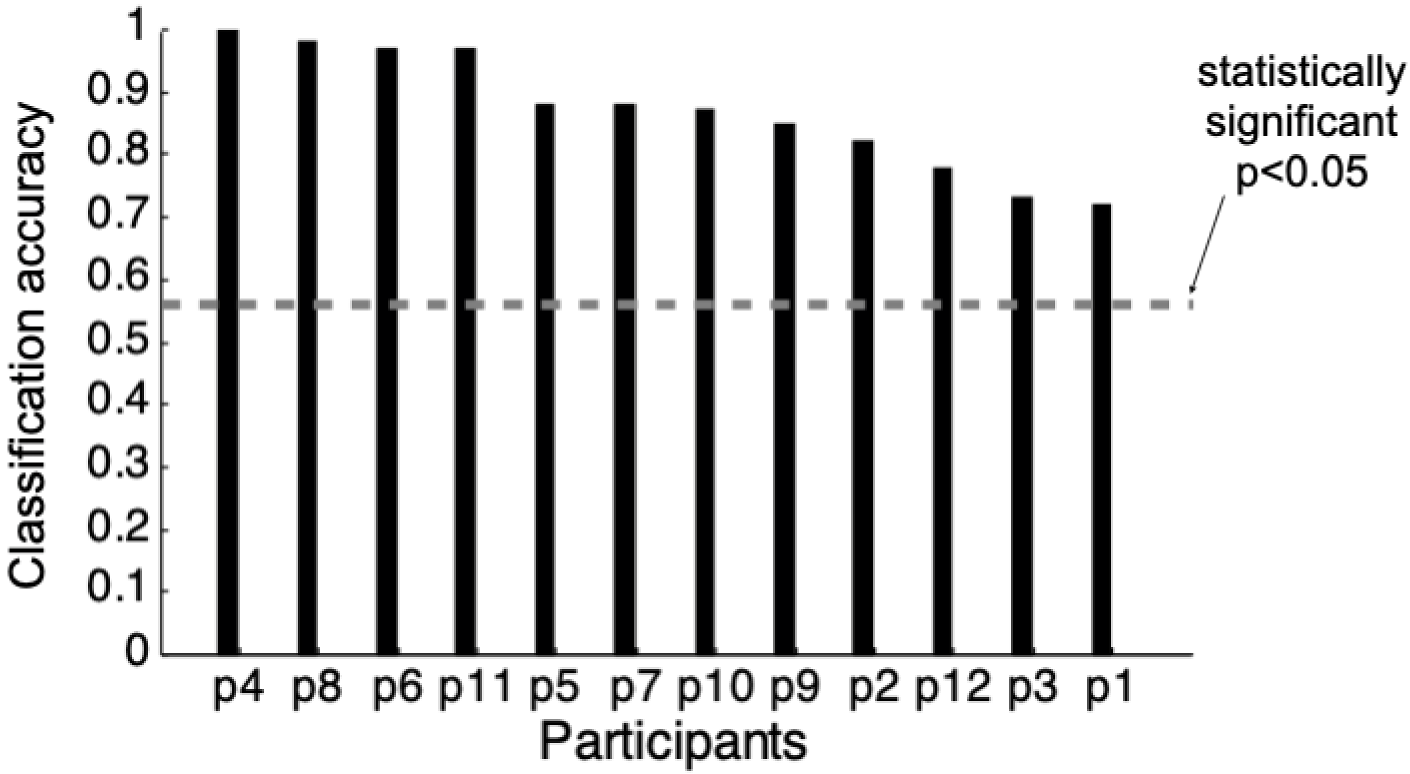
Estimating Parameters: Y discrete, X_i continuous

$$\hat{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta(Y^j = y_k)$$

- i: index of feature
- j: index of data point
- k: index of class
- δ function is 1 if argument is true and 0 otherwise

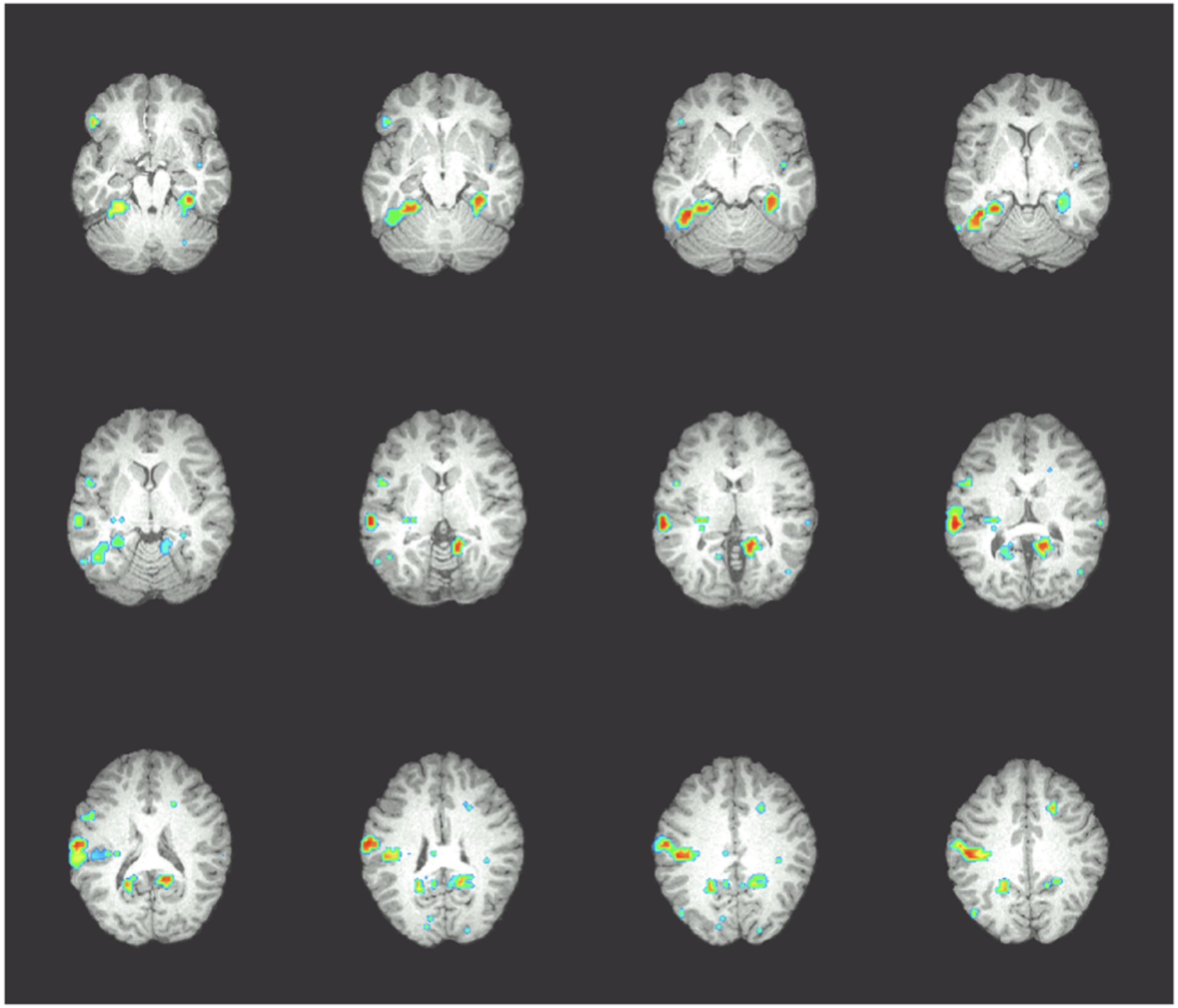
$$\hat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k)$$

Classification task: is person viewing a “tool” or “building”?



Where is information encoded in the brain?

Accuracies of cubical 27-voxel classifiers centered at each significant voxel [0.7-0.8]

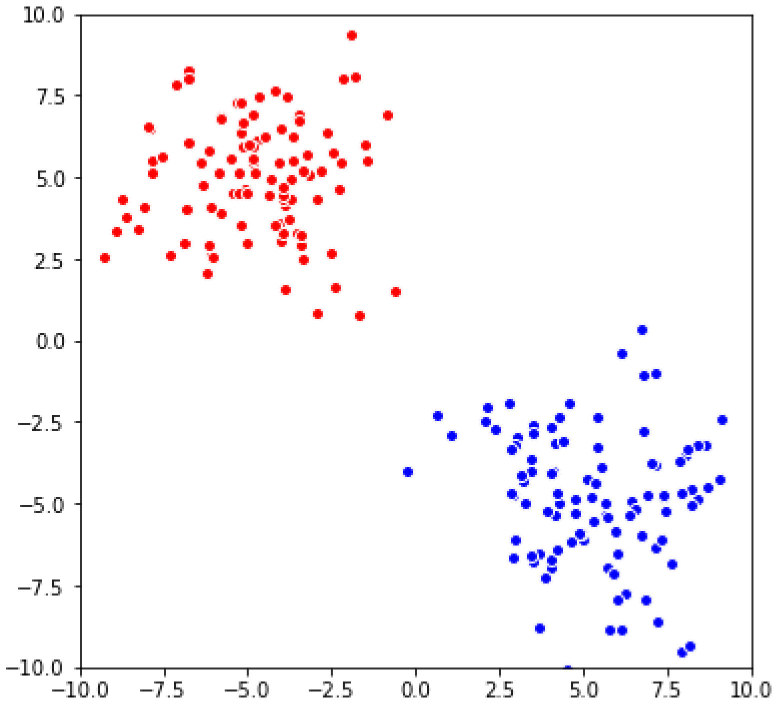


Let's simulate the behavior of GNB!

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import norm

x1 = np.linspace(-10,10,1000)
x2 = np.linspace(-10,10,1000)

# Assume I know the true parameters, this is not the case usually!
mu_1_1 = -5; sigma_1_1 = 2
mu_2_1 = 5; sigma_2_1 = 2
mu_1_0 = 5; sigma_1_0 = 2
mu_2_0 = -5; sigma_2_0 = 2

# Sample data from these distributions
X_positive = norm.rvs(loc=[mu_1_1,mu_2_1], scale=[sigma_1_1, sigma_2_1], size = (1
00,2))
X_negative = norm.rvs(loc=[mu_1_0,mu_2_0], scale=[sigma_1_0, sigma_2_0], size = (1
00,2))
```

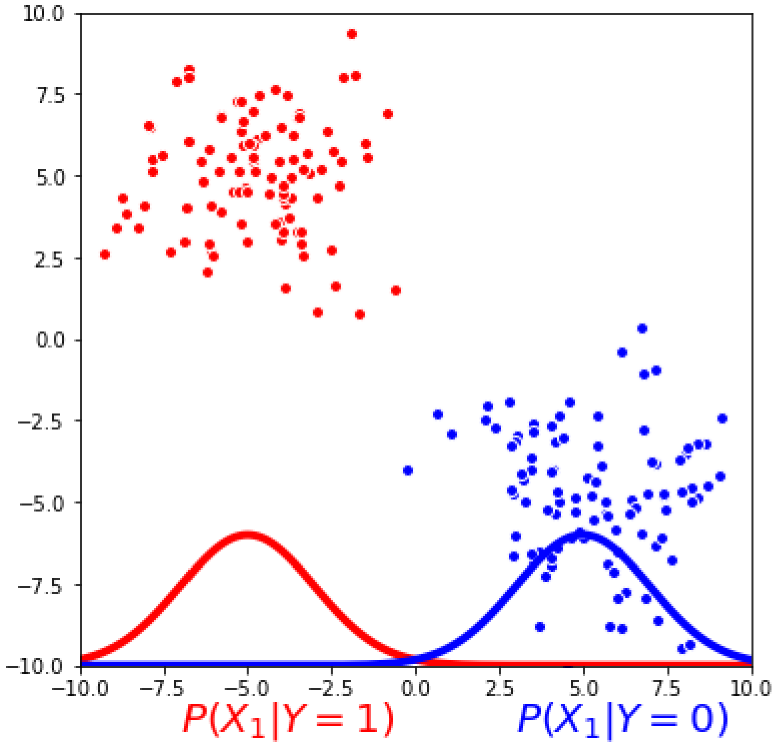
In [16]:

```
plt.figure(figsize=(6,6))
plt.scatter(X_positive[:, 0], X_positive[:, 1],facecolors='r', edgecolors='w')
plt.scatter(X_negative[:, 0], X_negative[:, 1],facecolors='b', edgecolors='w')
plt.axis([-10,10,-10,10],'equal');
```



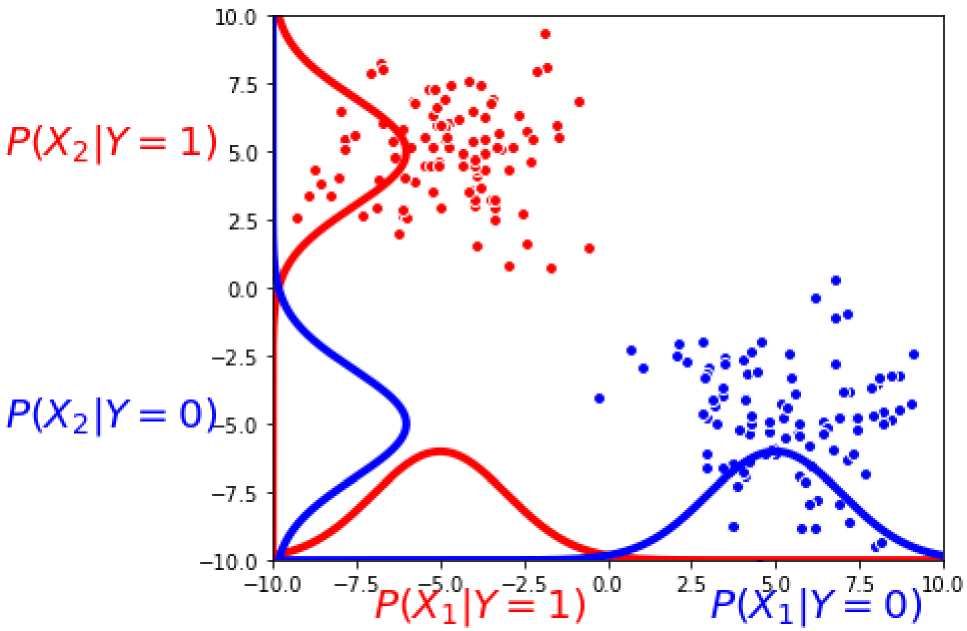
In [17]:

```
P_X1_1 = norm.pdf(x1,mu_1_1,sigma_1_1)
P_X2_1 = norm.pdf(x1,mu_2_1,sigma_2_1)
P_X1_0 = norm.pdf(x1,mu_1_0,sigma_1_0)
P_X2_0 = norm.pdf(x1,mu_2_0,sigma_2_0)

plt.figure(figsize=(6,6))
plt.scatter(X_positive[:, 0], X_positive[:, 1],facecolors='r', edgecolors='w')
plt.scatter(X_negative[:, 0], X_negative[:, 1],facecolors='b', edgecolors='w')
lim_plot = 10
plt.plot(x1,P_X1_1*2*lim_plot-lim_plot,'r',linewidth=4)
plt.text(-7, -12, r'$P(X_1|Y=1)$', color = 'red',fontsize=20)
plt.plot(x1,P_X1_0*2*lim_plot-lim_plot,'b',linewidth=4)
plt.text(3, -12, r'$P(X_1|Y=0)$', color = 'blue',fontsize=20)
plt.axis([-10,10,-10,10],'equal');
```



In [18]:
```
plt.figure(figsize=(6,5))
plt.scatter(X_positive[:, 0], X_positive[:, 1],facecolors='r', edgecolors='w')
plt.scatter(X_negative[:, 0], X_negative[:, 1],facecolors='b', edgecolors='w')

lim_plot = 10
plt.plot(x1,P_X1_1*2*lim_plot-lim_plot,'r',linewidth=4)
plt.text(-7, -12, r'$P(X_1|Y=1)$', color = 'red',fontsize=20)
plt.plot(x1,P_X1_0*2*lim_plot-lim_plot,'b',linewidth=4)
plt.text(3, -12, r'$P(X_1|Y=0)$', color = 'blue',fontsize=20)
plt.plot(P_X2_1*2*lim_plot-lim_plot,x1,'r',linewidth=4)
plt.text(-18,5,  r'$P(X_2|Y=1)$', color = 'red',fontsize=20)
plt.plot(P_X2_0*2*lim_plot-lim_plot,x1,'b',linewidth=4)
plt.text(-18,-5,  r'$P(X_2|Y=0)$', color = 'blue',fontsize=20)
plt.axis([-lim_plot,lim_plot,-lim_plot,lim_plot],'equal')
```

Out[18]:
```
[-10, 10, -10, 10]
```

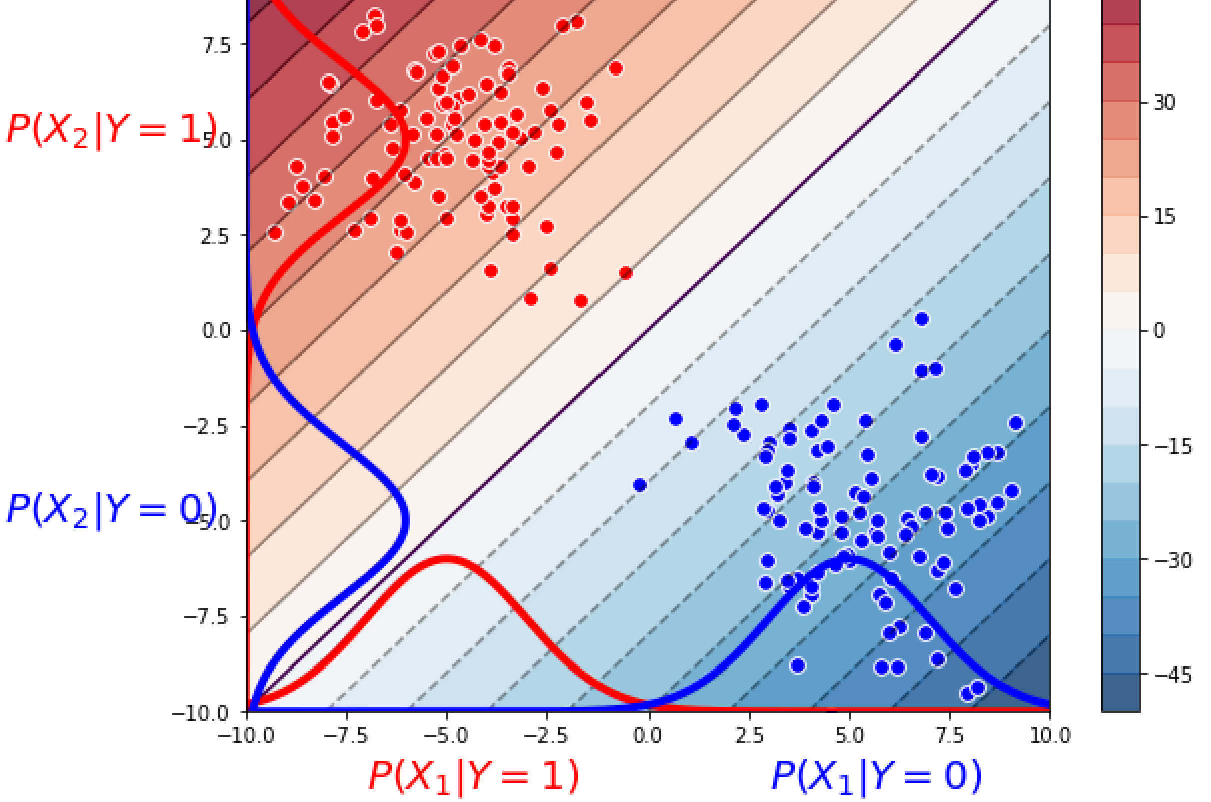


In [19]:

```python
# Compute log( P(Y=1|X)/ P(Y =0|X))
# as log( P(Y=1)P(X1|Y=1)P(X2|Y=1)  / P(Y =0|X))P(X1|Y=0)P(X2|Y=0) )
# Using the real parameters. Usually, we have to estimate these!
X1,X2 = np.meshgrid(x1, x2)
def ratio_log(X1,X2):
    pY0 =0.5; pY1 = 1- pY0
    pY1pXY1 = pY1*norm.pdf(X1,mu_1_1,sigma_1_1)*norm.pdf(X2,mu_2_1,sigma_2_1)
    pY0pXY0 = pY0*norm.pdf(X1,mu_1_0,sigma_1_0)*norm.pdf(X2,mu_2_0,sigma_2_0)
    return np.log(pY1pXY1/pY0pXY0)
fX = ratio_log(X1,X2)
```

In [20]:

```
plt.figure(figsize=(9,7))

# plot contour plot
cs = plt.contourf(X1, X2, fX,20,cmap='RdBu_r',alpha=0.8);
plt.colorbar()
contours = plt.contour(cs, colors='k',alpha=0.4) # this redraws the lines in black
plt.contour(contours,levels=[0],linewidth=5)   # this makes the 0 line wider


# previous stuff
plt.scatter(X_positive[:, 0], X_positive[:, 1],facecolors='r', edgecolors='w',s=60
)
plt.scatter(X_negative[:, 0], X_negative[:, 1],facecolors='b', edgecolors='w',s=60
)


lim_plot = 10

plt.plot(x1,P_X1_1*2*lim_plot-lim_plot,'r',linewidth=4)
plt.text(-7, -12, r'$P(X_1|Y=1)$', color = 'red',fontsize=20)

plt.plot(x1,P_X1_0*2*lim_plot-lim_plot,'b',linewidth=4)
plt.text(3, -12, r'$P(X_1|Y=0)$', color = 'blue',fontsize=20)

plt.plot(P_X2_1*2*lim_plot-lim_plot,x1,'r',linewidth=4)
plt.text(-16,5,  r'$P(X_2|Y=1)$', color = 'red',fontsize=20)

plt.plot(P_X2_0*2*lim_plot-lim_plot,x1,'b',linewidth=4)
plt.text(-16,-5,  r'$P(X_2|Y=0)$', color = 'blue',fontsize=20)



plt.axis([-lim_plot,lim_plot,-lim_plot,lim_plot],'equal');
```



$P(X_2|Y=1)$
$P(X_2|Y=0)$
$P(X_1|Y=1)$
$P(X_1|Y=0)$
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
30
15
0
−15
−30
−45

In [21]:

```
def ratio_log_updated (X1,X2,params): # just as an example, not used here
    pY0 =0.5; pY1 = 1- pY0
    pY1pXY1 = pY1*norm.pdf(X1,params['mu_1_1'],params['sigma_1_1'])*norm.pdf(X2,pa
rams['mu_2_1'],params['sigma_2_1'])
    pY0pXY0 = pY0*norm.pdf(X1,params['mu_1_0'],params['sigma_1_0'])*norm.pdf(X2,pa
rams['mu_2_0'],params['sigma_2_0'])
    return np.log(pY1pXY1/pY0pXY0)
```

In [23]:

```python
def plot_GNB(X_positive,X_negative,params):
    pY0 =0.5; pY1 = 1- pY0
    P_X1_1 = norm.pdf(x1,params['mu_1_1'],params['sigma_1_1'])
    P_X2_1 = norm.pdf(x1,params['mu_2_1'],params['sigma_2_1'])
    P_X1_0 = norm.pdf(x1,params['mu_1_0'],params['sigma_1_0'])
    P_X2_0 = norm.pdf(x1,params['mu_2_0'],params['sigma_2_0'])
    X1,X2 = np.meshgrid(x1, x2)
    # faster way to compute the log ratio, or can use fX = ratio_log_updated(X1,X2,params)
    fX = np.log(pY1/pY0) + np.log(P_X1_1.reshape([1000,1]).dot(P_X2_1.reshape([1,1000]))/
                                  P_X1_0.reshape([1000,1]).dot(P_X2_0.reshape([1,1000])))
    plt.figure(figsize=(10,8))
    # plot contour plot
    cs = plt.contourf(X1, X2, fX,20,cmap='RdBu_r',alpha=0.8);
    plt.colorbar()
    contours = plt.contour(cs, colors='k',alpha=0.4)
    plt.contour(contours,levels=[0],linewidth=5)
    plt.scatter(X_positive[:, 0],X_positive[:, 1],facecolors='r',edgecolors='w',s=60)
    plt.scatter(X_negative[:, 0],X_negative[:, 1],facecolors='b',edgecolors='w',s=60)
    lim_plot = 10
    plt.plot(x1,P_X1_1*2*lim_plot-lim_plot,'r',linewidth=4)
    plt.text(-7, -12, r'$P(X_1|Y=1)$', color = 'red',fontsize=20)
    plt.plot(x1,P_X1_0*2*lim_plot-lim_plot,'b',linewidth=4)
    plt.text(3, -12, r'$P(X_1|Y=0)$', color = 'blue',fontsize=20)
    plt.plot(P_X2_1*2*lim_plot-lim_plot,x1,'r',linewidth=4)
    plt.text(-16,5,  r'$P(X_2|Y=1)$', color = 'red',fontsize=20)
    plt.plot(P_X2_0*2*lim_plot-lim_plot,x1,'b',linewidth=4)
    plt.text(-16,-5,  r'$P(X_2|Y=0)$', color = 'blue',fontsize=20)
    plt.axis([-lim_plot,lim_plot,-lim_plot,lim_plot],'equal')
```

The features X_1 and X_2 in the simulation where conditionally independent

What if:

- we make them dependent (use a non-diagonal covariance matrix to sample multivariate gaussian)
- We still use conditional independence as an assumption for GNB

1st: case where same variance

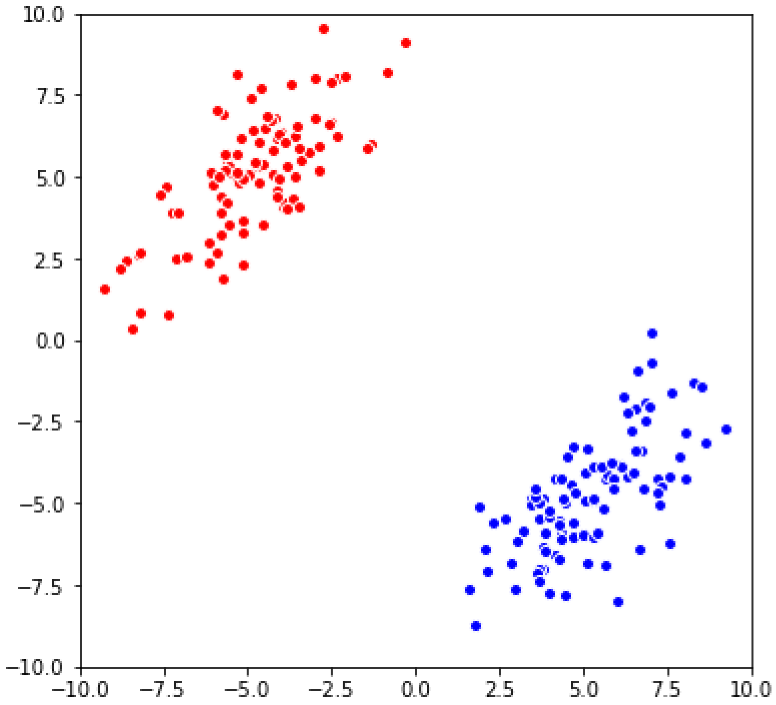
In [ ]:

```
from scipy.stats import multivariate_normal

# Same param as before
mu_1_1 = -5; sigma_1_1 = 2; mu_2_1 = 5; sigma_2_1 = 2
mu_1_0 = 5; sigma_1_0 = 2; mu_2_0 = -5; sigma_2_0 = 2

cov_positive = np.array([[sigma_1_1**2,3], [3,sigma_2_1**2]] )
cov_negative = np.array([[sigma_1_0**2,3], [3,sigma_2_0**2]] )

print(cov_positive)

# Sample data from these distributions
X_positive = multivariate_normal.rvs(mean=[mu_1_1,mu_2_1],cov=cov_positive,size=(1
00))
X_negative = multivariate_normal.rvs(mean=[mu_1_0,mu_2_0],cov=cov_negative,size=(1
00))
```

In [20]:

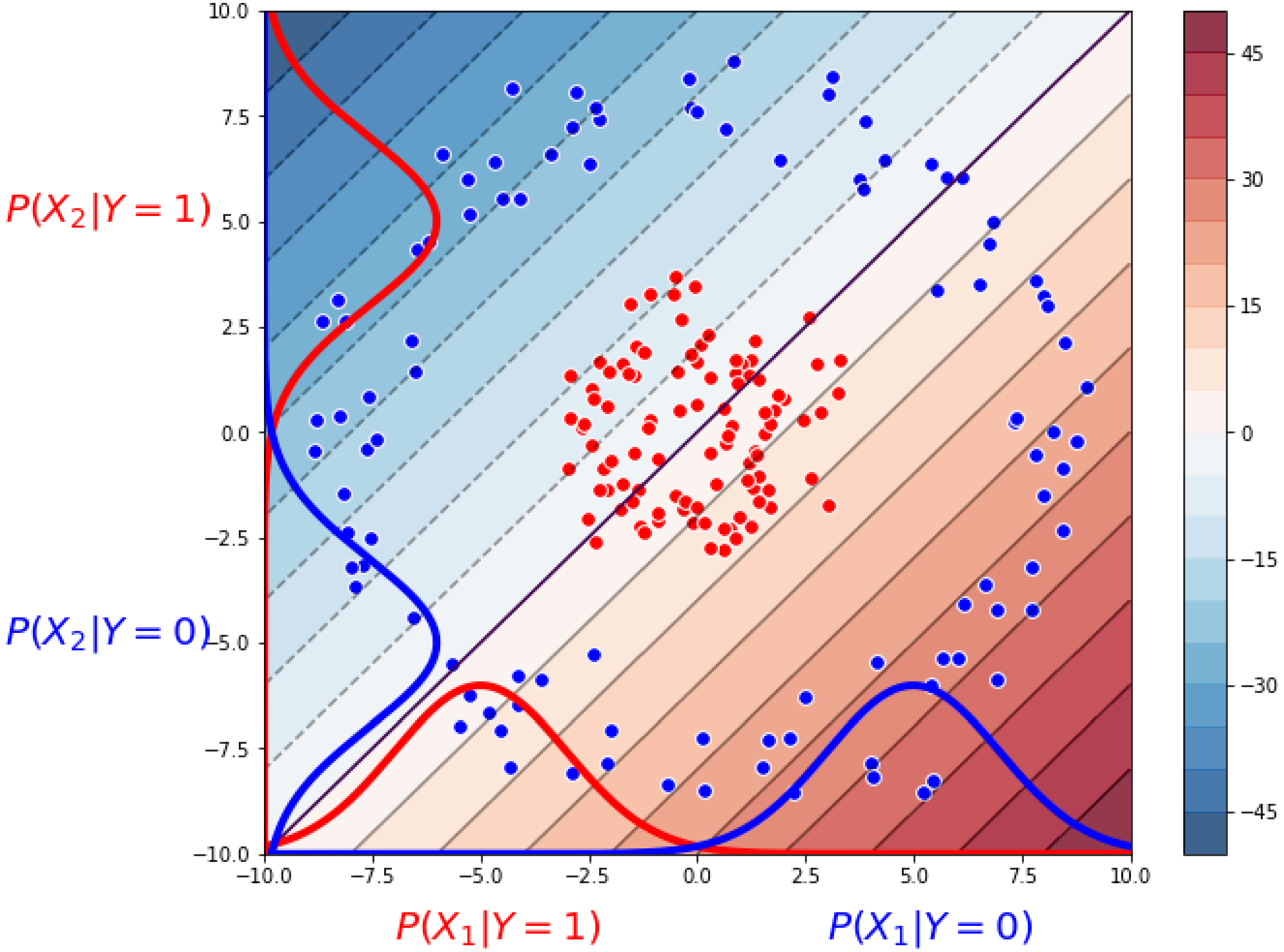
```
plt.figure(figsize=(6,6))
plt.scatter(X_positive[:, 0], X_positive[:, 1],facecolors='r', edgecolors='w')
plt.scatter(X_negative[:, 0], X_negative[:, 1],facecolors='b', edgecolors='w')
plt.axis([-10,10,-10,10],'equal');
```

```
[[4 3]
 [3 4]]
```



In [39]:

```
# Assume I perfectly estimate the parameters (not true for limited data!)
params = dict(mu_1_1 = -5, sigma_1_1 = 2,mu_2_1 = 5, sigma_2_1 = 2,
              mu_1_0 = 5, sigma_1_0 = 2,mu_2_0 = -5, sigma_2_0 = 2)

plot_GNB(X_positive,X_negative,params)
```



In [22]:

```python
# Estimate

mu_1_1, mu_2_1 = np.mean(X_positive,axis=0)
mu_1_0, mu_2_0 = np.mean(X_negative,axis=0)

# Same Variance!

sigma_1_1, sigma_2_1 = np.std(X_positive,axis=0)
sigma_1_0, sigma_2_0 = np.std(X_negative,axis=0)
print(sigma_1_1, sigma_2_1)
print(sigma_1_0, sigma_2_0)
```

```
1.7556505128707445 1.830266323858797
1.8626233472002263 1.9122394301165095
```

Is GNB a linear separator?

- It depends on whether we allow it to learn different standard deviations for each class

Decision rule:

$$\ln \frac{P(Y=1|X_1 \ldots X_d)}{P(Y=0|X_1 \ldots X_d)} = \ln \frac{P(Y=1)}{P(Y=0)} + \sum_i \ln \frac{P(X_i|Y=1)}{P(X_i|Y=0)} \qquad > \text{ or } < 0?$$

If X_is are $\mathcal{N}(\mu_{ik}, \sigma_{ik})$:

$$p(X_i = x | Y = y_k) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}} \exp\Big(-\frac{1}{2}\frac{(x_i - \mu_{ik})^2}{\sigma_{ik}^2}\Big)$$

$$\begin{aligned} \ln \frac{P(Y=1|X_1 \ldots X_d)}{P(Y=0|X_1 \ldots X_d)} &= \ln \frac{P(Y=1)}{P(Y=0)} + \sum_i \ln \frac{P(X_i|Y=1)}{P(X_i|Y=0)} \\ &= \ln \frac{P(Y=1)}{P(Y=0)} + \sum_i \ln \frac{\sigma_{i0}}{\sigma_{i1}} + G(X) \end{aligned}$$

$$G(X) = \sum_i \ln \exp\Big(- \frac{1}{2}\frac{(x_i - \mu_{i1})^2}{\sigma_{i1}^2} + \frac{1}{2}\frac{(x_i - \mu_{i0})^2}{\sigma_{i0}^2} \Big)$$

$$= -\frac{1}{2} \sum_i \Big(x_i^2 (\frac{1}{\sigma_{i1}^2} - \frac{1}{\sigma_{i0}^2}) - 2x_i (\frac{\mu_{i1}}{\sigma_{i1}^2} - \frac{\mu_{i0}}{\sigma_{i0}^2}) + (\frac{\mu_{i1}^2}{\sigma_{i1}^2} - \frac{\mu_{i0}^2}{\sigma_{i0}^2}) \Big)$$

What happens if we force $\sigma_{i0} = \sigma_{i1}$?

- We get a linear decision boundary. Otherwise, it's a quadratic decision boundary.

In [ ]:

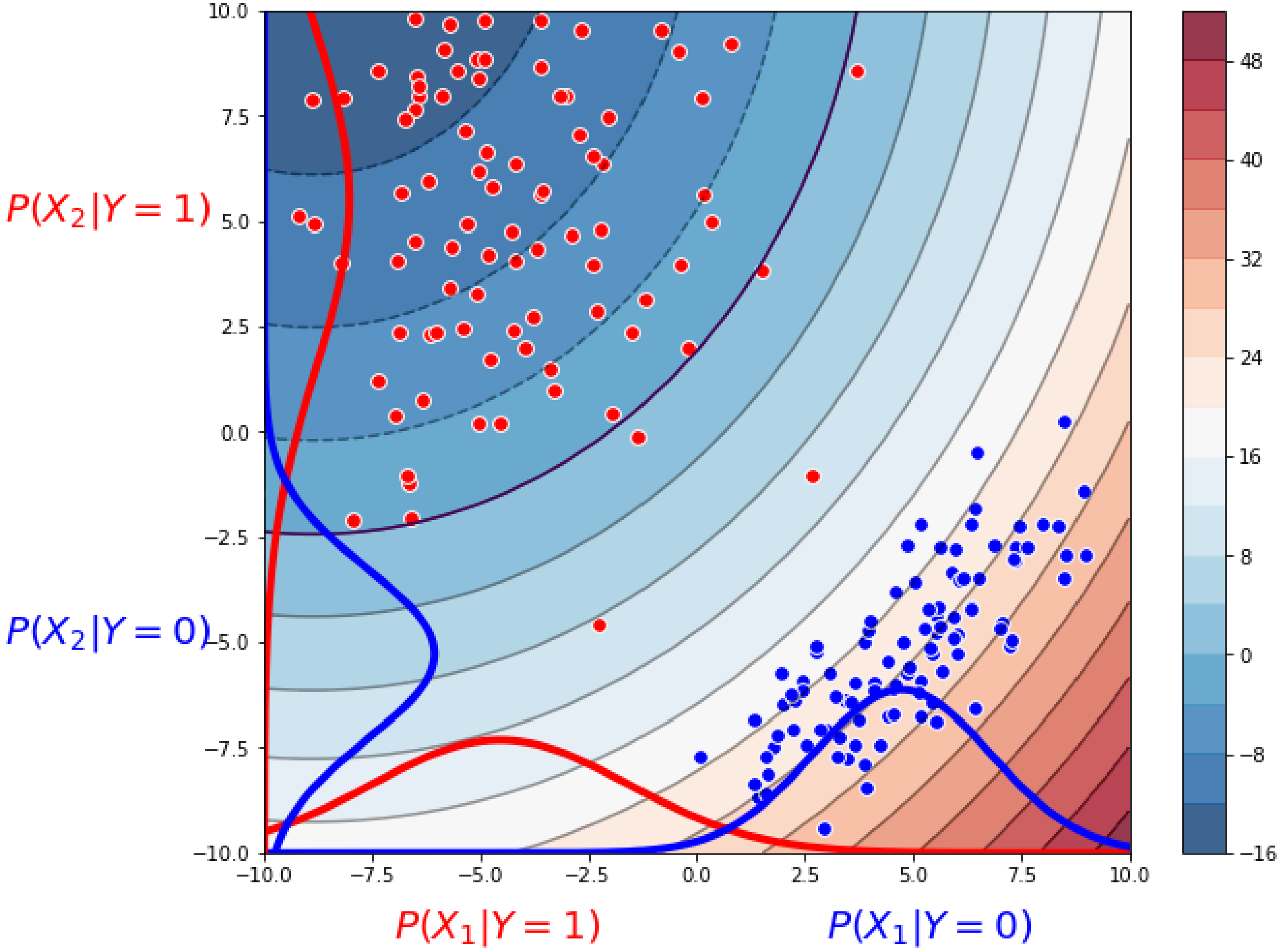
```
# Same param as before
mu_1_1 = -5; sigma_1_1 = 2
mu_2_1 = 5; sigma_2_1 = 2
mu_1_0 = 5; sigma_1_0 = 2
mu_2_0 = -5; sigma_2_0 = 2

cov_positive = np.array([[sigma_1_1**2,3], [3,sigma_2_1**2]] )
cov_negative = np.array([[sigma_1_0**2,3], [3,sigma_2_0**2]] )

# Sample data from these distributions
X_positive = multivariate_normal.rvs(mean=[mu_1_1,mu_2_1], cov=cov_positive, size
= (100))
X_negative = multivariate_normal.rvs(mean=[mu_1_0,mu_2_0], cov=cov_negative, size
= (100))
```

In [28]:

```
params = dict()
# Estimate - Different variance because of limited sample size
params['mu_1_1'], params['mu_2_1'] = np.mean(X_positive,axis=0)
params['sigma_1_1'], params['sigma_2_1'] = np.std(X_positive,axis=0)
params['mu_1_0'], params['mu_2_0'] = np.mean(X_negative,axis=0)
params['sigma_1_0'], params['sigma_2_0'] = np.std(X_negative,axis=0)
```

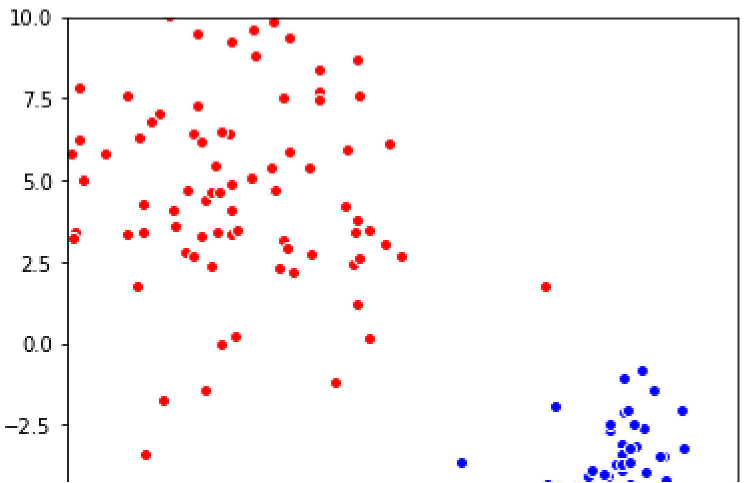
In [29]:

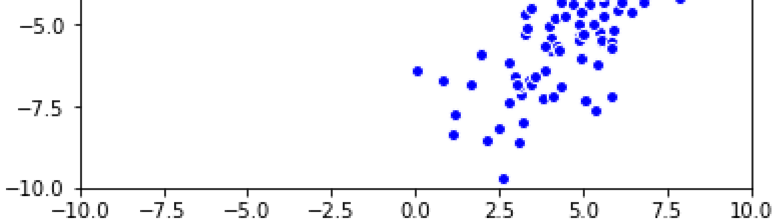
```
plot_GNB(X_positive,X_negative,params)
```



In [40]:
```
# Let's set up another example in which the variances are actually different
mu_1_1 = -5; sigma_1_1 = 3
mu_2_1 = 5; sigma_2_1 = 4
mu_1_0 = 5; sigma_1_0 = 2
mu_2_0 = -5; sigma_2_0 = 2

cov_positive = np.array([[sigma_1_1**2,3], [3,sigma_2_1**2]] )
cov_negative = np.array([[sigma_1_0**2,3], [3,sigma_2_0**2]] )

# Sample data from these distributions
X_positive = multivariate_normal.rvs(mean=[mu_1_1,mu_2_1],cov=cov_positive,size=(1
00))
X_negative = multivariate_normal.rvs(mean=[mu_1_0,mu_2_0],cov=cov_negative,size=(1
00))

plt.figure(figsize=(6,6))

plt.scatter(X_positive[:, 0], X_positive[:, 1],facecolors='r', edgecolors='w')
plt.scatter(X_negative[:, 0], X_negative[:, 1],facecolors='b', edgecolors='w')
plt.axis([-10,10,-10,10],'equal')
```

Out[40]:
```
[-10, 10, -10, 10]
```



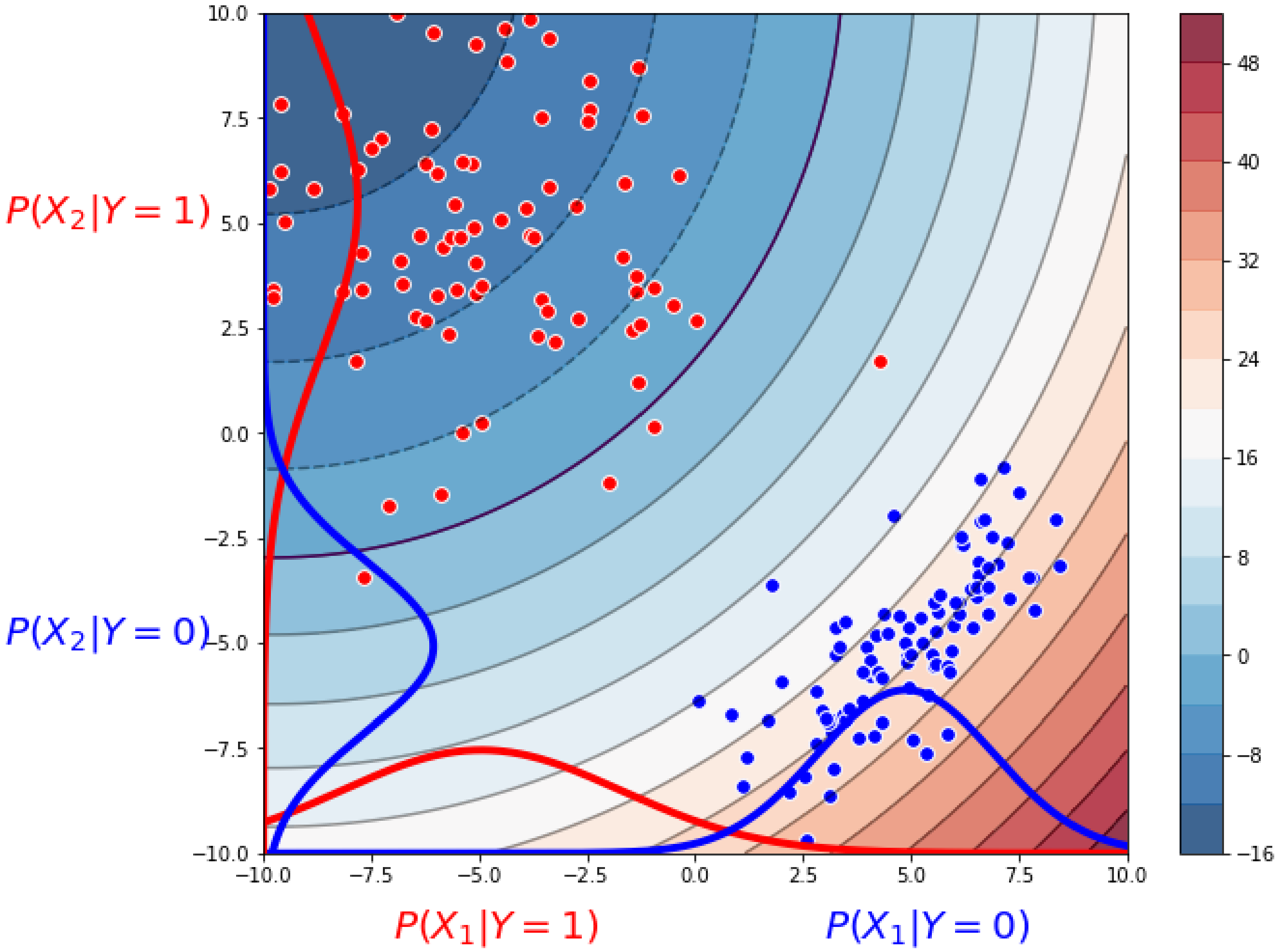

−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

In [41]:

```
params = dict()
# Estimate - Different variance
params['mu_1_1'], params['mu_2_1'] = np.mean(X_positive,axis=0)
params['sigma_1_1'], params['sigma_2_1'] = np.std(X_positive,axis=0)
params['mu_1_0'], params['mu_2_0'] = np.mean(X_negative,axis=0)
params['sigma_1_0'], params['sigma_2_0'] = np.std(X_negative,axis=0)
```

In [42]:

```
plot_GNB(X_positive,X_negative,params)
```



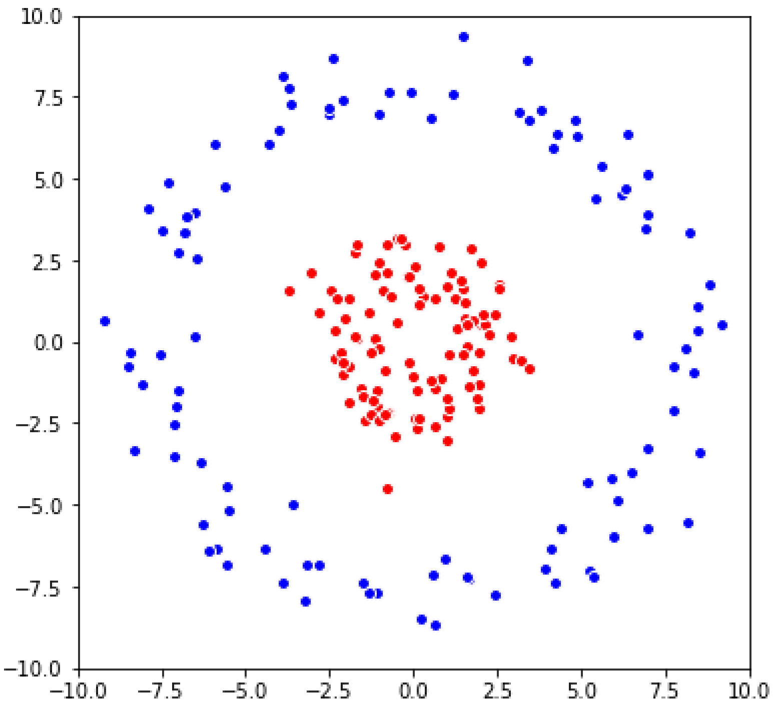
In [43]:

```
from sklearn import datasets

plt.figure(figsize=(5,5))
X, y = datasets.make_circles(n_samples=200, factor=.25,noise=.1)

# scale
X_positive = X[y==1]*8
X_negative = X[y==0]*8
```

```
<Figure size 360x360 with 0 Axes>
```

In [44]:
```
plt.figure(figsize=(6,6))

plt.scatter(X_positive[:, 0], X_positive[:, 1],facecolors='r', edgecolors='w')
plt.scatter(X_negative[:, 0], X_negative[:, 1],facecolors='b', edgecolors='w')
plt.axis([-10,10,-10,10],'equal')
```

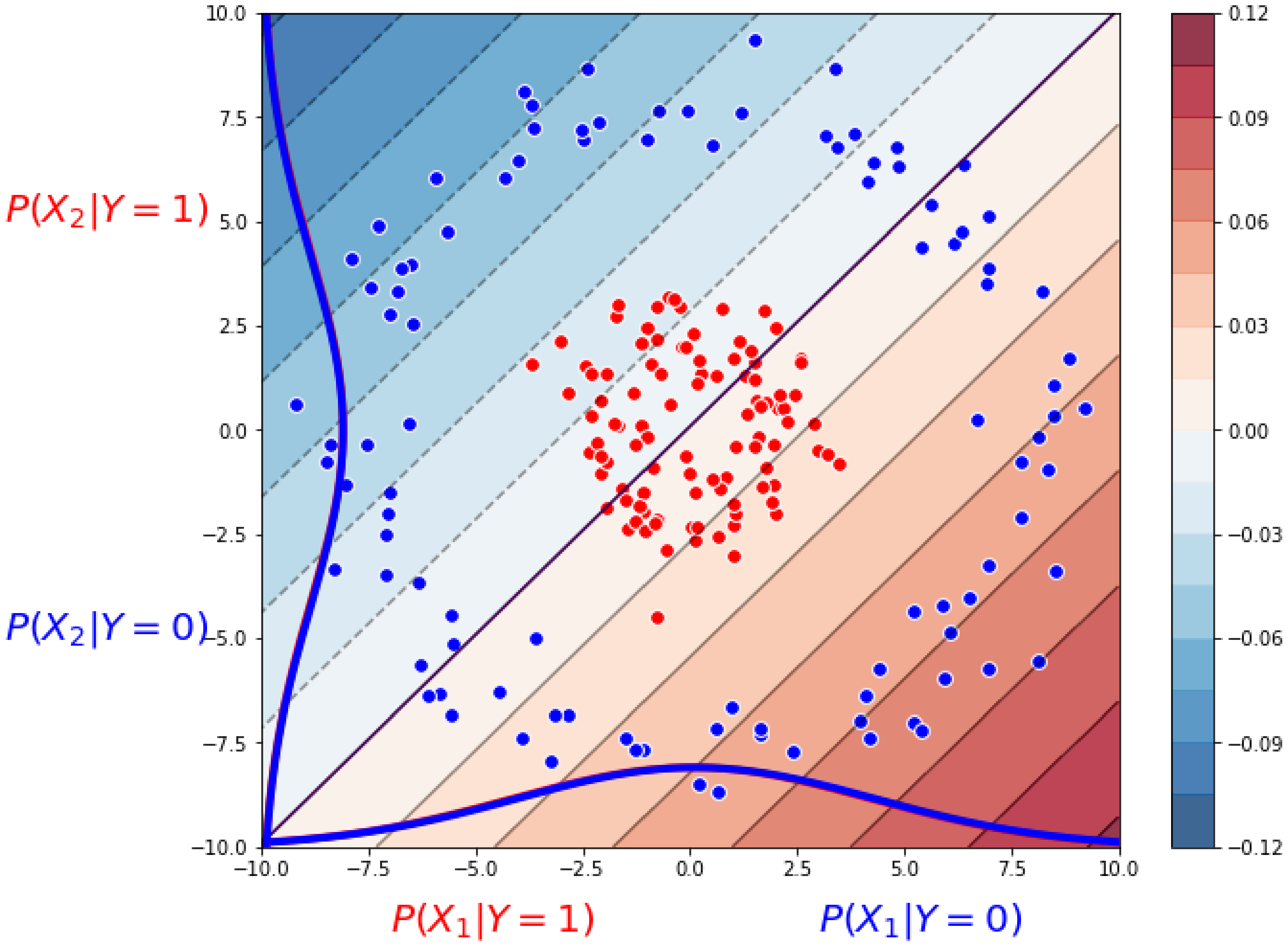
Out[44]:
```
[-10, 10, -10, 10]
```



In [45]:

```
params = dict()
# Artificially force same variances
params['mu_1_1'], params['mu_2_1'] = np.mean(X_positive,axis=0)
params['sigma_1_1'], params['sigma_2_1'] = np.std(np.vstack([X_positive,X_negative
]),axis=0)
params['mu_1_0'], params['mu_2_0'] = np.mean(X_negative,axis=0)
params['sigma_1_0'], params['sigma_2_0'] = np.std(np.vstack([X_positive,X_negative
]),axis=0)
```

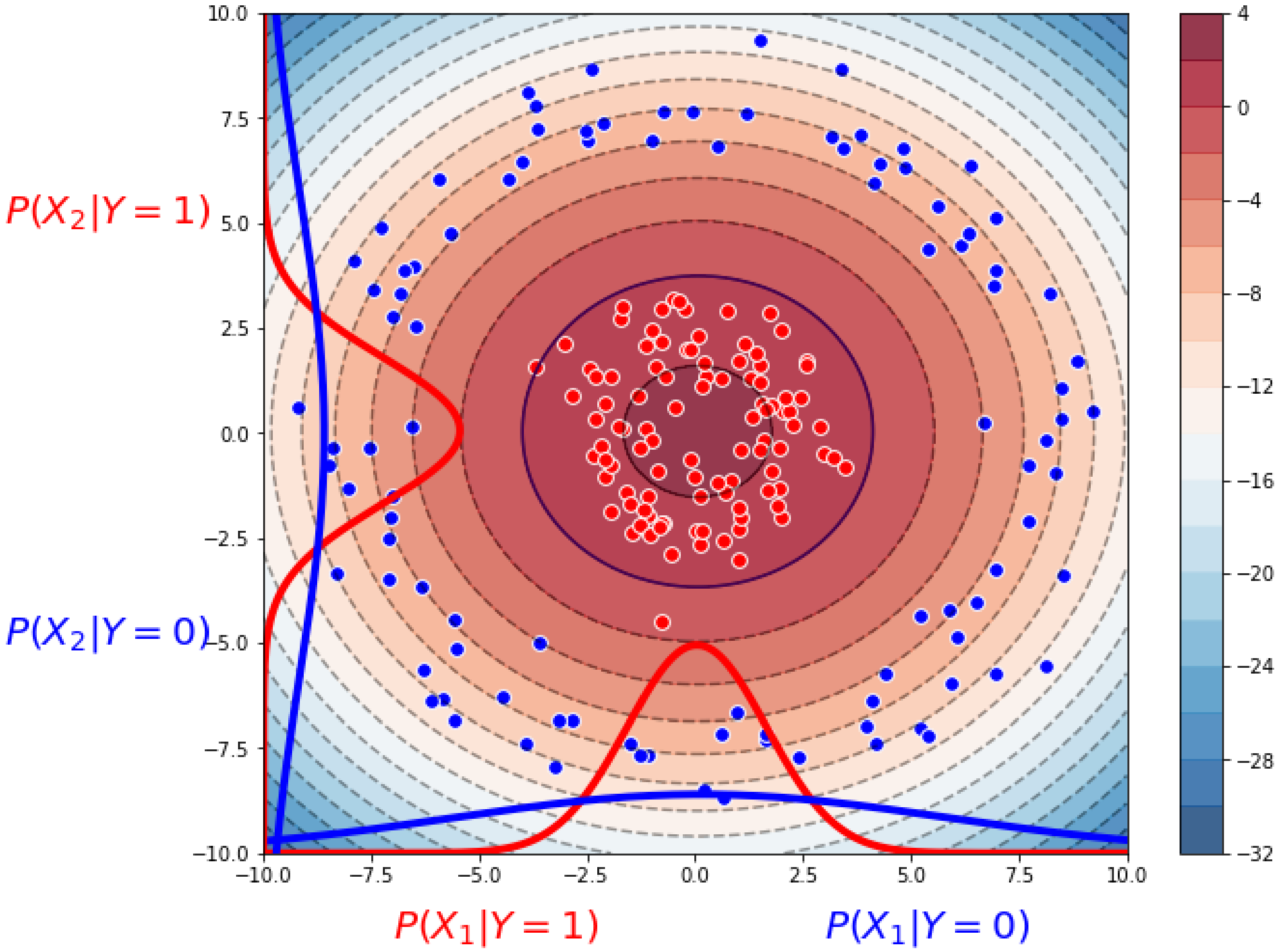
```
In [46]: plot_GNB(X_positive,X_negative,params)
```



In [47]:

```
params = dict()
# Estimate - Different variance
params['mu_1_1'], params['mu_2_1'] = np.mean(X_positive,axis=0)
params['sigma_1_1'], params['sigma_2_1'] = np.std(X_positive,axis=0)
params['mu_1_0'], params['mu_2_0'] = np.mean(X_negative,axis=0)
params['sigma_1_0'], params['sigma_2_0'] = np.std(X_negative,axis=0)
```

In [48]:
```
plot_GNB(X_positive,X_negative,params)
```



The last example is a case where the conditional independence assumption is incorrect

- but GNB does very well

What you should know

Naïve Bayes classifier

- What’s the assumption
- Why we use it
- How do we learn it
- The different observations we made about it
- Why is Bayesian estimation important